Viper 🐍

An amalgamation of all our favorite language quirks.

A hosted copy of this manual may be found here.

Authors:

- Mustafa Eyceoz (me2680)
- Tommy Gomez (tjk2132)
- Trey Gilliland (jlg2266)
- Matthew Ottomano (mro2120)
- Raghav Mecheri (rm3614)

0 Contents 📌

1. Selector Statements
    1. if
    2. if/elif/else
2. Iterator Statements
    1. for
    2. while
3. Jump Statements
    1. skip
    2. abort
4. Function Statement
    1. Arrow Functions
    2. [Attribute Calls]

6. Expressions 🖥️
    1. Truth-Value Expressions
    2. Guard Expressions
7. Operators ➗
    1. Unary Operators
        1. ! (NOT)
        2. ++ (increment)
        3. -- (decrement)
    2. Binary Operators
        1. + (addition)
        2. - (subtraction)
        3. * (multiplication)
        4. / (division)
        5. % (modulo)
    3. Comparative Operators
        1. > (greater than)
        2. >= (greater than or equal to)
        3. < (less than)
        4. <= (less than or equal to)
        5. == (equals)
        6. != (not equals)
    4. Logical Operators
        1. and
        2. or
    5. Variable Operators
        1. += (quick add)
        2. -= (quick subtract)
        3. *= (quick multiply)
        4. /= (quick divide)
        5. = (assign)
        6. ? : (ternary operators)
    6. Precedence of Operators
        1. Unary
        2. Binary
        3. Comparative
        4. Logical
        5. Variable

1 Overview 🚁

Viper is a statically-typed imperative programming language that incorporates powerful functionality into a clean syntax. By requiring users to declare the types of functions and variables, Viper benefits from the safety mechanisms and increased efficiency of type checking. It also includes useful features like pattern matching, arrow functions, and an intuitive standard library. See the following sections for a complete introduction to the language.

1.1 Background

Modern day scripting languages like Python and Javascript are incredibly convenient. They make it incredibly easy to write short, readable code that makes both prototyping and collaboration a breeze. The issue, however, is that this level of convenience and accesibility comes at a cost: computational efficiency. While simple, forgiving, dynamically-typed languages like the previously mentioned are useful, it requires one to forgo the traditional compiler, and instead use an interpreter when attempting to execute code. The process of simultaneously translating into machine code and executing takes orders of magnitude more time then executing a pre-compiled piece of code, and even the efforts of just-in-time compilers like PyPy have been unsuccessful in completely bridging the gap. To achieve similar efficiency to languages like C and C++, a proper compiler is a necessity.

We set into this project with exactly that in mind: come up with a language that retains all of the simplicity, ease-of-use, and functionality of modern scripting languages, while introducing a proper static typing system to allow for efficient compilation. We wanted it to feel like you are writing in Python or Javascript, but then afterwards to feel like you are running a C program. With that in mind, we first thought about all the features we wanted to carry over from the dynamic scripting languages. First, a user has to be able to open a file and just start typing. The top-level should be a place where code is directly executed, without the need for any complex class-structure. Additionally, we would require a lot of syntactic sugar for different means of iteration, declaration, and assignment. We would also need to include the functionality of mechanics like arrow functions and ternaries, as well as standard data structures like lists and dictionaries. At this point, we also had some new ideas for features like pattern matching to replace nested ternaries and switches, and built-in index values for iterators. All together, these core ideas came to be what we now call Viper.

1.2 Related Work

Most features found in Viper derive from one or more of Python, Javascript, or C. While our scoping and execution rules match Python closest, Viper does not use whitespace to identify scoping, but instead uses the standard brace notation from the other two languages. Additionally, while more abstract than C (as memory allocation is handled for the user), Viper also requires static typing for all variables upon declaration. Operator precedence and applicative-order reduction is the same in Viper as one might expect from any of the aforementioned programming languages.

The one other reference worth mentioning is in relation to Viper's pattern-matching eyntax. While not exactly the same functionally, we got the idea to include this feature after extensive use of pattern

matching in OCaml. Additionally, OCaml was used in almost all components of the Viper compiler pipeline, with the exception of the standard library being written in C.

`1.3` Goals - To Preserve While Becoming Compilable:

`Accessibility, Readability`

Despite changing to a statically typed language, we want to ensure that Viper code is just as readable and easy to learn as Python code. The only added level of difficulty should be with declaring and tracking static types. Any user should be able to begin their coding journey with this language, and and user should be able to read and understand another user's work without three cyphers and a thesaurus.

`Prototyping/Writing Efficiency`

In addition to readability, writability is also important. We want users to be able to express their ideas quickly and effectively, with common-sense, intuitive syntax. We want to preserve the idea that the thought->prototype pipeline should be as quickly traversable as possible.

`Functionality`

Finally, we want to make sure that in Viper, users can still do all the things they need to. While this goal is more of a continuous process rather than a current guarantee, the initial release of Viper still has a lot of the features that make languages like Python more immediately advantageous than those like C without as many supported data structures and operations.

`1.4` How to Obtain and Use Viper

To obtain the Viper code repository, simply clone this repo: https://github.com/raghavmecheri/viper

- Once cloned, type `cd src && make && cd ..`
- Next, write some Viper code in a (filename).vp file (details on how to write Viper in next sections)
  - For an example, open a `test.vp` and inside write `print("hello world");`
- Running `src/viper.native test.vp` will output the llvm code if desired
- Running `./exec.sh test.vp` will generate three files:
  - `a.ll` = llvm code
  - `a.s` = assembly code
  - `a.exe` = executable for code
- NOTE: Using `exec.sh` will also run `a.exe` for you
- If you add `-v`, like: `./exec.sh test.vp -v`, you will also receive the llvm output `viper.native` would provide

↩️ Back to Contents 📌

`2` Lexical Conventions 📝

`2.1` Comments

Viper allows for multi-line comments that begin with an opening forward slash followed by a star (/*) and end with a closing backward slash followed by a star (*\). All content within the bounds of these symbols is ignored.

```
/* Single-line comments anyone? */

/* How about
multi-line?
*/
```

2.2 Identifiers

All user-defined identifiers (variable and function names) must begin with an ASCII letter and can contain any mix of ASCII letters and numbers.

Example valid identifiers:

```
lambda_bamba
pythonCython
RatGhav
V1P3RisTh3b3sT
```

Example invalid identifiers:

```
68vip
--!x
V*x
```

2.3 Reserved Keywords

Any Viper reserved keywords can not be used as user-defined identifiers. A list of reserved Viper keywords include:

```
/* control flow */
if else for while return skip abort panic in has

/* function and types */
func char int float bool nah string dict group

/* operators and literals */
and or is not true false
```

2.4 Scoping

Viper uses a pair of opening and closing curly brackets ({}) to represents a scope of statements within control-flow and function definitions. All statements within the scope must be followed by a semi-colon, but are not required to be on a new line or indentation as previous statements. The core of the control-flow statement following the keyword must be surrounded by parenthesis as well.

Example bracket scope usage:

```
func void foo() {
    print("bar");
}

count = 0
```

```
while (count < 10) {
    if (count % 2 == 0) {
        count += 1;
    }

    count += 1;
}
```

For more information on statements and scoping, see Section 5.

2.5 Literals

Literals are the values that primitive types within Viper take on within the source code.

Literals include:

- boolean
- char
- int
- float
- nah

2.5.1 Char Literals

Char literals represent a single ASCII character and expressed as a letter within single quotes. They also can represent escape sequences and special tokens such as '\t' and '\n'. These individual literals can be combined to make up a String when surrounded by double quotes. These character literals are always assigned to variables of the type *char*.

Examples of char literals:

```
'a'
'+'
'\n'
```

2.5.2 Int Literals

Int literals represent a whole decimal number as an integer and always takes on the *int* type.

Examples of int literals:

```
0
42
-70843
```

2.5.3 Float Literals

A float literal represents a decimal floating point number and always takes on the *float* data type. A float literal consists of a sequence of numbers representing the whole-number part, followed by an ASCII decimal point, followed by a sequence of numbers representing the decimal portion.

Examples of float literals:

```
123.45
-0.007
```

```
3.485
```

2.5.4 Boolean Literals

Boolean literals are used to indicate the truth value of an expression and are represented by the *bool* data type. The two boolean literals used by Viper are the keywords *true* and *false*.

2.5.5 String Literals

String literals are a sequence of chars surrounded by double quotes. These literals can be assigned to variables of the type *string*.

2.5.6 Nah Literals

The nah literal represents a reference to a null value and always takes on the nah type. This literal is represented by *nah* made from ASCII characters.

Examples of string literals:

```
"Stringy123"
"ratghav merch boi"
"H3sKell >>>"
```

2.5.7 List Literals

All list literals consist of an opening square bracket, a sequence of objects/values all of the same type sepereated by commas, and a closing square bracket. List literals must be assigned to variables of the type *list* wrapping the same type the array literal contains. List literals can contain array list within themselves, leading to multi-dimensional lists.

Examples of valid list literals:

```
[1, 2, 3]
["a", "b", "c"]
[[1], [10]]
```

Examples of invalid list literals:

```
[1, 'a', "3"]
[nah, "beach"]
[1, [5, 9]]
```

↩️ Back to Contents 📌

3 Data Types💾

Viper supports the same primitive and higher-order data types as many modern languages. Primitive types are supported natively, while higher-order types are implemented in Viper's Standard Library 📚.

3.1 Primitive Data Types

The six primitive types supported by Viper are `char`, `int`, `float`, `bool`, `string` and `nah`. The table below summarizes their properties and declarations, with more details in the following sections.

Primitive Type	Size	Description	Declaration/Usage

`char`	1 byte	Represents single ASCII characters	`char a = 'a';`
			`char c = 'b' + 1;`
			`char newline = '\n';`

`int`	4 bytes	Stores signed integer values	`int pos = 12;`
			`int neg = -980;`
			`int sum = 4 + 5;`

`float`	4 bytes	Stores signed floating-point numbers	`float pos = 3.2;`
			`float neg = -29.7;`
			`float dec = 0.003;`
			`float whole_num = 2.0;`

`bool`	1 byte	Stores either `true` or `false`	`bool t = true;`
			`bool f = false;`
			`bool falsy = t and f;`

`string`	varies	Stores a sequence of `char`s representing a word	`string s = "yeehaw"` `string name = "mro";`

`nah`	1 byte	Viper's `null` value	`int nil = nah;`
			`char empt = nah;`
			`return nah;`

3.1.1 `char`

`char` is the type that represents single ASCII characters. In Viper, a `char` is represented as an ASCII character enclosed in single quotes. Special characters, like the newline and tab characters, are defined with an escape backslash (`'\n'` and `'\t'`, respectively). Each `char` behaves like an `int` in that it takes on the decimal value of its assigned ASCII character. Therefore, numerical operations that are valid for integers are also valid for `char`s.

3.1.2 `int`

`int`s represent signed integer values. The minimum value of an `int` is -2^{31}, and the maximum value is $2^{31} - 1$. Negative integer values must be defined with a preceding minus (-) symbol, but positive integer values cannot be defined with a preceding plus (+) symbol.

3.1.3 `float`

`float`s represent signed floating-point numbers. To define a `float` at least one digit must precede a decimal point (.), and at least one digit must follow. For example, `.8` and `8.` are invalid, and result in syntax errors. These values are correctly defined as `0.8` and `1.0`, with padding zeroes to ensure that there is a least one digit on each side of the decimal point.

3.1.4 `bool`

`bool`s hold one of the two Boolean values: `true` or `false`. Expressions using the logical `and`, logical `or`, and equality operators are evaluated to `bool`s. For example, the expression `(1 < 2) and ('c' ==`

`'c')` evaluates to a `bool` with value `true`. Additionally, specific values of each primitive type evaluate to certain `bool` values. See the table below for details (note that `nah` always evaluates to `false`).

Primitive Type	`true` values	`false` values
`char`	All `char`s but `'\0'` and `''`	`'\0'` and `''`
`int`	$[-2^{31}, -1]$, $[1, 2^{31} - 1]$	0
`float`	All `float`s but 0.0	0.0
`bool`	`true`	`false`
`string`	All non-empty `string`s	`""`
`nah`	n/a	`nah`

3.1.5 `string`

The `string` type of Viper is implented as a `list` of `char`s. `string`s are defined with the standard double quote notation.

```
string name = "Ghav";
```

Internally, a `string` is stored as a sequence of defined chars, followed by the null terminal character `'\0'`. The `string` "rat" is internally `['r', 'a', 't', '\0']`.

3.1.6 `nah`

`nah` is Viper's `null` value. It can be used to initialize any other data type, and is a valid return value for any function, regardless of the expected return type. Functions with no return value are declared with type `nah`.

3.2 Higher-Order Data Types

Viper also supports various higher-order data types, including `list`, `group`, and `dict`. More details can be found in the Standard Library section.

Type	Description	Declaration/Usage
`list`	Ordered lists of any type	`int[0] array; /* Empty list */` `float[] scores = [9.7, 8.2];`

`dict`	Key-value pairs with random access	`[int: int] pos; /* Empty */`
		`[string: (string, int)] items = [`
		`"milk": ["dairy", 5],`
		`"apple": ["fruit", 3] ];`

3.2.1 `list`

Like many languages, Viper supports random access `list`s of any data type. A `list` is defined by specifying a non-`list` data type, followed by at least one set of square brackets (`[]`). Multi-dimesional lists can be created with additional sets of square brackets. `list`s have fixed types, and can be created in the following ways:

```
/* 0. Empty lists: */
int[] dust;
float[][] edges;

/* 1. Size given implicitly by the length of the list literal: */
string[] cheese = ["chewy", "bendy", "wiggly"];
bool[][] outcomes = [[false, false], [false, true]];

/* 2. Size given implicityly by copy construction */
string[] glizzy = cheese;
```

`list`s can be accessed and modified directly by specifying indices in square brackets. Indices are integers in the range [0, length - 1). Attempting to access or modify an index outside this range throw errors.

```
int[] nums = [4, 0, 8];
nums[2] = nums[1];   /* Sets the last element to 0 */
nums[1] = 2;         /* Sets the middle element to 2 */

int error = nums[3]; /* Throws an error */
```

3.2.3 `dict`

A `dict` is a mapping of key-value pairs. The types of both keys and values must be specified at creation, and keys must be unique. `dict` literals are defined with square brackets (`[]`), in which a colon (`:`) separates key and values, and commas separate key-value pairs.

```
[int: string] map = [1: "one", 2: "two", 3: "three"]
```

`dict`s can be accessed and modified similarly to `list`s. Instead of using indices, however, `dict`s only accept key values. Attempting to use a key of an unexpected type, or using a key with no mapped value will result in an error.

```
/* Note: nested dicts */
[char: [string: int]] wordmap = [
        'a': ["add": 2, "and": 3],
        'b': ["blob": 1, "bap": 14],
        'd': ["doink": 1]];
[string: int] b_words = wordmap['b']; /* Retrieves ["blob": 1, "bap": 14] */
b_words["bing"] = 4; /* Adds key-value pair ["bing": 4] to b_words */
int no_no = b_words["balloon"]; /* Error: b_words has no key "balloon" */
int bad_idea = wordmap["a"]; /* Error: key type is char, not string */
```

Empty dicts can also be declared.

```
[string: int] rat = [];
```

↩️ Back to Contents 📌

4 Type System 🗃️

Viper utilizes a static typing system to benefit from the provided type safety and optimizations of a staticly typed compiled language.

4.1 Explicit Types

Viper requires explicit user-specified types for variable declarations, function parameters, control flow, and return types in function definitions. An explicit type is required when new variables, placeholders, and parameters are created and need a type to be referenced against.

Variable intialization and assignment:

```
string wow;
char x = 'y';
wow = "doge"; /* Note: not required with assignment when type of identifier has been initialized */
```

Function definitions:

```
int func incrementer(int x) {
    x += 1;
}

int func sum (int c) => c + c;
```

Control-flow:

```
for (int i = 0; i <= 10; i++) {
    print(i);
}

int[] nums = [1,2,3,4];
for (int num in nums) {
    print(num);
}
```

4.2 Explicit Type Conversions

Viper utilizes casting functions available in the standard library to convert between types as needed. For example, casting up from an int to a float is a simple as wrapping an integer value expression in the *float* function.

Explicit type conversion functions include:

- toString(x) - converts x to a string
- toFloat(x) - converts x to a float
- toInt(x) - converts x to an int
- toBool(x) - converts x to a bool
- toChar(x) - converts x to a char

Examples of using explicit type conversions:

```
/* converts 1 into '1' */
char y = toChar(1);
```

```
int x = 2;
int y = 5;
/* "25" */
string xyz = toString(2) + toString(5);
```

Note: See Section 6 for more details on explicit type casting functions.

↩️ Back to Contents 📌

5 Statements 🗣️

Viper programs are composed of a list of statements. Statements are selector statements, iterator statements, jump statements, and function statements.

5.1 Selector Statements

Selector Statements are involved with Viper's control flow. These statements are the conditionals that Viper uses to control the flow of a program. These statements include the if statement and the if/elif/else statement.

5.1.1 If Statement

The if statement takes in a boolean expression within parentheses and runs the statements within its scope if the boolean expression returns true.

5.1.2 If / Else if / Else Statement

The if statement has optional statements that can come after it such as elif and else. Else if will be run if the previous if statement's boolean expression was false. An else if statement is like an if statement in that it takes in a boolean expression in parentheses and if the boolean expression returns a value of true, then the statements within its scope will be run. There can be infinitely many elif statements after an if statement. The else statement must come after the if and all else if statements, if any. The else statement will run the statements inside its scope if all the if statements and elif statements have a boolean expression that returns false.

```
if (a == b){
    print(a);
}
else if (a > b){
    print(b);
}
else{
    print("something is wrong");
}
```

If statements also can use a special keyword "has" to check if an element is in an array. The "has" keyword returns true if the element is in the array and false otherwise. The syntax is written by typing the name of the array, followed by "has" followed by the element.

```
if (arr has 42){
  print(true);
}
else{
```

```
  print(false);
}
```

5.2 Iterator Statements

Iterator Statements are involved with Viper's ability to loop through statements. These statements compose for loops and while loops.

5.2.1 For Statement

A for statement takes in an argument in the form of (assignment; condition; iterator), followed by a list of statements within its scope. The assignment creates a variable and initializes it to a given number. The condition is a boolean expression; if it returns true, the list of statements within the for statement's scope is run. The iterator changes the value of the variable in the assignment. Then the condition is checked with the new value and if it returns true, the statements are run again, otherwise the statements are not run again.

```
for (int i = 0; i < sizeof(arr); i++){
    print(arr[i]);
}
```

A for statement can take a second form as well. The second form of a for statement is an identifer, followed by the keyword in, followed by an object that is iterable. This statement will iterate over the values in the iterable object, using the identifier for each value, and run the statements in its scope. Once there is no elements left in the iterable object, the for statement will stop.

```
for (int element in arr) {
    print(element);
}
```

5.2.2 While Statement

A while statement takes in a boolean expression. If the boolean expression returns a value of true, the statements within its scope are run. After all statements are run, the boolean expression is evaluated again; if true then statements are run again, otherwise, the while statement is done. This process repeats until the boolean expression returns a value of false.

```
while (condition){
    print("chilling");
}
```

5.3 Jump Statements

Jump statements are statements located within the scope of an iterator statement which dictates how to proceed within the iterator statement.

5.3.1 Skip Statement

The skip statement appears in for statements and while statements. When the program encounters this statement, it will ignore any statements left in the iterator statement and go back to the beginning of the iterator statement.

```
for (int element in arr){
    if (element == 2) {
        print("I'm going to skip the remaining statements");
    }
    skip;
    print("This element isn't a 2");
}
```

5.3.2 Abort Statement

The abort statement appears in for statements and while statements. When the program encounters this statement, it will ignore any statements left in the iterator statement and leave the iterator statement, proceeding with other statements within the code, if any.

```
for (int element in arr){
    if (element == 2){
        print("found it");
    }
    abort;
}
```

5.4 Function Statement

Functions take input and may return output. Functions take the form of "returnType func functionName(parameter1, parameter2, ...)" The returnType is the type of the output that must be returned from the function. The func, is literally the word func. The functionName is the name of the function which must use the same convention as variables in Viper. The (parameter1, parameter2, ...), is the input of the function where each parameter is a variable. If a function is called, the statements in its scope will run, using any parameters given to the function and then returning the value of type, returnType, using the keyword return. Functions are called by writing the function name followed by a parantheses of parameters, if any.

```
nah func foo(){
    print("Hello World!");
}
foo();
```

5.4.1 Arrow Functions

Similar to arrow functions in Javascript, or Python lambda functions, users are able to define functions with arrow functions. Users are required to specify the type of the arrow function's return value and parameters. The syntax is as follows:

```
<ret_type> func <name> (<param_type> param1, ..., <param_type> paramN)
    => expression output;
```

Note that even with zero parameters or one parameter, the () are still necessary.

Example Function Calls:

```
int func y(int x, int y, int func z) {
    return z(x + y);
}
```

```
int func times (int a, int b) => a * b;
y(10, 20, times);
```

5.4.2 Attribute Calls

Viper supports attribute calls using a Java-like syntax with a period and parentheses in parameters. For example,

```
int[] list = [1,2,3,4];
list.contains(4); /*Attribute call of contains on list. */
```

An attribute call is equivalent to a stand-alone function call that prepends the calling object to the front of its list of parameters. The following two calls are equivalent:

```
list.contains(4);
contains(list, 4);
```

↩️ Back to Contents 📌

6 Expressions 🖥️

Expressions in Viper yield the recipe for evaluation. Expressions can be any data type in its simplest form and it can include operators in more complex forms. These include simple arithmetic expressions which yield a float or integer type, or boolean expressions which yield a true or false when evaluated.

6.1 Truth-Value Expressions

Truth-Value expressions in Viper are boolean expressions. They can include logical operators and when evaluated, must return a value of type bool.

6.2 Guard Expressions

Guard expressions are an alternative way of using conditional statements. When assigning a variable, Viper uses the symbol "??" to indicate the start of a guard expression. Each subsequent statement uses a "|", except the first one and last one, followed by a boolean expression, a ":", and then a value which fits the variable data type. If the boolean expression returns a value of true, then the expression to the right of the symbol ":" is used for the value of the variable. If the boolean expression is false, the program runs the next statement following the next symbol "|". The last statement in a guard expression contains a "??" followed by a value consistent with the data type for the variable. The first statement has neither a "|" nor a "??". This can be thought of as a combination of if, elif and else statements for assigning a variable.

```
int x = ??
4 == 4 : 42;
| 5 == 3 : 24;
?? 0;
print(x);
```

stdout:

```
42
```

↩️ Back to Contents 📌

7 Operators ➗

Operators are used on values to change them. This leads to interesting and complex expressions which can be useful. The different kinds of operators are Unary, Binary, Comparative, Logical and Variable.

7.1 Unary Operators

Unary operators act on only one value. These include the not operator, the increment operator and the decrement operator.

7.1.1 The NOT Operator

The NOT operator is given the symbol "!". When placed to the left of a bool, the value of the bool is flipped. If the value was true it is now false, vice versa.

```
bool example = true;
print(!example);
```

stdout:

```
false
```

7.1.2 The Increment Operator

The increment operator is given the symbol "++". When placed to the right of an integer, the value of the integer is incremented by one.

```
int example = 0;
print(example++);
```

stdout:

```
1
```

7.1.3 The Decrement Operator

The decrement operator is given the symbol "--". When placed to the right of an integer, the value of the integer is decremented by one.

```
int example = 0;
print(example--);
```

stdout:

```
-1
```

7.2 Binary Operators

Binary operators act on two values. These include the addition operator, the subtraction operator, the multiplicative operator, the division operator, and the modulus operator.

7.2.1 The Addition Operator

The addition operator is given the symbol, "+". It acts like addition in mathematics, i.e. it is written in between two values which result in the sum of the two values.

```
int example1 = 1;
int example2 = 2;
print(example1 + example2);
```

stdout:

```
3
```

7.2.2 The Subtraction Operator

The subtraction operator is given the symbol, "-". It acts like subtraction in mathematics, i.e. it is written in between two values which result in the difference of the two values.

```
int example1 = 1;
int example2 = 2;
print(example1 - example2);
```

stdout:

```
-1
```

7.2.3 The Multiplicative Operator

The multiplicative operator is given the symbol, "*". It acts like multiplication in mathematics, i.e. it is written in between two values which result in the product of the two values.

```
int example1 = 1;
int example2 = 2;
print(example1 * example2);
```

stdout:

```
2
```

7.2.4 The Division Operator

The division operator is given the symbol, "/". It acts like division in mathematics, i.e. it is written in between two values which result in the quotient of the two values.

```
int example1 = 1;
int example2 = 2;
print(example1 / example2);
```

stdout:

```
0.5
```

7.2.5 The Modulus Operator

The modulus operator is given the symbol, "%". It acts like modulus in mathematics, i.e. it is written in between two values which result in the remainder of the two values when divided.

```
int example1 = 4;
int example2 = 2;
print(example1 % example2);
```

stdout:

```
0
```

7.3 Comparative Operators

Comparative Operators compare two values and returns a bool.

7.3.1 The Greater Than Operator

The greater than operator is given the symbol, ">". When written in between two values, it returns false if the first value is less than or equal to the second value and returns true if the first value is greater than the second value.

```
int example1 = 2;
int example2 = 2;
print(example1 > example2);
```

stdout:

```
false
```

7.3.2 The Greater Than Or Equal To Operator

The greater than or equal to operator is given the symbol, ">=". When written in between two values, it returns false if the first value is less than the second value and returns true if the first value is greater than or equal to the second value.

```
int example1 = 2;
int example2 = 2;
print(example1 >= example2);
```

stdout:

```
true
```

7.3.3 The Less Than Operator

The less than operator is given the symbol, "<". When written in between two values, it returns true if the first value is less than the second value and returns false if the first value is greater than or equal to the second value.

```
int example1 = 2;
int example2 = 2;
print(example1 < example2);
```

stdout:

```
false
```

7.3.4 The Less Than Or Equal To Operator

The less than or equal to operator is given the symbol, "<=". When written in between two values, it returns true if the first value is less than or equal to the second value and returns false if the first value is greater than the second value.

```
int example1 = 2;
int example2 = 2;
print(example1 <= example2);
```

stdout:

```
true
```

7.3.5 The Equals Operator

The equals operator is given the symbol, "==". When written in between two values, it returns true if the first value is equal to the second value and returns false if the first value is not equal to the second value.

```
int example1 = 2;
int example2 = 2;
print(example1 == example2);
```

stdout:

```
true
```

7.3.6 The Not Equals Operator

The not equals operator is given the symbol, "!=". When written in between two values, it returns true if the first value is not equal to the second value and returns false if the first value is equal to the second value.

```
int example1 = 2;
int example2 = 2;
print(example1 != example2);
```

stdout:

```
false
```

7.4 Logical Operators

The logical operators take in two bool values and returns a bool value. These operators include the AND operator and the OR operator. The HAS operator, which takes in an object containing elements and an element, returning a bool value.

7.4.1 The AND Operator

The AND operator is given the symbol, "&&". When written in between two bool values, it returns true if both values are true and false otherwise.

```
bool example1 = true;
bool example2 = false;
```

```
print((example1 && example2));
```

stdout:

```
false
```

7.4.2 The OR Operator

The OR operator is given the symbol, "||". When written in between two bool values, it returns false if both values are false and true otherwise.

```
bool example1 = true;
bool example2 = false;
print((example1 || example2));
```

stdout:

```
true
```

7.5 Variable Operators

Variable operators act on a variable and an integer. These include +=, -=, *=, and /=.

7.5.1 The += Operator

The += operator is written in between a variable on the left hand side and an integer on the right hand side. The integer value on the right hand side is added to the variable value, which is updated as the new value for the variable.

```
int example1 = 1;
example1 += 1;
print(example1);
```

stdout:

```
2
```

7.5.2 The -= Operator

The -= operator is written in between a variable on the left hand side and an integer on the right hand side. The integer value on the right hand side is subtracted from the variable value, which is updated as the new value for the variable.

```
int example1 = 1;
example1 -= 1;
print(example1);
```

stdout:

```
0
```

7.5.3 The *= Operator

The *= operator is written in between a variable on the left hand side and an integer on the right hand side. The integer value on the right hand side is multiplied by the variable value, which is updated as the new value for the variable.

```
int example1 = 1;
example1 *= 1;
print(example1);
```

stdout:

```
1
```

7.5.4 The /= Operator

The /= operator is written in between a variable on the left hand side and an integer on the right hand side. The integer value on the right hand side divides the variable value, which is updated as the new value for the variable.

```
int example1 = 1;
example1 /= 1;
print(example1);
```

stdout:

```
1
```

7.5.5 The = Operator

The = operator is written between a variable name on the left hand side and a value on the right hand side. The value on the right hand side is assigned as the value for the variable on the left hand side. If the variable exists already, the value of the variable is overwritten, otherwise a new variable is created.

```
int example1 = 1;
print(example1);
```

stdout:

```
1
```

7.5.6 The Ternary Operator

The Ternary Operator is given the symbol "?". This operator provides a short hand for an if-else statement and saves the result in a variable. When using the ternary operator in assigning a variable, Viper expects a boolean expression followed by the ternary operator "?". After the ternary operator, a value that matches the type of the variable being assigned is expected, followed by a ":" and another value that matches the type of the variable being assigned. If the boolean expression returns a truth value of true, then the first value is assigned to the variable, otherwise the second value is assigned.

```
int x = 5 < 10 ? 42 : 0;
print(x);
```

stdout:

```
42
```

7.6 Precedence of Operators

The precedence of operators is important for determining how to write programs in Viper. It is important to note that any expression within parentheses has the highest precedence.

7.6.1 Precedence of Unary Operators

Unary operators receive the highest precedence, second to parentheses.

7.6.2 Precedence of Binary Operators

The multiplicative operator, division operator, and modulus operator are left associative and have a higher precedence than the addition operator and the subtraction operator. The addition and subtraction operator are also left associative.

7.6.3 Precedence of Comparative Operators

The >, >=, <, <= operators are given higher precedence than the != and == operators.

7.6.4 Precedence of Logical Operators

The and operator is given higher precedence than the or operator.

7.6.5 Precedence of Variable Operators

Variable operators are given a lower precedence than binary operators and are right associative.

↩️ Back to Contents 📌

8 Scope 👀

Viper uses curly braces to define scope. For example, a for loop can be established in a number of different ways:

```
for (string elem in list) {
    print(elem);
}

/* Is the same as: */

for (string elem in list)
{
    print(elem);
}

/* Is the same as: */

for (string elem in list)
{ print(elem); }
```

This will function in the same manner as expected with function definitions, conditionals, etc.

↩️ Back to Contents 📌

9 Standard Library 📚

Viper's standard library includes methods and functionalities that are used in nearly every program. This is to balance the tediousness of requiring numerous lines of imports and keeping compilation quick and program bloat low.

9.1 Math Functions

Viper provides built-in methods for common arithmetic operations.

9.1.1 `sqrt()`

`sqrt()` returns the square root of the given `int`, `float`, or `char` as a `float`. If a `char` is given, the decimal value of its ASCII symbol is used.

```
float four = sqrt(16); /* Returns 4.0 */
float two = sqrt(four); /* Returns 2.0 */
float eight = sqrt('@'); /* Returns 8.0 */
```

9.1.2 `pow()`

`pow()` can be polymorphically used into two ways. If only one `int`, `float`, or `char` input is given, it returns the square (x^2) of that input. If two `int`, `float`, or `char` inputs are given, it returns the `float` result of raising the first input to the power of the second. If `char`s are given, the decimal values of their ASCII symbols are used.

```
float one_four_four = pow(12); /* Returns 144.0 */
float a_milly = pow(10.0, 6); /* Returns 1000000.0 */
```

9.1.3 `floor()`

`floor()` takes a `float` input and returns the `int` result of truncating the `float`'s decimal components.

```
int zero = floor(0.999); /* Returns 0 */
int whole_num = floor(72.0); /* Returns 72 */
```

9.1.4 `ceil()`

`ceil()` does the opposite of `floor()`. It takes a `float` input and returns the closest `int` greater than or equal to the given value.

```
int five = ceil(4.1); /* Returns 5 */
int four = ceil(4.0); /* Returns 4 */
```

9.1.5 `round()`

`round()` takes a `float` input and returns the closest `int` to the given value. Values of ending in .5 always round to the next greatest `int`.

```
int three = round(3.2); /* Returns 3 */
int also_three = round(3.3); /* Returns 3 */
int neg_three = round(-3.5); /* Returns -3 */
```

9.1.6 min()

`min()` takes either two `float`s, two `int`s, or two `char`s as input and returns the smallest value between the two. If `char`s are given, the decimal values of their ASCII symbols are used. The function is overloaded, so the return type is the same as the input type.

```
int negative1 = min(-1, 1); /* Returns -1 */
float gpa = min(5.7, 4.0); /* Returns 4.0 */
char a_char = min('b', 'a'); /* Returns 'a' */
```

9.1.7 max()

`max()` takes either two `float`s, two `int`s, or two `char`s as input and returns the largest value between the two. If `char`s are given, the decimal values of their ASCII symbols are used. The function is overloaded, so the return type is the same as the input type.

```
int positive1 = max(-1, 1); /* Returns 1 */
float big = max(8.78, 9.9); /* Returns 9.9 */
char e_char = max('e', 'a'); /* Returns 'e' */
```

9.1.8 trunc()

`trunc()` takes a `float` and `int` as input, and returns the `float` truncated to the number of decimal points specified by the `int`. The `int` must be greater than zero.

```
float whee = trunc(0.123456789, 3); /* Returns 0.123 */
float almost_whole_num = trunc(whee, 1); /* Returns 0.1 */
float bad_bad_bad = trunc(0.99, 0); /* Throws an error */
```

9.2 Primitive Type Casting Functions

Viper's standard library provides methods for casting between types for ease of use and readability. Type casting functions include:

9.2.1 toChar()

`toChar()` converts to `char`s. The input can be an `int` in range [0, 255], for which the output is the `char` corresponding to the ASCII value of the `int`. The input can also be a `string`, for which the output is the `char` value of the first character in the `string`. Passing any other types or `nah` to `toChar()` results in an error.

```
char int_chr = char(36); /* Returns '$' */
char str_char = char(string(true)); /* Returns 't' */
char no_dont_do_it = char(33.4); /* Throws an error */
```

9.2.2 toInt()

`toInt()` casts certain types to integer values. Given a `char`, `toInt()` returns the decimal value of the `char`'s ASCII code. Given a `float`, `toInt()` returns the result of truncating all decimal components. Given a `bool`, `int()` returns 0 for values of `false`, and 1 for values of `true`. Passing any other types or `nah` to `toInt()` results in an error.

```
int chr_int = toInt('R'); /* Returns 82 */
int str_int = toInt(7.999); /* Returns 7 */
int zero = toInt(false); /* Returns 0 */
int one = toInt(true); /* Returns 1 */
```

9.2.3 `toFloat()`

`toFloat()` casts `int`s and `char`s to float values. Given an `int`, `toFloat()` returns the `float` equivalent of that `int`, appending a decimal point and a single 0. Given a `char`, `float` does the same thing with the decimal value of the `char`'s ASCII code. Passing any other types or `nah` to `toFloat()` results in an error.

```
float int_float = toFloat(333); /* Returns 333.0 */
float char_float = toFloat('&'); /* Returns 38.0 */
float noooooo = toFloat("if you smoke"); /* Returns an error */
```

9.2.4 `toBool()`

`toBool()` converts any data type to either `true` or `false`, depending on the value. It's called implicitly when using a non-boolean type in a boolean expression For example, the following code implicitly calls `toBool()` on `x`:

```
int x = 0;
if (x) {
    print("Yes");
}
```

See the below table for details on what values `toBool()` maps to true and result in `true` and what values result in `false`:

Type	`true` values	`false` values
`char`	All `char`s but `'\0'` and `''`	`'\0'` and `''`
`int`	$[-2^{31}, -1]$, $[1, 2^{31} - 1]$	0
`float`	All `float`s but 0.0	0.0
`bool`	`true`	`false`
`string`	All non-empty `string`s	`""`
`nah`	n/a	`nah`

9.2.5 `toString()`

`toString()` converts any type to a `string`, which is useful for printing. See the below examples for type-specific details.

```
string char_str = toString('z'); /* Returns "z" */
string int_str = toString(30); /* Returns "30" */
string float_str = toString(34.5); /* Returns "34.5" */
string bool_str = toString(false); /* Returns "false" */
```

9.3 Miscellaneous Functions

These functions are unclassified, and are useful in a variety of situation.

9.3.1 print()

`print()` prints the given `string` to a new line of standard output. It throws errors if it encounters a type other than `string`, and prints an empty new line if no inputs are given. To get around this, use the str() method with `string` concatenation (+).

```
print("Hola Mundo");
print();
print(str('a'));
print("I have " + str(388) + " bananas");
float nose = -0.3;
print(str(true) + str(nose));

/* print(4999); Uncommenting this line throws an error */
```

stdout:

```
Hola Mundo

a
I have 388 bananas
true-0.3
```

9.3.2 len()

The `len()` function is an all-purpose method that returns the `int` size of `string`s, `list`s, `group`s, and `dict`s. For `string`s, `len()` returns the number of characters in the `string` (excluding the null terminator at the end of its underlying `list` of `char`s). For `list`s, `len()` returns the number of elements in the `list`. For `group`s, `len()` returns the number of elements in the `group`, and for `dict`s, `len()` returns the number of key-value pairs.

```
int str_length = len("Nice sock!\n\t"); /* str_length = 12 */
int list_length = len([3, 1, 6, 76]); /* list_length = 4 */
(int, float) groupie = (8, 8.8);
int group_length = len(groupie); /* group_length = 2 */
int dict_length = len(["word": 3,
                       "knees": 5,
                       "port": 90]); /* dict_length = 3 */
```

9.3.3 contains()

The `contains()` function returns 1 if the element exists in the object of elements and 0 otherwise.

```
char[] chrlist = ['a', 'b', 'c'];
if (chrlist.contains('a'){        /* same as contains(chrlist, 'a'); */
  print(true);
}
else{
  print(false);
}
```

9.4 Lists

List functionality is provided through the standard library. Lists are mutable, static sequences of a single data type with fixed sizes. Lists are accessed and modified with square brackets ([]). See Higher-Order Data Types: lists for more details on instantiation and modification. The following sections describe additional list-specific operations implemented in the list api. These operations are all called on instances of lists, and thus take the form `list.operation(parameters)`.

9.4.1 append()

`append()` adds an input element to the end of a specified list. Because lists have fixed sizes, the original list remains unmodified, and `append()` returns a new list with the input element attached. The type of the input to `append()` must match the type of the list.

```
int[] channels = [31, 44, 21];
int[] new_channels = channels.append(54);
/* new_channels contains: [31, 44, 21, 54] */
```

9.4.2 contains()

`contains()` takes in an element of the list type and checks as to whether that element exists within the last. If the element exists, the function returns the `int` value of 1, and `int` value of 0 if the element does not exist.

```
char[] notes = ['a', 'c', 'd', 'c'];
print(notes.contains('a'))
print(notes.contains('b'))
/*
1
-1
*/
```

9.5 Dicts

Dictionary functionality is provided through the standard library. Dictionaries are mutable sequences of a two data types with dynamic sizes. Dictionary key, value pairs are accessed and modified with square brackets ([]). See Higher-Order Data Types: dicts for more details on instantiation and modification. The following sections describe additional dict-specific operations implemented in the list api. These operations are all called on instances of dict, and thus take the form `dict.operation(parameters)`.

9.5.1 add()

`add()` adds an input key and value, and adds it to the dictionary. If the key already exists, then the value for the same is overriden.

```
[string: int] word_counts = [];
word_counts.add("a", 1);
/* word_counts contains: [["a", 1]] */
```

9.5.2 keys()

`keys()` outputs a unique list of all the keys present the dictionary. The return type for the `keys()` function is dependant on the key type for the dictionary being called. An empty list is returned if no keys exist

```
[string: int] word_counts = [];
word_counts.add("a", 1);
print(word_counts.keys())
/* word_counts contains: ["a"] */
```

9.5.3 contains()

`contains()` takes in an of the dict's key type and returns an `int` value of `0` if the key does not exist in the dict, and `1` if it does.

```
[string: int] word_counts = [];
word_counts.add("a", 1);
print(word_counts.contains("a"))
/*
1
*/
```

↩️ Back to Contents 📌

10 Sample Code 🧩

Example programs written in Viper below.

10.1 Fizzbuzz examples:

```
/* standard fizzbuzz
for-loop solution */
for (int i = 1; i <= 100; i++) {
   if (i % 15 == 0) {
       print("fizzbuzz");
   } else if (i % 3 == 0) {
       print("fuzz");
   } else if (i % 5 == 0) {
       print("buzz");
   } else {
       print(i);
   }
}
```

```
/* fizzbuzz for-loop using an arrow function for divisibity*/
bool func isDivisible(int x, int div) => (x % div == 0);

for(int i = 1; i <= 100; i+=1) {
    if(isDivisible(i, 15)) {
        print("fizzbuzz");
    }
    else if(isDivisible(i, 3)) {
        print("fizz");
    } else if(isDivisible(i, 5)) {
        print("buzz");
    } else {
        print(i);
    }
}
```

10.2 Calculate Function Example

```
/* Choosing an operation to compute, using if-else */
int func calc(int a, int b, char op) {
    int res;
    if(op == '+') {
        res = a + b;
    } else if (op == '-') {
        res = a - b;
    } else if (op == '*') {
        res = a * b;
    } else if (op == '/') {
        res = a / b;
    } else {
        res = a;
    }
    return res;
}

/* The same code, using guard matching!*/
int func calc(int a, int b, char op) {
    int res = ??
        op == '+' : (a + b)
      | op == '-' : (a - b)
      | op == '*' : (a * b)
      | op == '/' : (a / b)
    ?? a ;

    return res;
}
```

10.3 Wordcounts in a string array

```
[string: int] func count_words(string[] str) {
    [string: int] word_counts = [];
    for(string x in str) {
        if(word_counts.contains(x) == 1) {
            word_counts.add(x, word_counts[x] + 1);
        } else {
            word_counts.add(x, 1);
        }
    }
    return word_counts;
}

string [] test = ["Hello", "world", "succotash", "am", "world", "hello", "world",
"viper", "Viper", "viper"];

[string: int] counts = count_words(test);

string[] unique_keys = counts.keys();

for(string key in unique_keys) {
    print(key);
    print("=");
    print(counts[key]);
    print("|");
}
```

11 Language Grammar

```
program -> decls | EOF

decls ->
  ''
| decls fdecl
| decls stmt

fdecl ->
  typ ID LPAREN formals_opt RPAREN LBRACE stmt_list RBRACE
| typ ID LPAREN formals_opt RPAREN ARROW stmt

formals_opt ->
  ''
| formal_list

formal_list ->
  typ ID
| formal_list COMMA typ ID

typ ->
  INT
| BOOL
```

```
| NAH
| CHAR
| FLOAT
| STRING
| typ FUNC
| typ ARROPEN ARRCLOSE
| LPAREN type_list RPAREN
| ARROPEN typ COLON typ ARRCLOSE

type_list ->
  typ
| typ COMMA type_list

stmt_list ->
  ''
| stmt_list stmt

stmt ->
  expr SEMI
| typ ID SEMI
| RETURN SEMI
| RETURN expr SEMI
| SKIP SEMI
| ABORT SEMI
| PANIC expr SEMI
| LBRACE stmt_list RBRACE
| IF LPAREN expr RPAREN stmt %prec NOELSE
| IF LPAREN expr RPAREN stmt ELSE stmt
| FOR LPAREN expr_opt SEMI expr SEMI expr_opt RPAREN stmt
| FOR LPAREN ID IN expr RPAREN stmt
| FOR LPAREN typ ID IN expr RPAREN stmt
| FOR LPAREN LPAREN formal_list RPAREN IN expr RPAREN stmt
| WHILE LPAREN expr RPAREN stmt

expr_opt ->
  ''
| expr

expr ->
  INTLIT
| CHARLIT
| FLOATLIT
| STRLIT
| TRUE
| FALSE
| ID
| list_exp
| dict_exp
| expr PLUS   expr
| expr MINUS  expr
| expr TIMES  expr
| expr DIVIDE expr
```

```
| expr MODULO expr
| ID PLUS_ASSIGN expr
| ID MINUS_ASSIGN expr
| ID TIMES_ASSIGN expr
| ID DIVIDE_ASSIGN expr
| expr EQ     expr
| expr NEQ    expr
| expr LT     expr
| expr LEQ    expr
| expr GT     expr
| expr GEQ    expr
| expr AND    expr
| expr OR     expr
| expr HAS     expr
| expr QUESTION expr COLON expr
| MINUS expr %prec NEG
| NOT expr
| expr PLUS PLUS %prec INCR
| expr MINUS MINUS %prec DECR
| typ ID ASSIGN expr
| ID ASSIGN expr
| LPAREN formal_list RPAREN ASSIGN expr
| expr ARROPEN expr ARRCLOSE
| expr ARROPEN expr ARRCLOSE ASSIGN expr
| typ ID ASSIGN MATCH pattern
| ID ASSIGN MATCH pattern
| ID LPAREN actuals_opt RPAREN
| expr DOT ID LPAREN actuals_opt RPAREN
| LPAREN expr RPAREN

pattern -> c_pattern MATCH expr

c_pattern ->
  expr COLON expr
| expr COLON expr BAR c_pattern

dict_exp -> ARROPEN dict_elems ARRCLOSE

dict_elems ->
  dict_elem
| dict_elem COMMA dict_elems

dict_elem -> expr COLON expr

list_exp -> ARROPEN list_elems ARRCLOSE

list_elems ->
  ''
| expr
| expr COMMA list_elems

actuals_opt ->
```

```
  ''
| actuals_list

actuals_list ->
  expr
| actuals_list COMMA expr
```

12 Project Plan

12.1 Specification Process

We started by creating an initial list of all the various features we thought would be cool to include in a language. While keeping in mind our initial goal of a statically-typed, compilable, scripting language, we thought about what the most important features and pieces of syntactic sugar would also have to be inherited from the inspiration languages Python and Javascript. We compiled a list of everything we thought necessary, from dicts, lists, and useful math operations to more complex features like arrow functions, ternaries, and iterator sugar. Additionally, we planned to add scoping identification through curly braces or indentation, but had to cut the white-space scoping due to time constraints.

From there, our first concrete specification came when we defined our lexicon and syntax, which then got translated into the scanner and parser, respectively. We figured out how many features we wanted to be expressible, and then decided how they would be expressed syntactically. This was then inscribed into our original Language Reference Manual.

As we continued working on semantic checks, codegen, and the standard library, we had frequent iterative updates to our specification due to new, better ideas, conflicts, or removed features. The only major changes were the removal of whitespace scoping and the addition of pattern matching. Otherwise, most changes were minor syntax changes. We also intended to add tuples (or "groups" as we called them), but ultimately decided against it in order to put resources into more interesting features.

12.2 Development Process

Development followed the stages of the compiler architecture. We began with the scanner, parser, and ast, with the three honestly being quite simultaneous, although each individual change occurred in that order. We then got `print("hello world")` working in codegen while simultaneously working on semantic checking. After hello world, we shifted into first desugaring syntax in layer one of semantic analysis, then type/validity checking in layer two. From here progress continued on updated iterations of semantic checking, while the rest of codegen and the standard library began to ramp up. The standard library was completed before the codegen, which was our final push on the project. It is worth noting that there was overlap in these tasks (they were not completed in isolation), and testing was set up at each one of these checkpoints that allowed us to easily identify where blocks were occurring.

12.3 Testing Process

Viper's test suite supports testing for semantic analysis and LLVM code generation. The test suite can also run both types of tests sequentially, allowing a complete end-to-end check of the compiler. Semantic checks scan and parse a program into an AST, desugar applicable nodes in the AST, and then semantically check the AST. Upon success, the check produces no output, indicating that the program is semantically valid and is ready to be broken down into LLVM. Upon failure, the check will print an error corresponding to the step that failed. LLVM tests have a few options. One test simply prints the LLVM of the code created in codegen.ml, allowing analysis of the generated basic blocks. The other option executes the LLVM and

prints the corresponding output if valid, or a Codgen error if invalid. Testing all of our test programs runs this last option, and compares each program's output to the corresponding .out file. If all outputs match their .out files, the test succeeds, but if any one fails, an error message is displayed describing the file that failed and the exception that resulted in failure.

12.4 Team Responsibilities

Raghav - Lexer, Parser, AST, Desugaring
Mustafa - Lexer, Parser, AST, C Libraries
Matthew - Semantics, Documentation Tommy - Semantics, Documentation
Trey - Code Generation, Documentation

12.5 Project Timeline

Date	Milestone
February 24th	Scanner, Parser, AST Implemented and LRM Written
March 24th	Hello World Implemented
April 7th	Desugar Implemented
April 16th	Semantic Checking Implemented
April 25th	Codegen Implemented
## 12.6 Software Development Environment	
We had the following programming and development environment:	

Programming language for building compiler : Ocaml version 4.05.0

Development environments: We used vscode and vim.

12.7 Programming Style

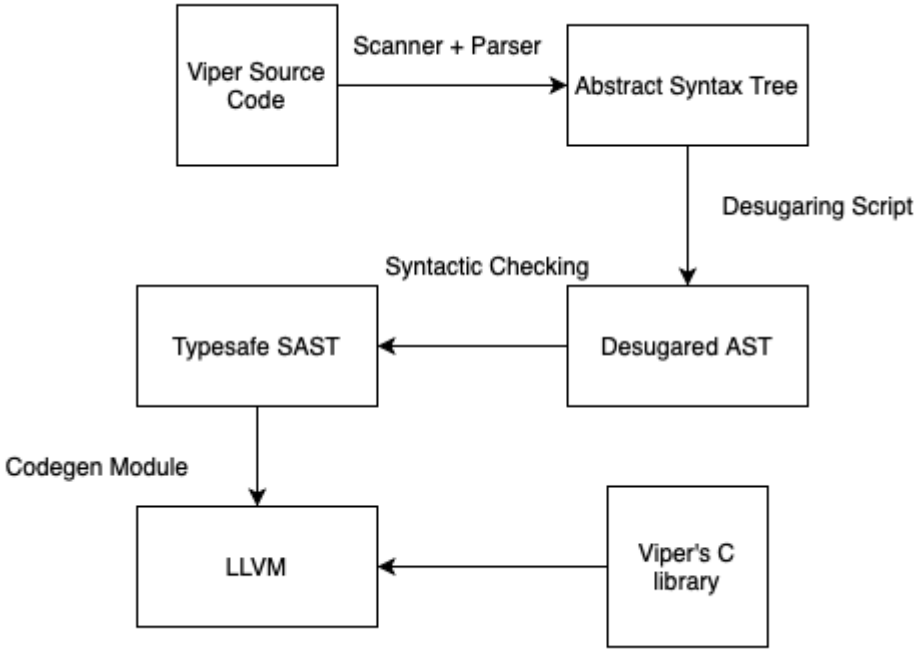
We generally wrote code according to the following style guidelines:

Some files begin with multi-line OCaml functions that describe their purpose, especially when the file name is vague.
Open and import statements are the first lines of compiled code in every file.
Modules, types, data structures, exceptions, and global variables follow open statements.
Non-nested global function declarations are preceded by multi-line OCaml comments describing general behavior.
Nested functions are only used in their parent function, and usually don't include descriptive comments. Particularly important nested functions include comments. OCaml "in" statements appear at the end of lines (when applicable), allowing following "let" statements to begin the next line.
Pattern matching guard statements for S/AST elements appear in the same order that they are defined in the S/AST.
Pattern matching on elements with tuples uses parentheses (for example, For(e1, e2, e3, s) instead of For e1, e2, e3, s.).
Lines generally wrap at around 80 characters, when applicable.

13 Architectural Design

13.1 The Compiler

Our compiler closely follows the structure we learned about in lecture, with the addition of a desugaring step between the parser and semantic checker. See below for more details.



13.2 The Lexer

The first step of the compiler is the lexer. The lexer scans the program and creates tokens based on spaces. The tokens include common tokens such as assignment(=), operations(+,-,) etc. Some of the less common tokens Viper accepts are guards(|) and ternaries(??).

13.3 The Parser

After the lexer returns a list of tokens, the compiler then parses through the tokens and returns types based on the Abstract Syntax Tree. The Abstract Syntax tree separates types by expressions and statements. Statements include things like if conditionals and loops while expressions include binary operations, unary operations and some of our cool syntactic sugar like ternaries. The parser also separates the Viper program into a tuple of global statements and function declarations. This allows Viper to support statements outside of functions. The current scoping system uses “{}”.

13.4 Desugaring

A big portion of the compiler is the desugaring stage. The cool parts of Viper are pattern matching, ternary operators, for loop iterators and more. These parts, while they seem new and interesting, are simply syntactic sugar. The desugaring stage takes the syntactic sugar and replaces the instances in the Abstract

Syntax Tree with simpler instances such as if conditionals and while loops. The for loop is also syntactic sugar and is desugared into a while loop.

13.5 The Semantic Checker

After desugaring the AST, it is sent to the semantic checker. The semantic checker is composed of several modules and driven by a driver file. First, all declarations and declaration assignments are checked for Nah types and duplicates. Viper allows for declarations inside of loops, therefore scoping becomes complicated. To achieve the desired scoping outcome, we implemented symbol tables that are connected like a tree, each having a parent which is in an outer scope. Using this method, we are able to check the declarations and declaration assignments inside of loops while creating symbol tables for the global statements and the function declarations which the driver uses. Viper also allows overloading functions, therefore the key for these symbol tables are a concatenation of the function name and the parameter types. After checking and creating these symbol tables, the driver semantically checks the global statements and then the function declarations, returning an SAST. The semantic checker then checks to see if a main exists; if it does then it does nothing, however if a main does not exist, it creates a main and puts all the global statements in it. Finally, the semantically checked global statements and function declarations are returned for code generation.

13.6 The Code Generator

The Code Generator is finally given two lists passed through the translate function. This translate function is the entire file as the entirety of codegen.ml is a sequence of let ... in statements that results in returning the module containing all the instructions for the OCaml LLVM. The functions list is mapped over a large function to build out the bodies of the functions. Statements is largely ignored as the previous compiler steps moved the top-level statements into a main function that is then added to functions. The two core functions for creating the function bodies are stmt and expr which respectively build out instructions for statements and expressions in function bodies. As a large portion of the functionality of our language is desugared prior to reaching codegen.ml, we cut out a large amount of headache of using llvm bindings. Much of our list and dictionary functionality is built out in our C library, but other than that our operators, assignments, declarations, etc. are all built efficiently through LLVM.

13.7 C Libraries

The C library is used for three components of Viper, Advanced math functions/operations, List data structure and methods, Dictionary data structure and methods The C library can be found in `library.c`, with optional tests for all standard library data structures and functions included in the main function (blocked by an ifdef). Note that `library.c` also includes `stdio.h`, `stdlib.h`, `math.h`, and `string.h`.

14 Testing

14.1 Scanner, Parser, AST

Tests for the scanning and parsing stage contain syntactically correct code which may or may not be semantically correct code. The tests also include demonstrations of incorrect syntax which yields a syntax error. There is a test that covers everything in the AST for robustness.

14.2 Semantic Checker

The AST tests are not the right kind to test on the semantic checker. This is because an assignment can be syntactically correct, however if the variable never was declared, the Semantic Checker will throw an error. Therefore a new list of tests had to be created to abide by the semantic checker. The most important tests regard type checking; this includes checking the types of declaration assignments, assignments, function return types, loop predicates, etc. This set of tests was forced to be type sensitive.

`14.3` Code Generation

Code generation code was the most straightforward to write. Given a program with a deterministic output, `.out` files were utilised to compare the viper `stdout` output to, in order to validate if the output was exactly as expected. This was incorporated into the existing test suite with a new command flag, and the test suite automatically flags and compares those test cases that have a corresponding `.out` file (ex: `test.vp` and `test.vp.out`) with the output that their LLVM code generates, in order to validate functionality.

`15` Lessons Learned

Raghav - Functional programming is crazy. It took me forever to figure out, but once I did it really changed the way I think about problem solving. Specifically with regards to my role in the project, PLT really taught me how much you can do with just breaking up complex things into simpler blocks. Be it the whole LLVM experience, or my role with regards to desugaring code into its more basic forms (for to while, pattern matching to else-ifs, etc) this class really showed me that spending a little time up-front writing code that can be broken down can literally save you whole nights of sleep.

Advice: Embrace the chaos. Things will break, sometimes parts and sometimes everything. Don't panic, and try to tackle problems one by one. Try to build small, and build an entire pipeline. Don't go module by module, like we did. Regardless though, just enjoy the experience because watching your own code compile using your own syntax via your own compiler is the most satisfying feeling you'll have at Columbia.

Mustafa - Grammars are wild and powerful. I had a ton of fun taking time experimenting with all of the funky syntax and features possible when building the grammar for our parser. That’s one of the bigger bottlenecks in your language’s power, so make sure to mess around with it and add some new stuff. Also, lambda calculus is cracked. Super cool lesson towards the end of the class, I absolutely loved it.

Advice: Don’t try to get everything working at once. Nothing will ever work. Get one thing working. Then another. And another. The grammar, parser, and AST will need to be kinda-complete before semantics and codegen, but start by making the simplest things fully compile, then start adding more and more.

Matthew - I have a newfound respect for semantic checking, which is where I spent a majority of my time coding. The semantic checker is the glue between the frontend and backend of the compiler. I needed to make sure that I was reaching every case in the AST properly, bend to the structure in which our program is situated, and then return a simplified SAST to make codegen as easy as it can be. I also learned how important communication is for a large project with many moving parts. A lot of time can be saved by collaborating and planning the architecture of the compiler together opposed to trying to figure out everyone's code alone.

Advice: Start early, ask questions, and try to have your demo working early so that you can build more and enjoy the potential of writing a compiler.

Tommy - The beauty of this course is that it bridges the gap between programmer and computer. Before PLT, I knew almost nothing about the compilation process, except that it magically sucks in a program and produces some output. Today, not only do I know how to think like a programmer, but I’ve also learned how to think like a computer. I now understand why programming languages look the way they do, and what

implications small changes in code can have for the computer running it, down to the lowest levels of compilation. Knowing how the code I write will be interpreted, checked, and eventually run has made me a more thoughtful developer and a more efficient debugger. In addition to this general knowledge, I've picked up a number of concrete skills. Some of these include generating Makefiles, better understanding the command line, thinking about low-level languages like LLVM and assembly, and of course, coding in the first functional language I've seen, OCaml.

Advice: My advice to future students would be to start with a smaller set of language functionality. We started by brainstorming our favorite features of many different languages, and then sought to encapsulate them all in Viper. This seemed really cool at first, but we didn't realize how much work it would create in the future. By the time we got to code generation, we were stretched pretty thin, and we weren't able to flush out all of our language's features as fully as I would have liked. I also would recommend not using as much desugaring as we did. While desugars seemed like clever and efficient ways to bring new features to Viper's syntax, they proved to be difficult to maintain in semantic checking and LLVM generation. Keeping them as their own statements makes for more work, but allows for better control and understanding of the language as a whole.

Trey - Many classes throughout college leave you feeling like you didn't really learn much. Much of my time at Columbia has been cramming finals and assignments last second. This is not a project you can do that with. This project is very hands on and forces you to learn about every single part of the compiler to work efficiently and effectively. MicroC shows you hints at where to start but that is a mere drop in the bucket for what is required to make the entire language. Coming out on the other side of this project, I can honestly say that I learned a lot about how to effectively manage a group project, my personal programming interests, how to work with documentation that is quite weak, and solve deep problems that require a lot of thinking. Overall, a solid project.

Advice: Whoever is lucky enough to work on codegen, I am about to spill the secret. When the way to write the llvm bindings to create the underlying llvm is not immediately clear, use the clang compiler to compile a similar C function. This will show you how to create the necessary instructions in LLVM. Then CTRL-F for the instruction in the OCaml LLVM bindings to see how they are used. Then write the instructions into codegen. LEARN HOW TO USE LLTYPES AND LLVALUES. Once you do this, codegen is a breeze. :)

16 Appendix

16.1 Git Logs

```
commit c69cc86bc7b2bbb43cede87ce2255e42792eed34
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Mon Apr 26 20:42:32 2021 -0700

    Update Report.md

commit 6bdc171792ee3cfc9c5e495496696da4cd60aa07
Merge: 9ebcf15 2dd7396
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Mon Apr 26 20:40:46 2021 -0700

    Merge pull request #57 from raghavmecheri/tgomezzzz-patch-1

    update table of contents
```

```
commit 2dd739660f8631c97afc782904e4978ba5828b6b
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Mon Apr 26 20:40:38 2021 -0700

    update table of contents

commit 9ebcf1512a3cd97ebbd7d2893a401665b5d3b485
Author: Raghav Mecheri <37787004+raghavmecheri@users.noreply.github.com>
Date:   Mon Apr 26 21:39:30 2021 -0600

    Update Report.md

commit cd2819853a1024d675571360d75b584e759cb57e
Merge: 2635853 04cc8e0
Author: Raghav Mecheri <raghav.mecheri@columbia.edu>
Date:   Mon Apr 26 21:37:15 2021 -0600

    Merge branch 'main' of github.com:raghavmecheri/succotash into main

commit 2635853648eca77c3e98586995678b235d4fb452
Author: Raghav Mecheri <raghav.mecheri@columbia.edu>
Date:   Mon Apr 26 21:37:09 2021 -0600

    Added report image

commit 04cc8e0f400e3d7efde05e3ecdfbf315cf73c87c
Merge: a633311 7757028
Author: treygilliland <jlg2266@columbia.edu>
Date:   Mon Apr 26 23:32:31 2021 -0400

    Merge branch 'main' of https://github.com/raghavmecheri/viper into main

commit a6333112af45db5b65b44f49c5841fe25191b67b
Author: treygilliland <jlg2266@columbia.edu>
Date:   Mon Apr 26 23:32:24 2021 -0400

    Added Codegen section

commit 7757028e2f09885f675784084aec154f2d7725e6
Author: Raghav Mecheri <raghav.mecheri@columbia.edu>
Date:   Mon Apr 26 21:30:01 2021 -0600

    Added src

commit 3fe21c77812c70413811a7638d699d292dcd25f5
Author: Raghav Mecheri <raghav.mecheri@columbia.edu>
Date:   Mon Apr 26 21:26:51 2021 -0600

    nuked extra files

commit bff065d80946b2bd189127926d7bd3591bb63676
Author: treygilliland <jlg2266@columbia.edu>
```

```
Date:   Mon Apr 26 23:24:34 2021 -0400

    Added advice and lessons

commit ce640008f8ea771be807e47146056797326acaf3
Author: Raghav Mecheri <raghav.mecheri@columbia.edu>
Date:   Mon Apr 26 21:19:30 2021 -0600

    Test suite

commit e12f296aae95e2cbb22681d4915116db741bea3a
Merge: c9dd5b5 1d17b1f
Author: Raghav Mecheri <raghav.mecheri@columbia.edu>
Date:   Mon Apr 26 21:16:27 2021 -0600

    Merge branch 'main' of github.com:raghavmecheri/succotash into main

commit c9dd5b54e9b16de0b645e9c613a4198dc722c09c
Author: Raghav Mecheri <raghav.mecheri@columbia.edu>
Date:   Mon Apr 26 21:16:24 2021 -0600

    Eh

commit 1d17b1ffefddb1448c6974ce37c81513081e0de4
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Mon Apr 26 23:15:18 2021 -0400

    testing process and sde

commit 5fb396c4a1b91d39f7a228e31e18bca3e5dbadff
Author: Mustafa Eyceoz <maxusmusti@gmail.com>
Date:   Mon Apr 26 23:12:54 2021 -0400

    Update Report.md

commit cf71b95a1da4f869d0dad09e521f751e10061f6c
Merge: 8cb9cf4 0cf7cc2
Author: treygilliland <jlg2266@columbia.edu>
Date:   Mon Apr 26 23:09:47 2021 -0400

    Merge branch 'main' of https://github.com/raghavmecheri/viper into main

commit 8cb9cf4b394fc4e2e4ee8aa8224e3c4a5d464074
Author: treygilliland <jlg2266@columbia.edu>
Date:   Mon Apr 26 23:09:36 2021 -0400

    Final documentation cleanup for codegen

commit 0cf7cc2b985eb50f8ed5197dec6050a10f325069
Author: Mustafa Eyceoz <maxusmusti@gmail.com>
Date:   Mon Apr 26 23:01:49 2021 -0400
```

```
    Update Report.md

commit d1bf544c5ff602731be84d668d9482b1af32799e
Author: Mustafa Eyceoz <maxusmusti@gmail.com>
Date:   Mon Apr 26 22:56:30 2021 -0400

    Update Report.md

commit 26a3061594bf7eddbaec59d23858b4ea7774d592
Author: Mustafa Eyceoz <maxusmusti@gmail.com>
Date:   Mon Apr 26 22:54:47 2021 -0400

    Update Report.md

commit a6731f55f741f095d1861a9610cd6b6a4041b40d
Author: Mustafa Eyceoz <maxusmusti@gmail.com>
Date:   Mon Apr 26 22:52:52 2021 -0400

    Update Report.md

commit 82d5136395ad1f93dddaf23c56573051dcfcceba
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Mon Apr 26 22:51:20 2021 -0400

    moved from google doc to markdown

commit f825d7379c87bf9d6747d17f09c71cc986b4326f
Merge: eab8435 3fa0bb7
Author: treygilliland <jlg2266@columbia.edu>
Date:   Mon Apr 26 22:40:21 2021 -0400

    Merge branch 'main' of https://github.com/raghavmecheri/viper into main

commit 3fa0bb742ec5fd14391297bfbd1a90b192a1ed5e
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Mon Apr 26 21:59:31 2021 -0400

    Headers done

commit eab8435180463b3317ed9f263e0cd831fbf1a42a
Author: treygilliland <jlg2266@columbia.edu>
Date:   Mon Apr 26 21:54:24 2021 -0400

    Finishing touches on demos

commit 16d2cc6ded7706ba8290edd4fd9ca055b224d595
Merge: 6b3b71d aeeb16f
Author: Raghav Mecheri <raghav.mecheri@columbia.edu>
Date:   Mon Apr 26 19:44:28 2021 -0600

    Merge branch 'main' of github.com:raghavmecheri/succotash into main
```

```
commit 6b3b71d1f34803cc1f486ea8d1cb86116cd45498
Author: Raghav Mecheri <raghav.mecheri@columbia.edu>
Date:   Mon Apr 26 19:44:26 2021 -0600

    Report

commit aeeb16f42c9bea101390e811dd6a963e2332a3a2
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Mon Apr 26 21:44:05 2021 -0400

    Added testing and architecture

commit 684a81119f810dbe6c73c8c9778f176642d12aa6
Author: treygilliland <jlg2266@columbia.edu>
Date:   Mon Apr 26 21:43:30 2021 -0400

    Updated Makefile

commit 30c3e43fe5b0473e948e4b3690e40c4dd38759fe
Author: Raghav Mecheri <raghav.mecheri@columbia.edu>
Date:   Mon Apr 26 19:43:03 2021 -0600

    Updated presentation

commit eb774d5a72aad18a0a55510e3ef647ba5cea8b96
Author: treygilliland <jlg2266@columbia.edu>
Date:   Mon Apr 26 21:41:01 2021 -0400

    Updated README

commit 626483c718f996e9543446258fd49ee1895a14a6
Merge: 272ca16 e181863
Author: treygilliland <jlg2266@columbia.edu>
Date:   Mon Apr 26 21:38:24 2021 -0400

    Merge branch 'main' of https://github.com/raghavmecheri/viper into main

commit 272ca165796f34a4fb3cb9340ddaa0892949f957
Author: treygilliland <jlg2266@columbia.edu>
Date:   Mon Apr 26 21:38:16 2021 -0400

    Prettified the demos, added int list prints

commit e181863e71de83070803d06182bd4f19bbb2e769
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Mon Apr 26 21:28:35 2021 -0400

    Added LRM in project timeline

commit 6a0e6617d7dbe5f3e40188af92d8c646001c45e8
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Mon Apr 26 21:26:03 2021 -0400
```

```
    Project plan mostly done, must do project log

commit e179bd716d1ada9c55d163f558b329e4e5390818
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Mon Apr 26 21:16:34 2021 -0400

    Newline

commit cf8d9196dc87731b44929f050e9ccbc839e1a732
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Mon Apr 26 21:14:57 2021 -0400

    Fixed newline for team responsibilities

commit c8fdbea382fc8bdefc160d3dd21d487f6b27ae6d
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Mon Apr 26 21:11:03 2021 -0400

    Added project plan

commit ec25d7e21772d090b7703f63a45ebbb6d96bd4cc
Merge: 85d5f4e ce76bce
Author: Raghav Mecheri <raghav.mecheri@columbia.edu>
Date:   Mon Apr 26 18:48:05 2021 -0600

    Merge branch 'main' of github.com:raghavmecheri/succotash into main

commit 85d5f4e9f52508a972d8ea67ed89ec0d4c158bd9
Author: Raghav Mecheri <raghav.mecheri@columbia.edu>
Date:   Mon Apr 26 18:47:56 2021 -0600

    Bump slides + add comments at top of file

commit ce76bce46569f693b753aa068a9ba0e41645332e
Author: treygilliland <jlg2266@columbia.edu>
Date:   Mon Apr 26 20:44:18 2021 -0400

    Started documenting codegen amongst A BUNCH OF CHANGES

commit 8d0d10f3605ffbcd347f37aad1b860733c3796af
Author: Mustafa <maxusmusti@gmail.com>
Date:   Mon Apr 26 20:40:24 2021 -0400

    Unique fix

commit 111f51bb97b8780701af6d4b7d5ca8ad3fc30c6f
Author: treygilliland <jlg2266@columbia.edu>
Date:   Mon Apr 26 19:46:43 2021 -0400

    Updated list printing functionality
```

```
commit 7009eaf013d8e0f32a0a43874b9b14f6298d6f5a
Author: treygilliland <jlg2266@columbia.edu>
Date:   Mon Apr 26 19:43:58 2021 -0400

    another mergey boy

commit 3e600308f67516012af538f9f998c36c6554e6fc
Merge: 6183ad4 2d6bd7c
Author: treygilliland <jlg2266@columbia.edu>
Date:   Mon Apr 26 19:43:00 2021 -0400

    mergey

commit 6183ad49cd63f37f4fce551edb50fc1414ad445b
Author: treygilliland <jlg2266@columbia.edu>
Date:   Mon Apr 26 19:41:42 2021 -0400

    Added keys functions, print for string lists, etc.

commit 2d6bd7c6a3fdfb6ae3fb1e5bdd68fdee7fd499f6
Author: Raghav Mecheri <raghav.mecheri@columbia.edu>
Date:   Mon Apr 26 17:05:09 2021 -0600

    Removed semanttest folder + added contents.py

commit 1fba933ada7c5b7d04ee5114edf6b098d7c0f0bb
Merge: 58b9aea 1e8e52c
Author: Raghav Mecheri <37787004+raghavmecheri@users.noreply.github.com>
Date:   Mon Apr 26 18:51:39 2021 -0400

    Merge pull request #56 from raghavmecheri/fix/parser

    Merged parser

commit 1e8e52c79b7f88790374acbac2b7c93cf27c4b2f
Author: treygilliland <jlg2266@columbia.edu>
Date:   Mon Apr 26 18:49:58 2021 -0400

    Wagu u silly boy

commit 41e9f5db01720b1aa50a2375fbae112036218fce
Author: treygilliland <jlg2266@columbia.edu>
Date:   Mon Apr 26 18:46:22 2021 -0400

    Added comment for bug

commit a737f61cf2da86fc32563837c720005636dba569
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Mon Apr 26 22:44:45 2021 +0000

    Fixed wordcount shit
```

```
commit fd54573d60544f833e46c96dc000e19279b40528
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Mon Apr 26 22:38:17 2021 +0000

    Decfor fix

commit 632f5109a1eeec56542eb3ef2e657be30e24a7b1
Author: treygilliland <jlg2266@columbia.edu>
Date:   Mon Apr 26 18:34:47 2021 -0400

    DEEP IN BUG WORLD BOYOS

commit 94c4fcde064ac9d51658028828a9109dbaebaff0
Merge: 7fdbf7e 58b9aea
Author: treygilliland <jlg2266@columbia.edu>
Date:   Mon Apr 26 18:05:54 2021 -0400

    Merge branch 'main' of https://github.com/raghavmecheri/viper into fix/parser

commit 58b9aea51f4ed67ce67e7540947df54571d1441e
Author: treygilliland <jlg2266@columbia.edu>
Date:   Mon Apr 26 17:59:34 2021 -0400

    Committing before branch switch

commit 7fdbf7e8c78ab8bd7ceb9ee7c19625f4c0fc47dd
Merge: 94be1d9 f8dc246
Author: Raghav Mecheri <raghav.mecheri@columbia.edu>
Date:   Mon Apr 26 15:58:57 2021 -0600

    Merge branch 'main' of github.com:raghavmecheri/succotash into fix/parser

commit 94be1d944304b9b76156abfef6dfc7855137764f
Author: Raghav Mecheri <raghav.mecheri@columbia.edu>
Date:   Mon Apr 26 15:58:52 2021 -0600

    Parser fix?

commit e0170fbf50e7f9b6a645b439498eca3cf7680be1
Author: Raghav Mecheri <raghav.mecheri@columbia.edu>
Date:   Mon Apr 26 15:58:41 2021 -0600

    Updated pres

commit f8dc246e76b927a0557394de5de2982cb82d5153
Author: treygilliland <jlg2266@columbia.edu>
Date:   Mon Apr 26 17:32:42 2021 -0400

    Yet again, I have added SO MUCH, contains for all lists works

commit 9f4b333090989cbb7fa3a7deb3d4fe9d86aa2535
Merge: a362a49 7912de9
```

```
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Mon Apr 26 21:12:43 2021 +0000

    Merge branch 'main' of https://github.com/raghavmecheri/viper into main

commit a362a496bd927743e49a3b3f6415bd48f45d6e32
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Mon Apr 26 21:12:28 2021 +0000

    Cleanup

commit 7912de9c84decbb86d1d9da6552381a954cf5417
Author: Mustafa <maxusmusti@gmail.com>
Date:   Mon Apr 26 17:09:10 2021 -0400

    Added contains for dicts

commit 270da9793dd3b9d35a3d794f3f7256af9c2443b8
Author: Mustafa <maxusmusti@gmail.com>
Date:   Mon Apr 26 16:21:56 2021 -0400

    REMOVE DICT KEYS WORKS

commit 081799e78bc9b0d143e3232be8434fe1a78c01be
Merge: fc6b470 0a473c9
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Mon Apr 26 20:17:06 2021 +0000

    Merge branch 'fix/ternary_desugar' into main

commit 0a473c9eb1cec1e4a89bb0203a1109b44ced1388
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Mon Apr 26 20:16:47 2021 +0000

    Fixes

commit fc6b4702433be49d1b49d1ce97d211f280c0020f
Merge: a25fe53 1bc36ee
Author: Raghav Mecheri <37787004+raghavmecheri@users.noreply.github.com>
Date:   Mon Apr 26 16:00:53 2021 -0400

    Merge pull request #55 from raghavmecheri/fix/ternary_desugar

    Fix/ternary desugar

commit a25fe5390f4d3364b50a8c806ed1ead19800883c
Author: treygilliland <jlg2266@columbia.edu>
Date:   Mon Apr 26 15:59:46 2021 -0400

    Added literally so much, all lists work (wow), string dicts (holy), this is great

commit 1bc36ee5c256301607d41d4a8d734757c0b5f778
```

```
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Mon Apr 26 19:56:22 2021 +0000

    Fixed shit

commit ed448be16e8eb3c07e36469c2a9f0257fd54b2ab
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Mon Apr 26 19:47:30 2021 +0000

    Updated demo

commit 57aae1ab4612a426b724701d51fe98c6ad265d90
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Mon Apr 26 15:38:06 2021 -0400

    Fixed ternaries

commit 1fd65ff843f3d37f7466945c0220d583e1b4b8c9
Author: treygilliland <jlg2266@columbia.edu>
Date:   Mon Apr 26 13:31:15 2021 -0400

    Added 2 new tests, character list printing, reformated print standard lib call

commit 7b5a16cb43f45e894ae1814761a52449453961e4
Author: Raghav Mecheri <raghav.mecheri@columbia.edu>
Date:   Mon Apr 26 11:29:37 2021 -0600

    Added wordcount demo

commit 290f02b583397dff9302b80383ff2427ba7e7e0d
Merge: d9b62f9 b8c6f49
Author: Raghav Mecheri <raghav.mecheri@columbia.edu>
Date:   Mon Apr 26 11:09:30 2021 -0600

    Merge branch 'main' of github.com:raghavmecheri/succotash into main

commit bcdee53fd70ffcb96f3857985b562a1d5dd4a7f9
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Mon Apr 26 17:08:04 2021 +0000

    Renamed semant stuff

commit 9ebe3e9643d257b21ec1bc2ff4f64fd4b6bd5de8
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Mon Apr 26 15:10:46 2021 +0000

    Removed call to runtests from exec

commit b4eb48ae039dfe1113c778dc469e8f231954e5fa
Merge: 2cb49f2 b8c6f49
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Mon Apr 26 15:09:06 2021 +0000
```

```
    Merge branch 'main' of https://github.com/raghavmecheri/viper into
fix/ternary_desugar

commit 2cb49f283f736df77333f491c7fb5d1017f4d6f4
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Mon Apr 26 15:03:04 2021 +0000

    Fixed nested ternary

commit 413c892f43c894587aeaf6da076262e8c6f4be0d
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Mon Apr 26 14:45:09 2021 +0000

    Fixed semant for decassign + ternop

commit b8c6f4923ee820579fdb6bebf2c8bcb04b4f9cc3
Author: Mustafa <maxusmusti@gmail.com>
Date:   Mon Apr 26 10:35:06 2021 -0400

    Added .keys() to dicts

commit 39ece16e7fd4c03f91deff23404920067fc744c9
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Mon Apr 26 14:21:53 2021 +0000

    Broken code

commit 1526114ab4637f033c43920152670a7b0c00fd79
Merge: 55ba632 f63ac08
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Mon Apr 26 01:47:32 2021 -0700

    Merge branch 'main' of https://github.com/raghavmecheri/viper into main

commit 55ba6328e112f7d84b6a04ef8d405aad8e8a351a
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Mon Apr 26 01:47:16 2021 -0700

    beef up test suite

commit f63ac08d092384b722f745b7655b6285709a9c0d
Author: treygilliland <jlg2266@columbia.edu>
Date:   Mon Apr 26 04:37:58 2021 -0400

    Added Incr/Decr

commit 3f063727beadf63c3e5b9eb63f5596dafbfadaf4
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Mon Apr 26 00:58:49 2021 -0700

    fix for loops
```

```
commit b285d4cb0f70870f3231369fac4637d2c13890fc
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Mon Apr 26 00:25:15 2021 -0700

    support dict/array literals

commit 04e6141080ba571f504f3e6c8e29297f394a6d4f
Merge: 037d7e9 b5bc4ca
Author: treygilliland <jlg2266@columbia.edu>
Date:   Mon Apr 26 02:14:34 2021 -0400

    Merge branch 'main' of https://github.com/raghavmecheri/viper into main

commit 037d7e9580c4aef41c72dd458a7a99f0994b1c59
Author: treygilliland <jlg2266@columbia.edu>
Date:   Mon Apr 26 02:12:54 2021 -0400

    Working on nested dicts

commit b5bc4cac40adeb13781b9991b47f181315c7275a
Merge: 277da09 2179365
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Sun Apr 25 23:03:02 2021 -0700

    Merge branch 'main' of https://github.com/raghavmecheri/viper into main

commit 277da0972b8aaba9848d5b6ecfc703430b7da18d
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Sun Apr 25 23:02:57 2021 -0700

    loop fixes

commit 21793654eee3fb267e3c2a2212031afe1e04583c
Author: Mustafa <maxusmusti@gmail.com>
Date:   Mon Apr 26 01:27:02 2021 -0400

    Removed Groups from Report

commit 58daf8a68876d1406403d1132274d3ab54b7114d
Author: Mustafa <maxusmusti@gmail.com>
Date:   Mon Apr 26 01:24:25 2021 -0400

    Added goal section

commit 3b2d7fcaefb666ba31ea78b8f6a4ba40da4b9e50
Author: treygilliland <jlg2266@columbia.edu>
Date:   Mon Apr 26 00:55:23 2021 -0400

    I finally understand how the llvm bindings work, char: int dicts work lol

commit ce645221ef7803bd7d9e6feb388c443ab1b163db
```

```
Merge: d5d87e2 12fc99a
Author: treygilliland <jlg2266@columbia.edu>
Date:   Sun Apr 25 23:37:01 2021 -0400

    Merge branch 'main' of https://github.com/raghavmecheri/viper into main

commit d5d87e27e2df34449da6879b266c9c9d2df3f6e4
Author: treygilliland <jlg2266@columbia.edu>
Date:   Sun Apr 25 23:36:59 2021 -0400

    Added Str library to viper.native and working on dicts. about to mess with access

commit 12fc99a9290f1377d4c44741fc9a75315983e94f
Author: Mustafa <maxusmusti@gmail.com>
Date:   Sun Apr 25 23:20:31 2021 -0400

    Goal Headers

commit c0cedc2f754dd9919a313a051db0eeeee23b131a
Author: Mustafa <maxusmusti@gmail.com>
Date:   Sun Apr 25 23:06:08 2021 -0400

    Finished how to use secion

commit 8fd592d4a01c63d42d6865a8ac95ce9b91e242f4
Author: Raghav Mecheri <raghav.mecheri@columbia.edu>
Date:   Sun Apr 25 21:37:39 2021 -0400

    Attempt for framework for desugared ternaries

commit 5e212c381e0524810b7467e342d69eb11ae80f61
Merge: 7f6a4be 8bce824
Author: treygilliland <jlg2266@columbia.edu>
Date:   Sun Apr 25 19:20:59 2021 -0400

    Merge branch 'main' of https://github.com/raghavmecheri/viper into main

commit 7f6a4be80bac4d0b9eefe18cfc939cfb501e58a5
Author: treygilliland <jlg2266@columbia.edu>
Date:   Sun Apr 25 19:20:55 2021 -0400

    Added dict functions

commit 8bce824a097ba5afe1d2e80aa73f23879618deee
Author: Mustafa <maxusmusti@gmail.com>
Date:   Sun Apr 25 18:30:01 2021 -0400

    Use pt. 1

commit 0769781081b5c6e55e3a629cf860a62f46994e4d
Author: Mustafa <maxusmusti@gmail.com>
Date:   Sun Apr 25 18:18:25 2021 -0400
```

```
    Added related work section

commit 929bc2696d3f3cc14896ed2a7c7d3166b78f78ee
Merge: 1b0f2db d2b880f
Author: Mustafa <maxusmusti@gmail.com>
Date:   Sun Apr 25 17:34:59 2021 -0400

    Added background to final report

commit 1b0f2dbf6e56fba945462d7fdff13730fc599a58
Author: Mustafa <maxusmusti@gmail.com>
Date:   Sun Apr 25 17:34:49 2021 -0400

    Added background

commit d2b880f06b795b8658a57081701a25aab6a881c1
Merge: 19365fc 6dc2822
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Sun Apr 25 14:26:33 2021 -0700

    Merge branch 'main' of https://github.com/raghavmecheri/viper into main

commit 19365fcb6ed6c6942088f174f28d81ca15357b4a
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Sun Apr 25 14:26:23 2021 -0700

    test nested skip and while statements

commit f22a17e2d2d77a5c615be9691198817169d6f0d0
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Sun Apr 25 14:26:11 2021 -0700

    desugar blocks of if and while statements

commit e519f8c51769103ba0b890fd7a72f3b84c36c14f
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Sun Apr 25 14:24:51 2021 -0700

    include empty print statements that just print new lines

commit 6dc2822d785b10e850d0135da13baa2438196b0d
Author: Mustafa <maxusmusti@gmail.com>
Date:   Sun Apr 25 17:08:27 2021 -0400

    Added final report file

commit bdd3be0e764fa91e112c652212befc763e0e86a6
Author: Raghav Mecheri <raghav.mecheri@columbia.edu>
Date:   Sun Apr 25 10:54:54 2021 -0400

    Skeleton code
```

```
commit d9b62f98799451de3237576bdbb948bdbb473652
Author: Raghav Mecheri <raghav.mecheri@columbia.edu>
Date:   Sun Apr 25 10:54:36 2021 -0400

    Presentation updates

commit 769db7b51e2f68d53e46b2ccab85482fdb037d7d
Merge: ffe3617 bb3c8bf
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Sun Apr 25 00:22:56 2021 -0700

    Merge pull request #54 from raghavmecheri/feature/abort

    implement abort

commit bb3c8bf213fa83f58e79a5e440be36908ce83d1c
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Sun Apr 25 00:22:09 2021 -0700

    implement abort

commit ffe3617583e0c8e84a04991e8209a79a5c928157
Merge: 6778a50 65816f1
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Sat Apr 24 23:11:35 2021 -0700

    Merge pull request #52 from raghavmecheri/feature/skip

    Feature/skip

commit 65816f10cd4cbdbf16a3c18ff81cc15fb84e010f
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Sat Apr 24 23:10:09 2021 -0700

    skip tests

commit 3b99a753452a36f3cb908ca6ee7cf73c1dc75c7b
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Sat Apr 24 23:07:10 2021 -0700

    implement basic block logic for skip

commit 7c48da88467b6a008a6a44bb6959b1160b197aae
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Sat Apr 24 23:06:57 2021 -0700

    update definition of While()

commit 1c12af775de2e4212a3a8bdc9861a4ed47ca22e5
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Sat Apr 24 23:06:29 2021 -0700
```

```
    include secret incrementer in while, used for skip

commit 358d8e5a8941fe6e25e4e7dcb745c37a2537476a
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Sat Apr 24 19:12:39 2021 -0700

    update while to take 3rd param for incrementation

commit 6778a50f3cefcec7f1a61d3c5f49610bcdad5752
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Sat Apr 24 22:39:24 2021 +0000

    Fixed exec code

commit bd0027d5d063b90b514f572e61b58c34a423554b
Merge: f7d1c77 a448c9f
Author: treygilliland <jlg2266@columbia.edu>
Date:   Sat Apr 24 18:22:47 2021 -0400

    Merge branch 'main' of https://github.com/raghavmecheri/viper into main

commit f7d1c77647744600ee30b55235d90d973bf5eaca
Author: treygilliland <jlg2266@columbia.edu>
Date:   Sat Apr 24 18:22:38 2021 -0400

    got more demos working

commit a448c9fe4dcdd6c3ffe1ad91e0359e966eb00122
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Sat Apr 24 22:03:08 2021 +0000

    Removed random top level files

commit e22f285195b971343b01419a960aabcf7db8be8b
Author: treygilliland <jlg2266@columbia.edu>
Date:   Sat Apr 24 17:14:48 2021 -0400

    removed print, added contains and len

commit ff7e843f9f31f984e57826f91dc8cbf557c06ea7
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Sat Apr 24 21:12:56 2021 +0000

    Removed some prints

commit 2962205e08de2675dc79635178ff566f19ed1b69
Merge: e0b2a11 28cef50
Author: treygilliland <jlg2266@columbia.edu>
Date:   Sat Apr 24 17:02:25 2021 -0400

    Merge branch 'main' of https://github.com/raghavmecheri/viper into main
```

```
commit e0b2a11feadc2ba268fceff805bb4f3ee65c237e
Author: treygilliland <jlg2266@columbia.edu>
Date:   Sat Apr 24 17:02:04 2021 -0400

    Adding exec.sh, use like ./exec.sh file.vp

commit 28cef50cc9cab716bd6e4f7afd0525b85e0785a1
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Sat Apr 24 21:00:40 2021 +0000

    Added binse demo

commit d4a8652e9936989d1913ee893ed74e8e25d56d96
Merge: 5a034ad 40f0da8
Author: treygilliland <jlg2266@columbia.edu>
Date:   Sat Apr 24 16:42:06 2021 -0400

    Merge remote-tracking branch 'origin/fix/builtin' into main

commit 40f0da89ccbd625f4fc74e41454de33bffced377
Author: treygilliland <jlg2266@columbia.edu>
Date:   Sat Apr 24 16:37:33 2021 -0400

    append function fixed?

commit 5a034ad823ae96c7263791aa0f36b1dc5547e907
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Sat Apr 24 20:21:41 2021 +0000

    Jank ternaries + demo update

commit 85ecf76d399ffc1343dac615e7c24701445e3a64
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Sat Apr 24 13:13:51 2021 -0700

    fix multiple function args

commit 8a6e39058bf03166f28affb473615c8304187264
Merge: 685c480 60be2be
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Sat Apr 24 19:25:20 2021 +0000

    Merge branch 'main' of https://github.com/raghavmecheri/viper into main

commit 685c480038a0383c08aba232db1ce1093e1b9368
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Sat Apr 24 19:25:15 2021 +0000

    Working demo programs

commit 2412a70c6575ec157d048a068ac216914d8476ca
```

```
Merge: fed254c 60be2be
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Sat Apr 24 12:19:46 2021 -0700

    Merge branch 'main' of https://github.com/raghavmecheri/viper into fix/builtin

commit 60be2bed0f6bb9d7ce49fa872b7d2f4b685f6cc0
Merge: 6ddfe9c faf6ff0
Author: treygilliland <jlg2266@columbia.edu>
Date:   Sat Apr 24 15:17:00 2021 -0400

    Merge branch 'main' of https://github.com/raghavmecheri/viper into main

commit 6ddfe9c71151903fd34a6b8d8ce0079858b05a09
Author: treygilliland <jlg2266@columbia.edu>
Date:   Sat Apr 24 15:16:53 2021 -0400

    Updated append to return void

commit fed254c09559c76b65bff2696b8f418eb757ab4e
Merge: c8244e7 faf6ff0
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Sat Apr 24 12:04:53 2021 -0700

    Merge branch 'main' of https://github.com/raghavmecheri/viper into feature/skip

commit faf6ff0b828f6b693fe8ad3a6cc1494a3f071355
Merge: fe12548 025cd14
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Sat Apr 24 18:54:34 2021 +0000

    Merge

commit 025cd145170ea6b49072443be15a49db3174e97c
Merge: 5a863a8 7088cd7
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Sat Apr 24 11:54:03 2021 -0700

    Merge pull request #51 from raghavmecheri/feature/ternary

    Feature/ternary

commit 7088cd748a6535edf5648bb9aca8cd0c3a87a23f
Merge: d2a8b5e 5a863a8
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Sat Apr 24 11:53:58 2021 -0700

    Merge branch 'main' into feature/ternary

commit fe1254863a891fa5c0a7d40f653766e3cd8de05f
Merge: de16cdd 5a863a8
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
```

```
Date:   Sat Apr 24 18:53:48 2021 +0000

    Merge branch 'main' of https://github.com/raghavmecheri/viper into main

commit d2a8b5e777b746c17111f3b51f9e401e2bf2f74e
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Sat Apr 24 11:52:46 2021 -0700

    finish ternary check

commit de16cdd5eb7e2f1bbd68b62d269b899b764a108a
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Sat Apr 24 18:48:48 2021 +0000

    Fixed ternary shit

commit 5a863a838b1889f4722b8c5e57e10f06051ef761
Merge: 2e8e07c c708875
Author: treygilliland <jlg2266@columbia.edu>
Date:   Sat Apr 24 14:44:00 2021 -0400

    resolved merge

commit 2e8e07c3775f32cc27c694263beab084504e368d
Author: treygilliland <jlg2266@columbia.edu>
Date:   Sat Apr 24 14:43:10 2021 -0400

    LISTS WORKING

commit 4dad6a2744690d5663d23d387db1164eeac02f4a
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Sat Apr 24 11:27:51 2021 -0700

    add semantic ternary operators

commit c7088753a2590d8b20e19f2f08f3eb957a470893
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Sat Apr 24 18:18:50 2021 +0000

    Demo folder (kinda)

commit c8244e751d04e273033453cb61bbc4211bd2db68
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Sat Apr 24 11:15:31 2021 -0700

    skip works, but loops need to be revisited

commit 3e4d87ffe198f6214430be1ec5ee06fbe3a5acf7
Merge: d79a1f9 8c0108b
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Sat Apr 24 17:21:06 2021 +0000
```

```
    Merge

commit d79a1f90cd8d0dd9ad78809d17af39c590e577c4
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Sat Apr 24 17:20:37 2021 +0000

    Manipulated crap

commit c48fff228dbffae9d0a391e79bb59e4475df3654
Merge: 0030723 8c0108b
Author: treygilliland <jlg2266@columbia.edu>
Date:   Sat Apr 24 12:40:56 2021 -0400

    Merge branch 'main' of https://github.com/raghavmecheri/viper into main

commit 0030723b686d60869e20a0358eebfcac42af85ca
Author: treygilliland <jlg2266@columbia.edu>
Date:   Sat Apr 24 12:40:43 2021 -0400

    Hella updates

commit 8c0108b57fc42fb1c3abb6f599f5a9c5e3ee0248
Author: Mustafa <maxusmusti@gmail.com>
Date:   Sat Apr 24 12:09:21 2021 -0400

    List pretty print

commit 3932dd166815772bdac034804b2707fee7abde0d
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Sat Apr 24 12:01:32 2021 -0400

    built-in functions now can have multiple parameters

commit c39d294e526e4af1b928c02722746ee6eb164c89
Merge: 1958e74 1d01702
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Sat Apr 24 11:37:09 2021 -0400

    Merge branch 'main' of https://github.com/raghavmecheri/viper into main
    Merge nested Dictionary

commit 1958e748cac366572abf42af2770af250410e36e
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Sat Apr 24 11:36:54 2021 -0400

    Fixed nested dictionary

commit 1d017023d4c8ad7ad16832a45088291f3b2e6d33
Author: Mustafa <maxusmusti@gmail.com>
Date:   Sat Apr 24 10:40:08 2021 -0400

    Fix ternops
```

```
commit 7244e12c0d78b3f8039ed62c40e65e844fcbf6d9
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Sat Apr 24 10:32:55 2021 -0400

    decs fixed

commit 7eed86a73f9e4fd6f645917e544ca76127d0531e
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Sat Apr 24 10:31:20 2021 -0400

    decs

commit 7aee7b078158588b1b2ede9c8f2e09beed0c9339
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Sat Apr 24 14:07:16 2021 +0000

    Fixes

commit d6ee2b6cb02ed7cb7707b248471436f0796d5b3d
Merge: 553275e 1d62dc1
Author: Raghav Mecheri <37787004+raghavmecheri@users.noreply.github.com>
Date:   Sat Apr 24 08:46:41 2021 -0400

    Merge pull request #50 from raghavmecheri/fix/sugaring

    Fix/sugaring

commit 1d62dc10129bd5471ce50670aaa371f696eb9901
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Sat Apr 24 12:41:03 2021 +0000

    Added functional recursion to desugar

commit 553275eed0b147ddbe70b333c0df4c0bd0893700
Author: treygilliland <jlg2266@columbia.edu>
Date:   Sat Apr 24 07:48:23 2021 -0400

    Added more tests back into LLVM

commit 21832f7fac796076a0c8ab19060ccaa1ba23e3b3
Merge: 5cded20 7b64814
Author: Trey Gilliland <treygilliland3@gmail.com>
Date:   Sat Apr 24 07:17:23 2021 -0400

    Merge pull request #49 from raghavmecheri/big-boy

    BIG BOY COMMIT

commit 7b64814ed8ad3dea631d1d09991ef25b6d100fdb
Merge: 7fda776 5cded20
Author: Trey Gilliland <treygilliland3@gmail.com>
```

```
Date:   Sat Apr 24 07:15:07 2021 -0400

    Merge branch 'main' into big-boy

commit 7fda77670a2812a9cc2855e30d3b5920d31d6fed
Author: treygilliland <jlg2266@columbia.edu>
Date:   Sat Apr 24 07:09:59 2021 -0400

    BIG BOY COMMIT

commit 346c71383c7ba241c0338c5167785068b14f70e4
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Sat Apr 24 11:09:18 2021 +0000

    Sugaring fix

commit 5cded20b5ae433813b2b666b434363797183a32b
Merge: 394a485 6756f95
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Sat Apr 24 01:09:47 2021 -0700

    Merge pull request #48 from raghavmecheri/feature/codegen-builtin

    Type casting on Type Literals

commit 394a485fe88060a06c7c71e1d144812f99c8bafd
Author: treygilliland <jlg2266@columbia.edu>
Date:   Sat Apr 24 03:49:22 2021 -0400

    Added mod

commit 6756f95035fd5cde5401c7e36877b78872e28d58
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Sat Apr 24 00:19:28 2021 -0700

    tester file for casting

commit 48cab3413bf082f72cab681115a01b69faf6155f
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Sat Apr 24 00:19:19 2021 -0700

    create defs for casting funcs

commit 7dde8427e9f49a4f0c19b0a1232087993d1323cb
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Sat Apr 24 00:19:04 2021 -0700

    additional changes

commit df304f81769fe4e9fb44f59f15b53a2c9329f49e
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Sat Apr 24 00:18:25 2021 -0700
```

```
    match for casting calls in codegen

commit 1cf81fb140d59f25322ab4951ddfa830b0df13e8
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Sat Apr 24 00:18:09 2021 -0700

    create initial casting functions

commit 02298410f2c1bef4071154f44ccd3f1b6e39f50b
Author: treygilliland <jlg2266@columbia.edu>
Date:   Sat Apr 24 02:56:39 2021 -0400

    Adding in pow, in progress

commit 559a71ee5902d087cbbc15b46ab231b48252aaac
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Fri Apr 23 22:22:10 2021 -0700

    change names of casting functions

commit 428fee6afefb0eed7a68caa076446db7bf6a2b2c
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Fri Apr 23 21:50:42 2021 -0700

    finish casting funcs

commit 0dd2f3d863c540ee1b5affe8f1132dd6bdda9290
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Fri Apr 23 18:35:44 2021 -0700

    implement int casting

commit 34ca929f01d8ed1c85e50d0ecec7cd0af2a84e94
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Fri Apr 23 18:08:29 2021 -0700

    implement to_char

commit b3f347e96aab6048d60857d738d71b1e60f7b6fd
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Fri Apr 23 18:00:37 2021 -0700

    Update char() in LRM

commit 6486009c77647932e42ce1c7e63e7dbc1ff9f120
Merge: 7d3de7f 7dbbe03
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Fri Apr 23 17:04:19 2021 -0700

    merge
```

```
commit 7d3de7fef90648fa78e3e04fa0b28962fec56122
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Fri Apr 23 17:03:25 2021 -0700

    djijkstra update

commit 7dbbe03b69220b4745adeef8a70877eea1417f0d
Author: treygilliland <jlg2266@columbia.edu>
Date:   Fri Apr 23 19:32:21 2021 -0400

    Added lvalue initial value type resolver

commit b9d4ae925bb332582b4890f5e589b566896945fd
Author: treygilliland <jlg2266@columbia.edu>
Date:   Fri Apr 23 19:02:15 2021 -0400

    Added all Viper types to ltype func

commit 3965b43a388db92b1c68ba41cbc87bc257ce2c4c
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Fri Apr 23 15:33:23 2021 -0700

    forgot a semicolon

commit 40b68da5c582a471528f51ac88fc74d5c25641f7
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Fri Apr 23 15:31:17 2021 -0700

    updated dijkstra demo

commit 587d9d3fe0b1348116d007481c3e3bd8ae097e4c
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Fri Apr 23 14:43:21 2021 -0700

    clean up decs

commit f0d06f0ebdbc49110d66dd672028d65e67ce59ed
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Fri Apr 23 14:28:27 2021 -0700

    clean up semant driver

commit a5275c4905104b3ad3310590cbb89025bbfd37eb
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Fri Apr 23 14:16:02 2021 -0700

    allow for empty dict literals

commit 158b94b0c1b9c9218cd969139f5c9f44e08f99bc
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Fri Apr 23 14:09:45 2021 -0700
```

```
    allow declarations of empty list literals

commit 4522685bc8cec32d1bcd32625678c7c0f432fa98
Merge: 5e36b70 c085326
Author: Raghav Mecheri <raghav.mecheri@columbia.edu>
Date:   Thu Apr 22 18:57:18 2021 -0400

    Merge branch 'main' of github.com:raghavmecheri/succotash into main

commit 5e36b70f78c30a519cddd8e77eaadbf042046a84
Author: Raghav Mecheri <raghav.mecheri@columbia.edu>
Date:   Thu Apr 22 18:57:14 2021 -0400

    Added initial; presentation

commit c0853261a3ca2eaa91132be49094cfa4e46c74f6
Merge: fca18f2 48b97a5
Author: Mustafa Eyceoz <maxusmusti@gmail.com>
Date:   Thu Apr 22 18:53:38 2021 -0400

    Fixed char:* dicts

    Fixed char:* dicts

commit 48b97a5de9f4c540401f35ee88d949cbe7ffceb8
Author: Mustafa <maxusmusti@gmail.com>
Date:   Thu Apr 22 18:52:42 2021 -0400

    Fixed char:int dicts

commit fca18f29c1200f25ac83016040cd2e9644630aba
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Thu Apr 22 13:43:24 2021 -0400

    fixed main statement order

commit c3510cff1742c4b93317cd337621c31e87b2b66f
Author: treygilliland <jlg2266@columbia.edu>
Date:   Thu Apr 22 13:41:30 2021 -0400

    Fixed empty syntax error for print type resovler

commit 4ea3cadf875a4e5cef258f423bb7a70641af8378
Author: treygilliland <jlg2266@columbia.edu>
Date:   Thu Apr 22 13:20:24 2021 -0400

    Updated print type resolver, added two .vp files for semantic bug

commit d2263c2852c972bdfa052718591b666647e8056c
Merge: 36bd54a 0af0a39
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Thu Apr 22 10:23:37 2021 -0400
```

```
    Merge branch 'main' of https://github.com/raghavmecheri/viper into main
    Merge for codegen

commit 36bd54ab5a2e43a0e3eda2e713b3662f21af1308
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Thu Apr 22 10:23:17 2021 -0400

    fixed built in function declarations

commit 0af0a39303861f8c6511d1644fcb0b18f379144e
Merge: e28d86a 161a838
Author: Mustafa Eyceoz <maxusmusti@gmail.com>
Date:   Thu Apr 22 09:58:19 2021 -0400

    Dict additions v1

    Dict additions v1

commit 161a8388848369f1538ef385c02a54e9143a69cb
Author: Mustafa <maxusmusti@gmail.com>
Date:   Thu Apr 22 09:56:18 2021 -0400

    Dict additions v1

commit e28d86aba039f2690dac0b63a7f27b8fadc4ce7e
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Thu Apr 22 09:41:44 2021 -0400

    fixed built-in function calls

commit a91934856662726913f23e63a8a115e488834aa7
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Thu Apr 22 09:11:54 2021 -0400

    Function return type fixed for codegen

commit 67bee324a0d27e76ff191f1f9333df6e925ba4e1
Author: treygilliland <jlg2266@columbia.edu>
Date:   Thu Apr 22 04:54:42 2021 -0400

    Added AttributeCall but it is so sketch and I cant test until function calls work

commit dfd12f258df43b86fdcd6ba1165402353b144a1c
Author: treygilliland <jlg2266@columbia.edu>
Date:   Thu Apr 22 04:47:12 2021 -0400

    Added OpAssign

commit 0098083b64dd605ddf2260ae0a184bc1a2f7ed06
Author: treygilliland <jlg2266@columbia.edu>
Date:   Thu Apr 22 04:04:31 2021 -0400
```

```
    Added formals to locals

commit 68dc011cc7bf58b1ec325d0beba18270260cf476
Author: treygilliland <jlg2266@columbia.edu>
Date:   Thu Apr 22 03:40:48 2021 -0400

    Reorganized code and added more statements/expressions we need to implement

commit b8d6ec691b6347cf339e664425053f1b95a2b56f
Author: treygilliland <jlg2266@columbia.edu>
Date:   Thu Apr 22 03:17:27 2021 -0400

    removed duplicate SWhile check, added SWhile and SDecAssign

commit 9effde55925d7e78b9355099ca3751c5f7fafc05
Author: treygilliland <jlg2266@columbia.edu>
Date:   Thu Apr 22 03:03:11 2021 -0400

    added if statements, lowkey sus

commit e1454a357a70a9528173dae7e9b6288a3fa60250
Author: treygilliland <jlg2266@columbia.edu>
Date:   Thu Apr 22 02:46:27 2021 -0400

    Added local variables to functions (SId, SAssign, SDec). removed print from
variable resolution in decs.ml

commit 7b0d1486e2a327c54adf212a5e9c8dd2d78a8367
Author: treygilliland <jlg2266@columbia.edu>
Date:   Thu Apr 22 00:04:51 2021 -0400

    function calls working

commit bdd998fe6103b15f594592a2db85eba2652d7240
Merge: cbd4e4f 5768b52
Author: treygilliland <jlg2266@columbia.edu>
Date:   Wed Apr 21 23:18:38 2021 -0400

    merge? hlp

commit 5768b52ccdfdf2e8bdc818f653b008571cfc5ff6
Merge: ae30857 63b129a
Author: Trey Gilliland <treygilliland3@gmail.com>
Date:   Wed Apr 21 23:08:29 2021 -0400

    Merge pull request #45 from raghavmecheri/feature/mainfunc

    collect global statements into main if main isn't defined

commit 63b129a2da3be4f2102459de9e6081f0cd9e0dc9
Author: Tommy Gomez <tjk2132@columbia.edu>
```

```
Date:   Wed Apr 21 19:57:25 2021 -0700

    collect global stmts into main if main isn't defined

commit cbd4e4f5d051d234dcd321b06fa180f8c2616015
Author: treygilliland <jlg2266@columbia.edu>
Date:   Wed Apr 21 21:55:35 2021 -0400

    Function declarations working

commit 269dce4173281fa5c98ee224d0af947469462188
Author: treygilliland <jlg2266@columbia.edu>
Date:   Wed Apr 21 20:52:50 2021 -0400

    Unbroke the LLVM, going to attempt to add function declarations

commit ae30857947f39f94558d94476fced3f6564d9144
Merge: 702a568 db5db3f
Author: Mustafa Eyceoz <maxusmusti@gmail.com>
Date:   Wed Apr 21 19:28:38 2021 -0400

    Added list assignment

    Added list assignment

commit db5db3f0828d22a0449ae8c88f865fe299b6092c
Author: Mustafa <maxusmusti@gmail.com>
Date:   Wed Apr 21 19:27:06 2021 -0400

    Added list assignment

commit 702a568144430627bb31b60a2321b22dc5ce77d6
Merge: 9b6950d 1c3f226
Author: Mustafa Eyceoz <maxusmusti@gmail.com>
Date:   Tue Apr 20 16:08:33 2021 -0400

    Standard Library

    Standard Library

commit 9b6950d53b5c3491fcd0a7bf6d8cda8508621cf8
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Mon Apr 19 22:48:34 2021 -0400

    disabled warnings 42 and 45; not needed

commit 5e52ab428e9d54b430f15283bd7c2231c174dfe0
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Mon Apr 19 21:56:50 2021 -0400

    fixed applicable warnings and updated todo
```

```
commit 43db32b49d37c8ce92a09a1695db58140a45e298
Merge: b34c8bd b0db03c
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Mon Apr 19 21:50:10 2021 -0400

    Merge pull request #43 from raghavmecheri/feature/semantics2

    Feature/semantics2

commit b837ba28dbe9e81ca780732ed07cc0a3157dacdf
Author: treygilliland <jlg2266@columbia.edu>
Date:   Mon Apr 19 21:47:59 2021 -0400

    Code is very broken, but is on the right path

commit b0db03c518e8c76192f4ee238413988d5f578a45
Merge: 9ef1f48 b34c8bd
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Mon Apr 19 21:47:47 2021 -0400

    Merge branch 'main' into feature/semantics2

commit b34c8bdfc70fb773e4f83d065f6bf831d115f866
Merge: f9db679 97155d9
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Mon Apr 19 20:57:23 2021 -0400

    Merge branch 'main' of https://github.com/raghavmecheri/viper into main
    Merge with Tommy's code

commit f9db679ba83a122e3c50f4bad91caec7277f1efc
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Mon Apr 19 20:57:11 2021 -0400

    Implemented attribute call

commit 97155d904a8514e43198c79a2c08db0ef16f7ee8
Author: Raghav Mecheri <37787004+raghavmecheri@users.noreply.github.com>
Date:   Mon Apr 19 16:41:31 2021 -0400

    Update LRM.md

commit 9ef1f48cf8817ffd033fdd664c76e7f2b07e9059
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Mon Apr 19 12:28:30 2021 -0700

    implement Access checking

commit 6dc3bec68c58a8e15836ad3f8b3c4413c7815cef
Author: treygilliland <jlg2266@columbia.edu>
Date:   Mon Apr 19 15:25:21 2021 -0400
```

```
    Added temp explanatory comments to decs.ml, added SAST pretty printing to viper.ml

commit fce3d48ab1eb59b4d25b4c99e52d48532866fa52
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Mon Apr 19 12:11:37 2021 -0700

    implement accessassign

commit 271137b1c9be5a40592261a16c2fd34de952949e
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Mon Apr 19 11:04:20 2021 -0700

    fix decs merge

commit 91db42e99d66866eeba14fd02c7d169d3f51561a
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Mon Apr 19 10:18:57 2021 -0700

    terminate comment

commit 1c59b0528de0c2a01d9fec7682276ddb1342b88d
Merge: 165ecbe af6be04
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Mon Apr 19 10:16:33 2021 -0700

    merge

commit 165ecbee16ef8abb975e7732e4846289ec2eb720
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Mon Apr 19 09:56:29 2021 -0700

    add dictionary to access assign

commit af6be0409d389de669a382045acaf13c41ea79f1
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Mon Apr 19 12:49:29 2021 -0400

    fixed nah return check

commit a84bc1e45c5b0df89e9730e68508e7054b7dbf09
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Mon Apr 19 12:47:37 2021 -0400

    Checks for existence of return statements in functions and accessassign must be
implemented for dictionaries

commit 6672c335605a931647ac0f7220fa5b3fb6432c3c
Author: treygilliland <jlg2266@columbia.edu>
Date:   Mon Apr 19 11:12:03 2021 -0400

    Started updating codegen to use SAST
```

```
commit 05b87f8320d484f2333f8c7e8a691030a39c94d7
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Mon Apr 19 10:59:29 2021 -0400

    fixed warnings. The warnings that are appearing can be ignored.

commit 1b58099b675884a7b5a9a263c09d3a3ad03d1885
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Mon Apr 19 10:24:12 2021 -0400

    Implemented call in semantic checker

commit 7763dd8a5eb7c84d2e22c0f2178a1ec25ef9e8b5
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Sun Apr 18 23:14:11 2021 -0400

    access assign done, updated todo list, Tommy has to merge

commit b4054c977843f01b2f3a613342690602fb0bf50f
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Sun Apr 18 19:13:20 2021 -0700

    check accessassign

commit bd26b05bed050e6110289e71bb1a74c2278e531d
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Sun Apr 18 18:03:21 2021 -0700

    started deconstruct, ran into problems

commit 9e3a9825fe98163037a764fb764211a1fc2974b3
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Sun Apr 18 20:07:52 2021 -0400

    opassign

commit 7700de56e96fd111039532cbf0b9eeeae93feeef
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Sun Apr 18 19:20:41 2021 -0400

    Semantic checker: Lists, Dictionaries, Built in Function Declarations, Skip,
Panic, Abort

commit 5e6666884b6d1042b2df6d3d4617da53c7fc51f2
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Sun Apr 18 15:50:17 2021 -0700

    check for built in func declarations

commit 077464ee450bf8fdc2c352182ef72d8b868218ac
Merge: 8332a58 add5c7f
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
```

```
Date:   Sun Apr 18 03:56:43 2021 +0000

    Merge branch 'main' of https://github.com/raghavmecheri/viper into main

commit 8332a586d8bd602882a1f20df8250313b52f9265
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Sun Apr 18 03:56:19 2021 +0000

    Added sample code for dijk

commit add5c7f950457a583a9744b8ef66c0c83b2a8ffc
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Sat Apr 17 19:31:28 2021 -0400

    added to todo

commit c99743605e60708d0e4fa8a517e95e7f9d9a99b7
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Sat Apr 17 19:24:19 2021 -0400

    added and updated todo for semantic checker

commit de63090060089992d7819151df0ba6cc0e3acf26
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Sat Apr 17 18:31:39 2021 -0400

    fixed return types for functions

commit aed5a0c59a2bdd5c59b08a7d025ea3d5ed658276
Merge: cdb09f3 566d580
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Sat Apr 17 18:18:53 2021 -0400

    Merge branch 'main' of https://github.com/raghavmecheri/viper into main
    Semantic checker

commit cdb09f37a91635267f390ab10e50e05375c0ab4a
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Sat Apr 17 18:18:33 2021 -0400

    Semantic checker

commit 566d58022e596635a15cf0bea637605751a13823
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Sat Apr 17 22:11:22 2021 +0000

    Added sample code (but in Python)

commit 0f31018017bcfc619655e9a38ff8d7a0af256812
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Sat Apr 17 10:34:23 2021 -0400
```

```
    edited todo and fixed decs in functions

commit 642ba6f6ebdf3f54096731992b4607ddcd8ee568
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Sat Apr 17 00:32:20 2021 -0400

    function return types are flawed and statementlist is including statements in
functionlist bodys

commit 12eee0cbc70d374fad62dca2ccd79e1c81d5fb80
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Sat Apr 17 00:25:15 2021 -0400

    blah

commit 778b21757e3e8131e7673f38d0eab2918eb6a441
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Fri Apr 16 23:02:12 2021 -0400

    fixed scoping for functions, functions not working in general

commit 3bc1f62fd0b09d2f2c02dbb1992a9b350f12f189
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Fri Apr 16 20:21:51 2021 -0400

    updated todo

commit e4c8854c3263bc0f3c4fef1e335fd7546cafb230
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Fri Apr 16 20:16:57 2021 -0400

    updated todo

commit 90ae300391aa9b383793c519985b2cf176006a05
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Fri Apr 16 20:16:20 2021 -0400

    fixed scoping issue, now we can add declarations in while and for loops

commit 741e597eeb6f03fc7dd042ca8d611e77229c5fab
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Fri Apr 16 19:35:09 2021 -0400

    added skip, panic, abort to todo

commit b8e53b9a190e8b94bd4e13ef73e42823b2a3ba0c
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Fri Apr 16 19:20:04 2021 -0400

    added built-in funcdecls in the todolist

commit 8836ec754e25a8259fdbaaf4aeda9949b4132d63
```

```
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Fri Apr 16 19:12:42 2021 -0400

    fixed bugs in semantdriver and wrote some tests for the checker. Todo list was
made.

commit ff83f432bdc6fb761eda1bcc6125af985245de68
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Fri Apr 16 10:35:22 2021 -0400

    minor tweaks to viper.ml

commit 27a7189aede735f67e6e307c21c284e65772ed73
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Fri Apr 16 10:13:32 2021 -0400

    semantdriver fully compiles

commit c0c639951f3a20c0a9f1390856aa85171536e851
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Thu Apr 15 22:38:22 2021 -0400

    syntax error fixed, new foundational issue with scope found in DecAssign and Dec

commit 71971c1c4190fe688d04d346c1119eefa6fcee55
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Thu Apr 15 20:39:45 2021 -0400

    syntax bug

commit f7785228ea51c8b95a39bdbd94b7cf083edfae16
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Thu Apr 15 17:48:45 2021 -0400

    fixed uop

commit a4d11c486ebc927852f8a1761968b5aed17c7939
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Thu Apr 15 17:16:14 2021 -0400

    Added a bunch of stuff for semantic checking. Expr needs to be completed and then
functions need to be taken care of appropriately. Built in function declaration should
go into decs.ml

commit 9d1f57f5da76ce458f32e7cd7234a98dbea6d0f3
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Wed Apr 14 21:03:13 2021 -0400

    semant driver, updated ast and sast

commit 749a9871dd1bf62ac341aef741caeb53a9aefda6
Merge: 110312d a44466d
```

```
Author: Raghav Mecheri <37787004+raghavmecheri@users.noreply.github.com>
Date:   Tue Apr 13 13:12:22 2021 -0400

    Merge pull request #42 from raghavmecheri/fix/decompose_sugar

    Decomposed Syntactic Sugar

commit 1c3f2266cee5bc1ee2302eb263d66f68c4d9bb00
Author: Mustafa <maxusmusti@gmail.com>
Date:   Mon Apr 12 00:27:59 2021 -0400

    Membership checks

commit a44466d326221abba3dbc20ac7dae33dc6429bc3
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Sun Apr 11 03:49:19 2021 +0000

    Updated SAST with the types we're going to need

commit 780fe50c2e98fb225914576f091c9294eb6e3b81
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Sun Apr 11 03:48:18 2021 +0000

    Working de-sugaring for all the things we need

commit ee3b5f0ac9f5ae2863073780324982068 86fed4f
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Sat Apr 10 22:53:52 2021 +0000

    Sick this actually works that's wild

commit 6f9ca763a0ad5ccf7a84070f02aad51fd2f58a53
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Sat Apr 10 22:49:05 2021 +0000

    Initial try for ForIter + DecForIter

commit 930978cf0d06da0e4d2a8b5c192071ceaefc599a
Author: Mustafa <maxusmusti@gmail.com>
Date:   Fri Apr 9 01:52:26 2021 -0400

    Arraylists typed with access + other functions

commit 856300158dce8b6303b6f657eb2b88524c669b1e
Author: Mustafa <maxusmusti@gmail.com>
Date:   Sun Apr 4 17:31:31 2021 -0400

    Building Standard Library

commit 110312d7f8b9170df824c50ee75d48348dcb5cf2
Merge: faa5653 c0ed7a7
Author: Raghav Mecheri <37787004+raghavmecheri@users.noreply.github.com>
```

```
Date:   Tue Apr 6 21:47:04 2021 -0400

    Merge pull request #40 from raghavmecheri/feature/cleanup_ast

    AST Cleanup

commit c0ed7a7eaa21a8170b54c1ef7121f34abc8406cd
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Wed Apr 7 01:37:12 2021 +0000

    Ottomano fix

commit 0ef9b803058ca506557101332439ae2c18fd8693
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Tue Apr 6 17:55:25 2021 +0000

    Fixed issue with semant.ml

commit 547dd8524115467f1fe9556c8a55019d5ac016e5
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Mon Apr 5 04:19:51 2021 +0000

    Working, albeit buggy semant.ml

commit 7754c61704f7b44454aae215de894814d339b451
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Sun Apr 4 22:31:40 2021 -0400

    fixed bug for make

commit 681ba4957cb0a1572ef7fb00c41220d0d8c82dcf
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Sun Apr 4 17:29:53 2021 -0700

    pass ast to Semant in viper.ml

commit 70a2ac0404f0a98c4d09b9b3439fc0761775069c
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Sun Apr 4 17:30:32 2021 +0000

    Tried to get semant.ml to work

commit c96f298edfe9057251f40c65ec790af4d493cc2d
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Sat Apr 3 22:42:31 2021 +0000

    Removed de-const stuff + added it back to SAST

commit 215dd0e3675a985bcedea2cc6d67b35aab66a0af
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Sat Apr 3 21:47:09 2021 +0000
```

```
    Added code to pattern match + sanitize everything except foriters

commit 11d73204eb573264b9c8dac6c99857e2ddfefc48
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Sat Apr 3 10:19:54 2021 -0700

    func decs dont allow for dupe local vars

commit 6e9a67f3b1de9ed1a1dee79b536bd59fd086a2fa
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Fri Apr 2 22:24:58 2021 -0700

    collect func decls

commit 5dff49f4f1405963ae770c2f172e72967df3ab99
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Fri Apr 2 15:55:24 2021 -0700

    symbol_table -> scope_table

commit fb2f9c352beda582d0b36a4af06ba0220a73fcf7
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Thu Apr 1 17:07:02 2021 -0700

    complete variable symbol table aggregation

commit 86eb0cb4023df651d076ae345bdcf138242e55ae
Author: Raghav Mecheri <37787004+raghavmecheri@users.noreply.github.com>
Date:   Wed Mar 31 23:13:05 2021 -0400

    Update semant.ml

commit 1a0b8862d4ccee250bcfac937f20da780e71d3b5
Author: Raghav Mecheri <37787004+raghavmecheri@users.noreply.github.com>
Date:   Wed Mar 31 23:10:26 2021 -0400

    Update sast.ml

commit 2b12211605c6692a220b82378ab661f6a54bb931
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Thu Apr 1 03:09:15 2021 +0000

    Laid more groundwork in semant + fixed sast.ml to remove the types that I'm
getting rid of

commit 4d878b642d39bbb43a1605498e311e926d2540f3
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Thu Apr 1 01:12:00 2021 +0000

    Laid the groundwork for the de-syntactic sugaring process

commit 5af193247d549c0e18ad02c2f011176064a73b35
```

```
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Wed Mar 31 17:37:08 2021 -0700

    support more expr types

commit 182763d7d674f87deb7f21c81f27b0d49b1960ea
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Wed Mar 31 14:22:07 2021 -0700

    simplify get_stmt_decs

commit f8c729c5e6a4c7e2b708d00fc83429ada53595f6
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Wed Mar 31 14:16:49 2021 -0700

    rename for clarity

commit 041715228a33a35df8a2632cda61fe0924b0248b
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Wed Mar 31 14:08:03 2021 -0700

    check for nah declarations

commit ac85904772d086155f028f6be4a3c1468b3f8903
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Wed Mar 31 14:07:43 2021 -0700

    add support for foriter and while'

commit faa5653ee681a8998f1f9eba491db9f9518f8dd9
Author: Raghav Mecheri <raghav.mecheri@columbia.edu>
Date:   Tue Mar 30 21:17:48 2021 -0400

    fixed workflow file now (?)

commit 0e94dc94f696020dcf373dfc2411fc91079ff4e4
Author: Raghav Mecheri <raghav.mecheri@columbia.edu>
Date:   Tue Mar 30 21:05:46 2021 -0400

    Fixed workflow stuff

commit f6aef59bfcb3294f6393a4e29514864a4a8ac854
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Tue Mar 30 17:10:58 2021 -0700

    dup check for loops

commit 69ba3cea62244fe46e64ba0d58a66efc283a9eb1
Merge: 6e79484 bf5b9fb
Author: Raghav Mecheri <37787004+raghavmecheri@users.noreply.github.com>
Date:   Tue Mar 30 19:50:00 2021 -0400
```

```
    Merge pull request #38 from raghavmecheri/fix/ast_tests

    Fixed test suite

commit bf5b9fbb5d7836d21a78522a22d27713d45e8743
Merge: 2510872 6e79484
Author: Raghav Mecheri <37787004+raghavmecheri@users.noreply.github.com>
Date:   Tue Mar 30 19:49:42 2021 -0400

    Merge branch 'main' into fix/ast_tests

commit 2510872307154e4d363aee1cf821b75be6c512f7
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Tue Mar 30 23:45:28 2021 +0000

    Modified runtests.sh to fix LLVM test bug + added AST test call to Github Actions

commit a8437ec203a96212accea2fb76ee2f7cd7275d88
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Tue Mar 30 16:45:03 2021 -0700

    dup check for if statements

commit fdab10e2e5ca0fa3d732d6fc84422af661ab9f00
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Tue Mar 30 23:43:50 2021 +0000

    Re-added AST tests + added flag to run AST tests

commit 1afbdfc08a544eb32a372eb43c97a79b2bb1843e
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Tue Mar 30 09:21:15 2021 -0700

    fix scope bug

commit 003fe9e129d5022995dcdf273c5e676c5783ecdf
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Mon Mar 29 19:22:14 2021 -0700

    edit other test files to remove duplicate variables

commit 5d89102b442dcd3bdfc63387bd91ad389f3dc320
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Mon Mar 29 19:21:51 2021 -0700

    new test file for duplicate checking

commit c29f7ebc988422f18d749ff6529517432e8f9d5a
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Mon Mar 29 19:21:34 2021 -0700

    check for duplicate declarations in global stmts
```

```
commit 8b938ff627590ee22630bd341074cad83dcce289
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Mon Mar 29 18:22:12 2021 -0700

    finally got blocks to work

commit 6e794845bf6588be4af163f5597f01dd3bb74148
Author: treygilliland <jlg2266@columbia.edu>
Date:   Sun Mar 28 23:17:33 2021 -0400

    Added expression evaluation for Binops

commit 6104748c1070e0a33624596f6866f4e730232767
Author: treygilliland <jlg2266@columbia.edu>
Date:   Sun Mar 28 21:59:37 2021 -0400

    Added boolean literal evaluation and print value resolver function. Updated make
to be less annoying

commit 874866573aa25dedba7ad38519071871db288caa
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Sun Mar 28 17:59:07 2021 -0700

    fix compilation errors

commit c38255f60efae6b8d8635faa3ac26823c6fde5f4
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Sun Mar 28 17:54:31 2021 -0700

    include dec checking in viper.ml

commit a9b5adf3b002f50bc2a7c18673f4c9d074d8c0a2
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Sun Mar 28 17:53:08 2021 -0700

    create decs.ml

commit 9126c5b693364f2830fd183e975174ffa0fddd9a
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Sun Mar 28 17:32:50 2021 -0700

    leftover changes

commit c6ee980b6807d1c462f0d7bcc9f7dfebeeeec6df
Merge: d50cc0b 0de030d
Author: Raghav Mecheri <37787004+raghavmecheri@users.noreply.github.com>
Date:   Thu Mar 25 11:32:09 2021 -0400

    Merge pull request #34 from raghavmecheri/fix/fileorg

    Restructured files to remove src from the top level
```

```
commit 0de030d04b597c5ec5a0e398fffe728622da80be
Author: Raghav Mecheri <37787004+raghavmecheri@users.noreply.github.com>
Date:   Thu Mar 25 11:17:48 2021 -0400

    Update push.yaml

commit e8cb2f13783a1f08d01d9c55d3684f9fb326d094
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Thu Mar 25 14:19:44 2021 +0000

    Restructured files to remove src from the top level

commit d50cc0bde77d03b6e94a1694fbc370b38e206907
Author: Mustafa <maxusmusti@gmail.com>
Date:   Wed Mar 24 22:53:51 2021 -0400

    removed unused case

commit b82f21b3b60427ce2c7fe5fb69b6bc10656fb090
Author: treygilliland <jlg2266@columbia.edu>
Date:   Wed Mar 24 22:36:34 2021 -0400

    Cleaned up comments in codegen.ml, ready for submission

commit 59ef31e4798787a32773e35b95dbe6b8d255f016
Author: treygilliland <jlg2266@columbia.edu>
Date:   Wed Mar 24 22:29:26 2021 -0400

    Chaned expression evaluation types to match Viper LRM

commit 8dbcbb87af2c150e57a11ee95d8c073d9939e5e8
Author: treygilliland <jlg2266@columbia.edu>
Date:   Wed Mar 24 22:21:27 2021 -0400

    Codegen clean and added tests for printing floats

commit 4bf8d3d6c67d8da7c25a5d3071fbfe604a66777e
Author: treygilliland <jlg2266@columbia.edu>
Date:   Wed Mar 24 20:52:33 2021 -0400

    Moved expression evaluation to seperate function

commit c1a4bd676f926dc6f03590b20cf5baa07ee69d53
Author: treygilliland <jlg2266@columbia.edu>
Date:   Tue Mar 23 21:58:49 2021 -0400

    Updated print format chars

commit 89c396d9670f74a16141681559b2e41987a1186a
Author: treygilliland <jlg2266@columbia.edu>
Date:   Tue Mar 23 21:55:19 2021 -0400
```

```
    ratghav literally ruins everything

commit 9470ee091e3399d2072ee9b74a1dec9b3fb376c6
Merge: 3a4c6b2 243f681
Author: Raghav Mecheri <37787004+raghavmecheri@users.noreply.github.com>
Date:   Tue Mar 23 21:25:48 2021 -0400

    Merge pull request #32 from raghavmecheri/feature/codegen

    Feature/codegen

commit 243f681bc391565bf24d3760d625ec68df62f29a
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Wed Mar 24 01:14:32 2021 +0000

    Changed a.out --> a.ll

commit c3376d3d0d5c12c9398351c6a40bc24961b96a52
Author: treygilliland <jlg2266@columbia.edu>
Date:   Tue Mar 23 21:02:36 2021 -0400

    Simple comments

commit b3c447b56a9004123149782b0d4855a7ef9e81bf
Author: treygilliland <jlg2266@columbia.edu>
Date:   Tue Mar 23 17:50:27 2021 -0400

    Updated runtests.sh to run on bash, moved simple examples to test

commit 6ed9808e366b1788f3a5af6c3eb28b0372f8cd56
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Tue Mar 23 21:22:49 2021 +0000

    Added .out files so that tests don't fail

commit f1fd11ec2c718f9fb0e8572094f35bd373c0dc1f
Author: treygilliland <jlg2266@columbia.edu>
Date:   Tue Mar 23 16:50:21 2021 -0400

    Moved working tests to test

commit e17b0a0991518d1627f169d06c40aad8fdaf671f
Author: Raghav Mecheri <raghav.mecheri@columbia.edu>
Date:   Tue Mar 23 16:39:30 2021 -0400

    Modified runtests.sh to remove a.out

commit 1b1093e793088bf0c2d2bf564d52312d10f0023f
Author: Raghav Mecheri <raghav.mecheri@columbia.edu>
Date:   Tue Mar 23 16:39:03 2021 -0400
```

```
    Removed a.otu

commit 6fcd403a23e80305ab1e84a46e53cd9ba360f37e
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Tue Mar 23 20:34:59 2021 +0000

    Moved all the test cases to samples (for now) + implemented integration tests with
.out files

commit 18990543d0e488fbf444a2d68fc0f8541264536a
Author: treygilliland <jlg2266@columbia.edu>
Date:   Tue Mar 23 15:39:41 2021 -0400

    MILESTONE: can use multiple print statments to print chars, ints, or strings

commit 477bcf47fce79bef5cb8692b09a5a8a20c32bed2
Author: treygilliland <jlg2266@columbia.edu>
Date:   Tue Mar 23 12:59:04 2021 -0400

    MILESTONE: allow for multiple print statements on top level

commit 37201fe18cac90a0884918e180a73ae7e3d2127a
Author: treygilliland <jlg2266@columbia.edu>
Date:   Tue Mar 23 12:52:22 2021 -0400

    MILESTONE: Printing a number works

commit bbaabde697288121ffdd2f8135cfbc4b8a935a12
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Mon Mar 22 22:08:15 2021 -0700

    continue function semantic checking

commit 6458a71629c8be1d2f74c7e3330fcf2f7e540314
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Mon Mar 22 21:04:21 2021 -0700

    implement find_func

commit 9d5370466bc7c491ff6114e1ad0282a9361831ca
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Mon Mar 22 19:42:27 2021 -0700

    reorganize let statements

commit 677fb9657ec04751dea8a78617edd9420eafb48f
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Mon Mar 22 19:31:43 2021 -0700

    improve overloaded func checking to use unique keys

commit c6d802d5dca59cb6e9fd58d227868a0f005cd0c5
```

```
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Mon Mar 22 18:35:53 2021 -0700

    globals -> statements

commit 7a45797964acd498aa9fd58afaee327d6cf87c29
Merge: 62e8089 a0022bf
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Mon Mar 22 17:11:19 2021 -0700

    merge

commit 62e8089d8a27ea50a652adb0f847903e7649f75e
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Mon Mar 22 17:09:47 2021 -0700

    add new test for overloaded functions

commit 6bd2c0dbed23c842a53b6fa038fc892e03661152
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Mon Mar 22 17:08:34 2021 -0700

    include semant checking to allow testing

commit 277534d5472eb5fc650d512cbbbc0b7349310b75
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Mon Mar 22 17:08:01 2021 -0700

    allow for overloaded functions in function checking

commit 5137a9500a90b6fc5d492e0c0f25eb3bc7e99905
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Mon Mar 22 13:56:16 2021 -0700

    fix pattern matching

commit 57ddc6ff20bb2a043599d02883fa6c4951653ef7
Author: treygilliland <jlg2266@columbia.edu>
Date:   Mon Mar 22 16:04:43 2021 -0400

    Added some dumb test files, updated viper.ml to allow for -l flag, added a ton to
codegen

commit 08aa79c936b263dbda1618a890f8ac6e1415bd0c
Author: treygilliland <jlg2266@columbia.edu>
Date:   Mon Mar 22 03:50:43 2021 -0400

    Broke ground on codegen.ml, simplified helloworld.vp bc Ratghav is too try hard

commit 4d8a5b35e61e95ca87f2cb23d3e7f39ccf72bd11
Author: Raghav Mecheri <raghav.mecheri@columbia.edu>
Date:   Sun Mar 21 22:02:49 2021 -0400
```

```
    Added test/samples/helloworld.vp

commit e4e9c11ea7154444d1b9a0e8b2c00e404ec0efdf
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Mon Mar 22 01:55:10 2021 +0000

    Commented out semant stuff so that it just returns a random ast

commit a0022bf3bec9dd602749fefc4ad9ab6c6f627a02
Merge: ce83011 3a4c6b2
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Mon Mar 22 01:34:48 2021 +0000

    Merge branch 'main' of github.com:raghavmecheri/viper into feature/semantics

commit ce830110167072abb732b44d3bdd40b1c414787b
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Mon Mar 22 01:34:44 2021 +0000

    Attempting to build sast.ml

commit b3c81ace40ec299de5897fa0adf8af6589d8970e
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Sat Mar 20 22:34:47 2021 -0700

    simplify pattern matching

commit 3a4c6b223992f186fb0f2543256afa8e057d60a4
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Sat Mar 20 18:43:42 2021 -0700

    fix errors

commit 208e6114ed6155502051feacd410ceb2af09061a
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Sat Mar 20 18:42:55 2021 -0700

    implement function dup/overwrite checking

commit 07840d078e1de70ab704b317b871b3cb2d4db73d
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Sat Mar 20 17:34:10 2021 -0700

    check for nah defs and duplicate vars

commit dee53d3fb4ad994aeb76ad093921b93309e75d8d
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Sat Mar 20 17:00:19 2021 -0700

    void -> nah
```

```
commit 6dc600b59927410807f832726047b3d176e8cd39
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Thu Mar 18 17:03:39 2021 -0700

    fix indent

commit e7d8ad291ada48b3d4ca3a71dd3a3db3ad6e6aa4
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Thu Mar 18 17:00:48 2021 -0700

    update pretty printing in sast.ml

commit 81cccc872a0fd26f92bc6210fcfb1dc30fbbaa2f
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Thu Mar 18 15:31:50 2021 -0700

    update sast.ml with viper sexprs and sstmts

commit 5d05d32140ca94a55409d6b101053cceb0050b5d
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Thu Mar 18 15:13:24 2021 -0700

    Tuple -> Group

commit 39575cb0d4ecfdb45f5da9466102a1727d90be0d
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Thu Mar 18 15:10:15 2021 -0700

    change microc -> viper

commit d5e375777d3c497e9276d20416d892381364076b
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Mon Mar 15 22:23:07 2021 -0700

    Add short script to quickly build MicroC container.

    I got tired of copying and pasting the command.

commit 9d62a4c66814f8fa82114a7c50f4628d64f30732
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Sun Mar 14 17:45:08 2021 -0400

    semantic checking

commit 89d7b96dfb834d3b405d38d287660b401ee8becb
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Thu Feb 25 13:26:14 2021 -0800

    Rebased to fix issues with CI

    Fix minor issues
```

```
    Update Makefile

    Update push.yaml

    Update Makefile

    Update Makefile

    Hopefully fixed CI

    Update push.yaml

commit 92a38f634d6cb7822cc50d7dd731bc698ba26d52
Merge: 118a8d7 790a35b
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Wed Feb 24 23:22:10 2021 -0800

    merge and add header

commit 118a8d7a68946d2687a87cbb6bd37a3250ceb6bf
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Wed Feb 24 23:19:24 2021 -0800

    add code example

commit 790a35bb3e5c60e0796ab3b5938b5bcd152e5ff5
Author: Raghav Mecheri <raghav.mecheri@columbia.edu>
Date:   Wed Feb 24 23:51:25 2021 -0500

    Added spaces before table (export issue)

commit b19c0da129ee8bc8897c1c6680d524bd7eeef453
Merge: bfa443a a86efa9
Author: Raghav Mecheri <raghav.mecheri@columbia.edu>
Date:   Wed Feb 24 23:26:08 2021 -0500

    Merge branch 'main' of github.com:raghavmecheri/succotash into main

commit bfa443af32908b6186597c062f4078ea6ea8303e
Author: Raghav Mecheri <raghav.mecheri@columbia.edu>
Date:   Wed Feb 24 23:26:02 2021 -0500

    Removed duplicate #10 entry on ToC

commit a86efa9f6523142a9ed27f6d08f7f969610933d4
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Wed Feb 24 20:20:45 2021 -0800

    Format headers

commit c230cb488878c76f4fa625774018b43590371409
Author: Raghav Mecheri <37787004+raghavmecheri@users.noreply.github.com>
```

```
Date:   Wed Feb 24 23:16:42 2021 -0500

    Update LRM.md

commit fe0fdde802cc10f39428cb543ed2d0fef8cd872f
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Wed Feb 24 20:11:43 2021 -0800

    Fix link issues

commit 2eba2e5d453e5a6ae18c6779512e07c4fbc04cdb
Author: Raghav Mecheri <37787004+raghavmecheri@users.noreply.github.com>
Date:   Wed Feb 24 23:07:38 2021 -0500

    Update LRM.md

commit ab1154dc0c853528c77206443fa8b8a418a81e8b
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Wed Feb 24 20:05:31 2021 -0800

    Fix merge conflicts

commit a3ed27219a5030535b57d5e2562d3e5b8289fb26
Merge: a08d41a 896c7ef
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Wed Feb 24 20:02:21 2021 -0800

    Merge branch 'main' of https://github.com/raghavmecheri/viper into main

commit a08d41a94dafa347c932fe9b0dd5fcc869e133f6
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Wed Feb 24 19:58:53 2021 -0800

    update table of contents

commit 4fcee828665fa95e2f89a8e3e4c930676d70ede3
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Wed Feb 24 19:46:58 2021 -0800

    add len() to std lib

commit 6fb72c4356f04e18a2dc3f525b756a238383e057
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Wed Feb 24 19:28:39 2021 -0800

    add list API

commit 896c7ef23b73147495db24406b933791725ce345
Author: treygilliland <jlg2266@columbia.edu>
Date:   Wed Feb 24 22:13:57 2021 -0500

    Added contents for section 4
```

```
commit cfbff52bd490e4fe67a53ca88e00275eb885c30b
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Wed Feb 24 18:49:17 2021 -0800

    add type functions to stdlib

commit 6bdbefa15b840b4cf64162cfc8f14974db85c66d
Author: Raghav Mecheri <37787004+raghavmecheri@users.noreply.github.com>
Date:   Wed Feb 24 21:41:12 2021 -0500

    Update Makefile

commit 588f73406e159bc036df42e42f74d5cab458d1e2
Merge: 274c6b0 4cc900c
Author: Raghav Mecheri <raghav.mecheri@columbia.edu>
Date:   Wed Feb 24 21:36:31 2021 -0500

    Merge

commit 274c6b0516fdd1716602e65f72680b916a7496b9
Author: Raghav Mecheri <raghav.mecheri@columbia.edu>
Date:   Wed Feb 24 21:34:37 2021 -0500

    LRM tweaks + sample addition

commit 4cc900cce392b1028a547a606b47c41d73c30c37
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Wed Feb 24 21:28:01 2021 -0500

    Update LRM.md

commit 425171503f88cfb9580d719fbd8b0a09bf6fc28d
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Wed Feb 24 21:11:50 2021 -0500

    Update LRM.md

commit d5ba957e777e8f1e076663aa1cb8d6cf1c22a896
Author: Mustafa <maxusmusti@gmail.com>
Date:   Wed Feb 24 20:42:58 2021 -0500

    Fixed decl pretty print

commit f244c97c6a484a84f14835989b0fc46ebc7dd480
Author: Mustafa <maxusmusti@gmail.com>
Date:   Wed Feb 24 20:34:31 2021 -0500

    Fixed dict access

commit bd6a5cf2e26764a997973c089d14c6f007da90a7
Author: Tommy Gomez <tjk2132@columbia.edu>
```

```
Date:   Wed Feb 24 17:32:11 2021 -0800

    add string to primitive tables

commit cc309c68540e1260324690148l8cbf44e8520868
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Wed Feb 24 17:26:58 2021 -0800

    add math functions to std lib

commit 13e23d729d3249e289a0f723264ba624ba8676a0
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Thu Feb 25 00:59:59 2021 +0000

    Dictionary support with one r/r error

commit fdf7b2261de647d9613e3f26bd63ad2cff1071e9
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Wed Feb 24 16:42:23 2021 -0800

    redefine structure of std lib

commit 9467c246d890ed1428329b2d0c8346f61bb52e02
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Wed Feb 24 16:21:11 2021 -0800

    remove explicit size declaration from lists

commit ab00d9b25f16319c07ffba42dd9e96b8d92a4ed6
Merge: 0fb7664 d921788
Author: Raghav Mecheri <37787004+raghavmecheri@users.noreply.github.com>
Date:   Wed Feb 24 19:12:44 2021 -0500

    Merge pull request #24 from raghavmecheri/feature/callable_expr

    Functional expr.call() + tests

commit d921788fe4a335148a77f5c9857db3fdb2a7b4c4
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Thu Feb 25 00:11:17 2021 +0000

    Modified test cases

commit 0fb7664d1fa29baf6363858c7d525da3a6f5fe44
Merge: dcaac11 5655609
Author: Raghav Mecheri <raghav.mecheri@columbia.edu>
Date:   Wed Feb 24 17:18:42 2021 -0500

    Merge branch 'main' of github.com:raghavmecheri/succotash into main

commit dcaac117cd365db53c09ac351bb44bb789599fa7
Author: Raghav Mecheri <raghav.mecheri@columbia.edu>
```

```
Date:   Wed Feb 24 17:18:38 2021 -0500

    Fixed init

commit 56556090ccfd7e738682378caa59a078c7dbc221
Author: Raghav Mecheri <37787004+raghavmecheri@users.noreply.github.com>
Date:   Wed Feb 24 17:12:49 2021 -0500

    Update push.yaml

commit 4c34e6466690975acf0ca100e55516a73f447f41
Author: Raghav Mecheri <raghav.mecheri@columbia.edu>
Date:   Wed Feb 24 17:11:12 2021 -0500

    Updated gitignore + CI

commit c59bc19ec0f600646cb7efd801bcdd95b6ac889b
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Wed Feb 24 22:03:19 2021 +0000

    Functional expr.call() + tests

commit 4ef88d1e509017f561119bccd9da76fa2d9217b0
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Wed Feb 24 00:44:56 2021 -0800

    Fix broken links

commit b735dd8c6fb6c73ef7a1a36408659629df3a0802
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Wed Feb 24 00:34:31 2021 -0800

    fix header links to work on github

commit 24ff09de53e9d4163676013dbaf3736c2409b543
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Wed Feb 24 00:21:24 2021 -0800

    test emoji header

commit d3df496b878878304d62eda0dbc0ecaf32bcb534
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Wed Feb 24 00:20:07 2021 -0800

    test headers in github

commit c601a9a752a6a1308cc3d24c18534f58ce5d668a
Merge: d749806 dcb4ba5
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Wed Feb 24 00:14:14 2021 -0800

    merge
```

```
commit d7498066564c05b7ef417cf2152ad4dade1efc0b
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Wed Feb 24 00:07:30 2021 -0800

    add emoji

commit aac50f11e04e7341ec78336da88f1c47b18a503d
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Tue Feb 23 23:44:23 2021 -0800

    Add return to contents at end of each section

commit f8903a085b37c42b1db776a2daa54ff6721ecd0d
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Tue Feb 23 23:34:21 2021 -0800

    implement clickable table of contents

commit 2fcbd4d1b881d7088796401fe396c576dbd05609
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Tue Feb 23 22:12:31 2021 -0800

    && + || -> and + or

commit dcb4ba575be3e5708a2f36d79dc4a3cf547a8597
Author: treygilliland <jlg2266@columbia.edu>
Date:   Wed Feb 24 01:12:28 2021 -0500

    Completed Type System, began working on section 6 Standard Library

commit df8e2d3c4e9713389bb34e713226d31ab66ebc9b
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Tue Feb 23 22:10:31 2021 -0800

    progress building table of contents

commit 615e7834b093b0fde511918754e0bc314661e55b
Merge: 6eaa36a f82512e
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Tue Feb 23 21:30:33 2021 -0800

    merge

commit 6eaa36af89ecc8c8355019e73a57b3062e2ad956
Merge: 1bcfecd e41218b
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Tue Feb 23 21:28:58 2021 -0800

    continue editing contents

commit f82512e7f4dd525844a48f432033d70ca84fb0f7
```

```
Merge: 596862e b5f38aa
Author: treygilliland <jlg2266@columbia.edu>
Date:   Tue Feb 23 22:56:27 2021 -0500

    Here goes nothing

commit 596862e897ff92514455ba490a42aeabb01d501f
Author: treygilliland <jlg2266@columbia.edu>
Date:   Tue Feb 23 22:54:17 2021 -0500

    Updated Section 2

commit b5f38aad6628909cfacf456820070472c32475dd
Author: Mustafa <maxusmusti@gmail.com>
Date:   Tue Feb 23 22:28:27 2021 -0500

    Cleaned repo

commit 4fcae81830483fd23c34ef51b6ae668a9a37347c
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Tue Feb 23 22:25:31 2021 -0500

    Update LRM.md

commit 0759cc036cc6917ea0fef036ba7d27ed7dd98a8e
Author: Mustafa <maxusmusti@gmail.com>
Date:   Tue Feb 23 22:22:00 2021 -0500

    LETS GO reduce/reduce vanquished

commit 1bcfecd549c22a6fc54f349feaabf6dfb5af9d59
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Tue Feb 23 19:18:31 2021 -0800

    start table of contents

commit 0305e976107c0e03adb19f5996b4d1958b9adb0f
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Tue Feb 23 21:53:03 2021 -0500

    Update LRM.md

commit 56d1d17d69bcbc31fbba69561820a81c0be30077
Merge: 87f4e18 e41218b
Author: treygilliland <jlg2266@columbia.edu>
Date:   Tue Feb 23 21:50:49 2021 -0500

    Idk what happened

commit 87f4e18204a78823bb2074938648623a74e97a07
Author: treygilliland <jlg2266@columbia.edu>
Date:   Tue Feb 23 21:49:20 2021 -0500
```

```
    Ended code block at end

commit e41218b9df5b2844d87cb363e0c73f3b37f547ed
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Tue Feb 23 21:45:28 2021 -0500

    Update LRM.md

commit 9165718ca9573031522c5e5210e643c71fe2e010
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Tue Feb 23 21:44:24 2021 -0500

    Update LRM.md

commit edfbd3c6aa16bf203f37ae90da7479fc16ac21b1
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Tue Feb 23 17:56:52 2021 -0800

    add overview

commit ce07a4e67342e14e6c9355cef6324c67b92c60aa
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Tue Feb 23 16:53:21 2021 -0800

    test contents

commit b555bd246ea11decf55335d9e449566e99ac4f40
Merge: 82513f8 38952b1
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Tue Feb 23 16:49:04 2021 -0800

    Merge branch 'main' of https://github.com/raghavmecheri/viper into main

commit 82513f8d5d152ce405130e6034854992ed0722df
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Tue Feb 23 16:48:57 2021 -0800

    link test

commit 38952b18e77878beea43c7ca6f69c22de4f79508
Author: Raghav Mecheri <37787004+raghavmecheri@users.noreply.github.com>
Date:   Tue Feb 23 19:15:10 2021 -0500

    Delete old_testo.vp

commit 8a024ef51fa62f5df180d85aee30f4ebaa2faca7
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Wed Feb 24 00:08:58 2021 +0000

    Closed #7, #18, #19, #9, #2
```

```
commit 6e717da2822920aaed889ecd675b6b99a46790c6
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Tue Feb 23 23:27:24 2021 +0000

    Functional ability to pass arrow functions (non anon

commit d93dfe9621f094289f5290ab0a5ae34d13033a1b
Merge: fad9525 c5165ee
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Tue Feb 23 23:17:10 2021 +0000

    Merge branch 'main' of github.com:raghavmecheri/viper into feature/tuples

commit fad9525c05cf77715a96ad053152d069128ddbbc
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Tue Feb 23 23:16:47 2021 +0000

    Tuple integration + test cases for loop syntactic sugar

commit c5165ee455334efaa4c2976d31c629e39dead2c4
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Tue Feb 23 17:46:43 2021 -0500

    Update LRM.md

commit b543c8f246f8ca428b7d65051f6146f2ba6ee140
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Tue Feb 23 17:42:22 2021 -0500

    Update LRM.md

commit 418a6f5fd8330061efaa3c979581f58131b1b089
Author: treygilliland <jlg2266@columbia.edu>
Date:   Tue Feb 23 17:00:24 2021 -0500

    Updated sample code syntax errors and layed out standard library format

commit 8e6214d779a29872156dc863aef9fd552d2435b4
Merge: 39fa76b 3de9841
Author: treygilliland <jlg2266@columbia.edu>
Date:   Tue Feb 23 16:45:04 2021 -0500

    Merge branch 'main' of https://github.com/raghavmecheri/viper into main

commit 39fa76b086ded5d8b065052f305b0b4be8bf8043
Author: treygilliland <jlg2266@columbia.edu>
Date:   Tue Feb 23 16:44:45 2021 -0500

    Added example code

commit 27b4809d158e89904c3a58696c17edb7bba9fd41
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
```

```
Date:   Tue Feb 23 21:44:31 2021 +0000

    Working tuples, but with one SR conflict

commit 7261ef7724a1cfdc26b8ea61e13bd951c60b2a6e
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Tue Feb 23 12:00:39 2021 -0800

    move string to primitives

commit 3de9841edea471e2a18dc3c23e3d70228d362208
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Tue Feb 23 13:37:36 2021 -0500

    Update LRM.md

commit 05244d627468251a82c090ac05d5e63a055ff500
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Tue Feb 23 13:36:41 2021 -0500

    Update LRM.md

commit 236cd27329554dd368e9772c04ee1337d56d62c3
Merge: ba5baee 2f1d449
Author: treygilliland <jlg2266@columbia.edu>
Date:   Tue Feb 23 13:11:02 2021 -0500

    Combined Trey's first commit and LRM updates

commit 2f1d44994e928ea932b2c59b1b2d355b94fba737
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Tue Feb 23 00:32:41 2021 -0800

    add colored code

commit 817836cf33f3e9f9439e589f07bde0c857ffbc6c
Merge: afbafbb 25e28a9
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Tue Feb 23 00:26:53 2021 -0800

    Merge branch 'main' of https://github.com/raghavmecheri/viper into main

commit afbafbb279f176ae1e7465a1a9f15a5e42da4a18
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Tue Feb 23 00:26:46 2021 -0800

    Update data types in LRM

commit 25e28a986e89765943a7583c690f6167ed076d10
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Tue Feb 23 07:07:36 2021 +0000
```

```
    Fixed array

commit ba5baee95823bce43cb80632f7698e7d87f9f0e5
Author: treygilliland <jlg2266@columbia.edu>
Date:   Tue Feb 23 01:56:29 2021 -0500

    Trey commit #1

commit bb7c499529b5a38f072bd36f87f9a6bf38d38426
Merge: a241a28 ce62732
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Tue Feb 23 06:52:13 2021 +0000

    Merged upstream changes

commit a241a280fde4f8f275b183d2803586491e683215
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Tue Feb 23 06:50:13 2021 +0000

    Type matching alpha chadism

commit 6d76b226ba920b37d77d15968ad500c6aa556c05
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Tue Feb 23 06:04:20 2021 +0000

    Broken pattern matching

commit ce62732a9a2a904702e2bdc0fc610f0d3e700b93
Author: Mustafa <maxusmusti@gmail.com>
Date:   Tue Feb 23 00:59:04 2021 -0500

    Expanded possibilities for chars and strings

commit 565c8ddc764c5a5f0b3b09f0617f7ac4c8118ec6
Author: Mustafa <maxusmusti@gmail.com>
Date:   Tue Feb 23 00:52:52 2021 -0500

    First string test

commit fefef74584be6b7d51b61843d77adc3513249f94
Author: Mustafa <maxusmusti@gmail.com>
Date:   Tue Feb 23 00:52:31 2021 -0500

    Strings and String Literals

commit f3fdece37f140389e811bce41cc105581e4ee821
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Tue Feb 23 04:35:55 2021 +0000

    Basic ternary integration

commit 7675a58caa31a875f463fd486744026214a5ffe6
```

```
Merge: 4d646b2 fe00f18
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Tue Feb 23 04:07:51 2021 +0000

    Merged upstream changes

commit 4d646b2c4f5f749a44a00180f95e59b60f74053d
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Tue Feb 23 04:06:18 2021 +0000

    Super saiyan runtests.sh

commit fe00f184731ad5ea05b2f530cae440d298b7c3d9
Author: Mustafa <maxusmusti@gmail.com>
Date:   Mon Feb 22 23:00:08 2021 -0500

    Added skip,abort,panic

commit d060be5fa5627e5d387aff6f026331042fa4dac1
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Tue Feb 23 03:55:39 2021 +0000

    Modified test script to bail on first failure

commit 7214de7ab7f9482a2982c11346eeee9471f58e89
Merge: b3cf6be 331875b
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Tue Feb 23 03:50:05 2021 +0000

    Merge upstream

commit b3cf6be4efc35ec0102e4efa57df414652d8d1e0
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Tue Feb 23 03:47:42 2021 +0000

    Added floats cause I'm the GOAT

commit f26b91ee3bbc47276e2144dfdfe534edab846fd8
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Tue Feb 23 03:36:33 2021 +0000

    Implemented operator shorthands, closed #16

commit 331875b3f9c0e580cab8c49436eb0ea9bc131dc3
Author: Mustafa <maxusmusti@gmail.com>
Date:   Mon Feb 22 22:21:50 2021 -0500

    Test cases for in and has

commit 173129764671ac5d0c04445ae9bcad81d07eb311
Author: Mustafa <maxusmusti@gmail.com>
Date:   Mon Feb 22 22:14:32 2021 -0500
```

```
    HAS operator

commit 3b65823fad412c2059ae1d8368b5239d7d8a91b2
Author: Mustafa <maxusmusti@gmail.com>
Date:   Mon Feb 22 21:55:33 2021 -0500

    In op p1

commit 6fdf1b66383b995961289ba785557a77df916911
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Tue Feb 23 02:51:05 2021 +0000

    Added condition + for fix

commit df49c2039fa23c980ff606c699058d28df54ea70
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Tue Feb 23 02:37:34 2021 +0000

    Closed #11

commit f3f56f639926aa148d351770a44e0a38081c5ddd
Merge: 193a3e8 e90cb0d
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Tue Feb 23 02:23:22 2021 +0000

    Merge branch 'main' of github.com:raghavmecheri/viper into main

commit 193a3e889b5a5bbe07a4829eb2b966963e6ffa46
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Tue Feb 23 02:22:49 2021 +0000

    Basic functionality + no errors :)))

commit a0221b737f28a0d16a4018ffebfaa4bac77c620b
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Tue Feb 23 01:48:08 2021 +0000

    Not yucky grammar

commit 67c0ebc3041b2e4a6e32ef6796e85a29ba8a2a3d
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Tue Feb 23 00:29:06 2021 +0000

    Redid test suite

commit 844be1d01de393782994ca4287b24a23b23ad80a
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Mon Feb 22 19:29:15 2021 +0000

    Formalised tests a little bit more
```

```
commit e90cb0de0ca79681e2ba96055af3bdd2843882cb
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Mon Feb 22 12:37:37 2021 -0500

    Update LRM.md

commit 5fc4864ac6b9f6def1ab61f1e6ecba0af35add69
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Mon Feb 22 12:34:21 2021 -0500

    Update LRM.md

commit 0bfe9932e4af67f18631d6f0706c146141cbe239
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Mon Feb 22 15:42:52 2021 +0000

    Broken, but somewhat valid attempt

commit 42057fece458597d75ea011213a85f57eb41a5ce
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Mon Feb 22 15:35:04 2021 +0000

    Declarations + decassigns both work

commit edbd6669981645da140797a69f5fb0bb1a39376c
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Mon Feb 22 14:30:37 2021 +0000

    Array indexed assignment to expression + closed #4

commit 0042d9957963c6725f75ef8e6ef75454f04f6abf
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Mon Feb 22 14:27:45 2021 +0000

    Array indexed access

commit be3aced4d0b46db7c029f19ec57ccd8adf4010c6
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Mon Feb 22 14:22:01 2021 +0000

    Musti was trolling

commit ef5fef858593147341a623809313b1a5f3f59b37
Merge: 5a8fbe3 6b40b7f
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Sun Feb 21 23:23:09 2021 -0800

    Merge branch 'main' of https://github.com/raghavmecheri/viper into main

commit 5a8fbe399a5a67f3d4d0b0013370dfb813701716
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Sun Feb 21 23:23:03 2021 -0800
```

```
    start adding higher-order data types

commit 6b40b7f778cd31b4ac8d3130ea6a9dfef39d401c
Merge: 65a0030 7e536cf
Author: Mustafa <maxusmusti@gmail.com>
Date:   Mon Feb 22 01:00:21 2021 -0500

    Merge branch 'main' of https://github.com/raghavmecheri/viper into main

commit 65a00308290c3d1e4b5b1ed164b1877e6cd783de
Author: Mustafa <maxusmusti@gmail.com>
Date:   Mon Feb 22 00:59:52 2021 -0500

    Lists part 1

commit 7e536cf545dcb303ddd0d9eb68db420f42d4ce3f
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Mon Feb 22 00:04:24 2021 -0500

    Update LRM.md

commit 8777652d9f2db8fc85b5df411d54584724bc808a
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Sun Feb 21 23:24:27 2021 -0500

    Update LRM.md

commit ca15a5d1ca9d0333b028319d494f71989ff29107
Merge: feab3ab 7f19210
Author: Raghav Mecheri <37787004+raghavmecheri@users.noreply.github.com>
Date:   Sun Feb 21 23:23:33 2021 -0500

    Merge pull request #8 from raghavmecheri/fix/brokenvdecls

    Working weird shape stuff

commit 7f192107af4add8a2ab1f5c709a2aac4c3ea2eba
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Mon Feb 22 04:22:52 2021 +0000

    Working weird shape stuff

commit 99da8a685d5da8a1186bc524658310da159083f4
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Sun Feb 21 19:26:56 2021 -0800

    update int, float, bool

commit feab3abf26247d453cfd5aece2609330e168329f
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Sun Feb 21 21:36:08 2021 -0500
```

```
    Update LRM.md

commit b1fcc62a1b9e560f1e2bd2c4e084262aa6432154
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Sun Feb 21 20:52:06 2021 -0500

    Update LRM.md

commit fba0a770706a5eb91e82b921a5bf4b386fd8521f
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Sun Feb 21 20:46:51 2021 -0500

    Update LRM.md

commit 010696560dab8b9fb2c22dce75fa1a163b3edc33
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Sun Feb 21 20:45:11 2021 -0500

    Update LRM.md

commit 4f077353f39451a803ab1d8a92861405b224a1c2
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Sun Feb 21 20:20:57 2021 -0500

    Update LRM.md

commit 4874739065314e0232fc0b5e354f7cc38afb60d9
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Sun Feb 21 17:19:04 2021 -0800

    Update data types and chars

commit 750d9548af980d96b62c4fb20f3ddd2b85f71800
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Sun Feb 21 19:50:47 2021 -0500

    Update LRM.md

commit 3e5e3c7e958d2f04e26817fc04de0cf8b879a07f
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Sun Feb 21 19:06:12 2021 -0500

    Update LRM.md

commit 3a1b05e287e5eaf1798b2e9eac94cc490dddeb0d
Merge: 76207ff 528b583
Author: Raghav Mecheri <37787004+raghavmecheri@users.noreply.github.com>
Date:   Thu Feb 18 16:05:31 2021 -0500

    Merge pull request #1 from raghavmecheri/feature/arrow
```

```
    Feature/arrow

commit 528b58358ebb45736908ce0b1c6e705011411ef2
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Thu Feb 18 20:06:16 2021 +0000

    Added example of second type of arrow function

commit 0523a908238cdd83200aae85178ce1749b874fbd
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Thu Feb 18 20:05:27 2021 +0000

    Got arrow functions to translate to regular functions :)

commit 0e3f3a3764f841af4f2bb3aa638f81321c0486a2
Author: Raghav Mecheri (Linux) <raghav@mecheri.in>
Date:   Thu Feb 18 20:05:13 2021 +0000

    Added gitignore

commit 76207ff128c40ecc17f1b74a464b1cda303c539d
Author: Mustafa <maxusmusti@gmail.com>
Date:   Wed Feb 17 23:39:00 2021 -0500

    Example

commit 0ab0bbaacc03d5080536013be52e33121d069c74
Author: Mustafa <maxusmusti@gmail.com>
Date:   Wed Feb 17 23:38:00 2021 -0500

    Additional files for Makefile functionality

commit b0c05fd1e940a5aab31854f0c93e34746d81746b
Author: Mustafa <maxusmusti@gmail.com>
Date:   Wed Feb 17 23:30:28 2021 -0500

    First run scanner, parser, and ast changes

commit 4b306c4441a3fbd7152a08bcf9f65b95e495aceb
Author: Mustafa <maxusmusti@gmail.com>
Date:   Wed Feb 17 21:10:25 2021 -0500

    Adding additional files

commit 76612774f11806196c938e55a621185a2065bf37
Author: Raghav Mecheri <37787004+raghavmecheri@users.noreply.github.com>
Date:   Tue Feb 16 21:53:27 2021 -0500

    Create LRM.md

commit 8dcf781cae965f914cf144a8663bbcd3110c52dd
Merge: f73fd69 0eba599
```

```
Author: Raghav Mecheri <raghav.mecheri@columbia.edu>
Date:   Tue Feb 16 02:30:32 2021 -0500

    Merge branch 'main' of github.com:raghavmecheri/succotash into main

commit f73fd692ac6395d79efa8252607f0c43edfcf269
Author: Raghav Mecheri <raghav.mecheri@columbia.edu>
Date:   Tue Feb 16 02:30:25 2021 -0500

    Added a couple of types to the scanner

commit 0eba599255a3d6ee436d71f18667324edf91fcbe
Author: Mustafa <maxusmusti@gmail.com>
Date:   Tue Feb 16 01:52:35 2021 -0500

    Fix README

commit 1ce18f321b552cb32b5b300a988982e5be816530
Author: Raghav Mecheri <raghav.mecheri@columbia.edu>
Date:   Tue Feb 16 01:49:51 2021 -0500

    minor changes

commit e62cdf63254f7722f0fb36fd086f989e7b4260d3
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Mon Feb 15 16:11:23 2021 -0500

    added scanner, parser and ast from microc

commit 1be7c0cadcdfea3a60ef56b8705b077fb7608087
Author: Raghav Mecheri <raghav.mecheri@columbia.edu>
Date:   Wed Feb 3 23:05:31 2021 -0500

    Fixed minor issues with proposal

commit 45daaf6c8b0c5a66f3fa9727d9e2c8d06763b116
Author: Raghav Mecheri <37787004+raghavmecheri@users.noreply.github.com>
Date:   Wed Feb 3 22:35:26 2021 -0500

    Update Proposal.md

commit 4997a6492de9afcc20bdb5f3d9d4918c83e1f3bc
Author: Raghav Mecheri <37787004+raghavmecheri@users.noreply.github.com>
Date:   Wed Feb 3 22:28:49 2021 -0500

    Update Proposal.md

commit 46a1a8d14a3194142cddffdfb01bf52709b6f192
Author: Raghav Mecheri <37787004+raghavmecheri@users.noreply.github.com>
Date:   Wed Feb 3 22:26:21 2021 -0500

    Update Proposal.md
```

```
commit 7821f7b96f085a4a427635c5ca19d1e1e58b0f17
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Wed Feb 3 17:14:35 2021 -0800

    Update Proposal.md

commit a628876b4d8d244bec686864e20a6f4ec023d5b8
Author: Raghav Mecheri <37787004+raghavmecheri@users.noreply.github.com>
Date:   Wed Feb 3 20:04:27 2021 -0500

    Update Proposal.md

commit a1a31369b3e4acac4fc3278c0c029f2b68e3fa84
Author: Raghav Mecheri <37787004+raghavmecheri@users.noreply.github.com>
Date:   Wed Feb 3 19:58:25 2021 -0500

    Update Proposal.md

commit 35e5b4a9f00992a24e78c368892dfaac4c851797
Author: Trey Gilliland <treygilliland3@gmail.com>
Date:   Wed Feb 3 18:45:28 2021 -0500

    Fixed semicolon

commit d63c8f86c39fb20fd6d8450a6a8b4eb7c3ff6928
Author: Mustafa Eyceoz <maxusmusti@gmail.com>
Date:   Wed Feb 3 18:39:43 2021 -0500

    Update Proposal.md

commit 3ee76505bdc89cae7b1871a2440c0f802477c312
Author: Trey Gilliland <treygilliland3@gmail.com>
Date:   Wed Feb 3 18:35:45 2021 -0500

    Update Proposal.md

commit 33fb83e33229e73b06a5ee9df9bb0d4599aad94c
Author: Mustafa Eyceoz <maxusmusti@gmail.com>
Date:   Wed Feb 3 16:04:45 2021 -0500

    Update Proposal.md

commit 1c18591c7596323c8d319a4acb19fa8a52b1da25
Author: Mustafa Eyceoz <maxusmusti@gmail.com>
Date:   Wed Feb 3 16:03:18 2021 -0500

    Update Proposal.md

commit 27c77205617dd7ad9d358dab5642af81c24c0cf5
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Wed Feb 3 15:53:24 2021 -0500
```

```
    Update Proposal.md

commit b2321fdeebff8b32f197c6157fd88c0acab62b43
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Wed Feb 3 15:51:46 2021 -0500

    Update Proposal.md

commit c7ff5dee18ea57f931ff295de1c0cd6a8640c371
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Wed Feb 3 15:46:46 2021 -0500

    Update Proposal.md

commit 91a6a1d9fcf6a1956f27cde4eabf5dc38843716d
Author: Raghav Mecheri <raghav.mecheri@columbia.edu>
Date:   Wed Feb 3 15:32:49 2021 -0500

    Fixed issues

commit 5a8949d091bfbd5e3631c9100d5140d6b83e29ef
Author: Mustafa Eyceoz <maxusmusti@gmail.com>
Date:   Wed Feb 3 15:21:28 2021 -0500

    Update Proposal.md

commit b966034090aa3b3cbaf8719697169144a9c21288
Author: Mustafa Eyceoz <maxusmusti@gmail.com>
Date:   Wed Feb 3 15:16:10 2021 -0500

    Update Proposal.md

commit 762af4033bf8df7a10763bcd69c0266adf1f5e92
Author: Mustafa Eyceoz <maxusmusti@gmail.com>
Date:   Wed Feb 3 15:11:37 2021 -0500

    Update Proposal.md

commit 97105b515e33dc995f6fd08832d5bae3bd8ea4bf
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Wed Feb 3 14:43:03 2021 -0500

    Update Proposal.md

commit cf60163ecc4693fdae4fc840536e442e1a1c32c6
Author: Dreams-Happen <ottomanomatthew@gmail.com>
Date:   Wed Feb 3 14:42:19 2021 -0500

    Update Proposal.md

commit bd4970131e3814fe8a1d193bf5e531d55b10a44b
```

```
Author: Mustafa Eyceoz <maxusmusti@gmail.com>
Date:   Wed Feb 3 14:12:38 2021 -0500

    Update Proposal.md

commit 3887c2c9cc33a2a5d0d663e50ff5b831568998ad
Author: Raghav Mecheri <37787004+raghavmecheri@users.noreply.github.com>
Date:   Wed Feb 3 14:09:33 2021 -0500

    Update Proposal.md

commit 7821d9f75b632e5fd016a14120992d22a1d1ad7f
Author: Raghav Mecheri <37787004+raghavmecheri@users.noreply.github.com>
Date:   Wed Feb 3 14:09:22 2021 -0500

    Update Proposal.md

commit 71e0746b5e2b5f2cce2cd545e973a46f60c97b94
Author: Raghav Mecheri <37787004+raghavmecheri@users.noreply.github.com>
Date:   Wed Feb 3 13:39:15 2021 -0500

    Update Proposal.md

commit dd127950e08eb4d4403bb0c090e4816b7eb39332
Author: Raghav Mecheri <37787004+raghavmecheri@users.noreply.github.com>
Date:   Wed Feb 3 13:30:52 2021 -0500

    Update Proposal.md

    Added section on functions

commit 0c44c2f83196014eeba8f6a0fa62715f2229c9e2
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Wed Feb 3 00:16:42 2021 -0800

    Add syntactic sugar to proposal🔍

commit b60116b863311e4373762da834dfe3bd350d9fdf
Author: Tommy Gomez <tjk2132@columbia.edu>
Date:   Tue Feb 2 22:19:20 2021 -0800

    Update README.md

commit 4213b2a3265365213d2950b4ed041fa67b49a4a5
Author: Raghav Mecheri <37787004+raghavmecheri@users.noreply.github.com>
Date:   Wed Feb 3 01:10:52 2021 -0500

    Update Proposal.md

commit ba5abb8c4dbb63f248a3089831a1b2e2ce1b45d2
Author: Raghav Mecheri <37787004+raghavmecheri@users.noreply.github.com>
Date:   Wed Feb 3 01:10:20 2021 -0500
```

```
    Update Proposal.md

commit e20a1ec24e1f404cf77b6f77934572570a1acf16
Author: Raghav Mecheri <37787004+raghavmecheri@users.noreply.github.com>
Date:   Wed Feb 3 01:08:37 2021 -0500

    Update Proposal.md

commit f465f53a1d66a56f02617639ed9de797950a41ab
Author: Raghav Mecheri <37787004+raghavmecheri@users.noreply.github.com>
Date:   Wed Feb 3 01:03:20 2021 -0500

    Update Proposal.md

commit 01202f796f339e885ffe6a368d11b983f9fa1826
Author: Raghav Mecheri <37787004+raghavmecheri@users.noreply.github.com>
Date:   Wed Feb 3 01:02:37 2021 -0500

    Delete REAMDE.md

commit a4e8105894b5765fb2bc93e7d76e04de76cc1877
Author: Raghav Mecheri <37787004+raghavmecheri@users.noreply.github.com>
Date:   Wed Feb 3 01:02:30 2021 -0500

    Create README.md

commit e55fc01f0a5d0b7d6b0b11ee5e15e7a7aee6f335
Author: Raghav Mecheri <37787004+raghavmecheri@users.noreply.github.com>
Date:   Wed Feb 3 01:02:01 2021 -0500

    Update Proposal.md

commit ed294641626dbae744500c786e35fcace1a4f1ab
Author: Raghav Mecheri <37787004+raghavmecheri@users.noreply.github.com>
Date:   Wed Feb 3 01:01:26 2021 -0500

    Update Proposal.md

commit f4ecd16fe0a6e827439057103ea2ffa45e38f7db
Author: Raghav Mecheri <37787004+raghavmecheri@users.noreply.github.com>
Date:   Wed Feb 3 01:00:11 2021 -0500

    Update Proposal.md

commit 92820d2c85f3ea284e25826327410e8cd02f5779
Author: Raghav Mecheri <37787004+raghavmecheri@users.noreply.github.com>
Date:   Wed Feb 3 00:59:29 2021 -0500

    Update Proposal.md

commit a4ef4123be3a12142f9dd851b58f87cdaf4b62ea
```

```
Author: Raghav Mecheri <37787004+raghavmecheri@users.noreply.github.com>
Date:   Wed Feb 3 00:56:06 2021 -0500

    Update Proposal.md

commit e82741b1b96314aacbbdfe0e6b8893ab46a836d6
Author: Raghav Mecheri <37787004+raghavmecheri@users.noreply.github.com>
Date:   Wed Feb 3 00:55:47 2021 -0500

    Update Proposal.md

commit d28723d7af44636618f574263e441dd70caf0e8e
Author: Raghav Mecheri <37787004+raghavmecheri@users.noreply.github.com>
Date:   Wed Feb 3 00:55:17 2021 -0500

    Update Proposal.md

commit ff696f231b94349b3f4cc37f29aff9c1cad62b9e
Author: Raghav Mecheri <37787004+raghavmecheri@users.noreply.github.com>
Date:   Wed Feb 3 00:54:35 2021 -0500

    Update Proposal.md

commit 5206b88bc79cf8e1dc6b2dcf2772dffbeebf827e
Author: Raghav Mecheri <37787004+raghavmecheri@users.noreply.github.com>
Date:   Wed Feb 3 00:52:17 2021 -0500

    Update Proposal.md

commit 3357d3818736c681f38a10979a6f0bb4ff350ec2
Author: Raghav Mecheri <37787004+raghavmecheri@users.noreply.github.com>
Date:   Wed Feb 3 00:50:40 2021 -0500

    Update Proposal.md

commit 9f151612d548c58ecb81292b361cfd5a4af8ccb1
Author: Raghav Mecheri <37787004+raghavmecheri@users.noreply.github.com>
Date:   Wed Feb 3 00:49:34 2021 -0500

    Update Proposal.md

commit ec3afce8f07e62b1c4db44ffc214543125277465
Author: Raghav Mecheri <37787004+raghavmecheri@users.noreply.github.com>
Date:   Wed Feb 3 00:42:56 2021 -0500

    Update Proposal.md

commit d57e7b82be31fda12f3ed6e286363ff5cc79bff9
Author: Raghav Mecheri <37787004+raghavmecheri@users.noreply.github.com>
Date:   Wed Feb 3 00:17:54 2021 -0500

    Create Proposal.md
```

```
commit 3f5c8e1c2555594630630797579 98d7e48681267
Author: Raghav Mecheri <raghav.mecheri@columbia.edu>
Date:   Tue Feb 2 23:54:03 2021 -0500

    Added README
```

16.2 Source Code

```
----------
File: demo/factorial.vp
int func factorial(int target) {
    if(target == 1) {
        print(target);
        return target;
    }

    int res = target * factorial(target-1);
    print(res);
    return res;
}

print("Factorial of 10:");
factorial(10);

----------
----------
File: demo/calc.vp
int func calc(int a, int b, char op) {
    int res = ??
        op == '+' : (a + b)
      | op == '-' : (a - b)
      | op == '*' : (a * b)
      | op == '/' : (a / b)
    ?? a ;

    return res;
}

int result = 0;

print("10 + 20 =");
result = calc(10, 20, '+');
print(result);

print("10 - 20 =");
result = calc(10, 20, '-');
print(result);

print("10 * 20 =");
result = calc(10, 20, '*');
```

```
print(result);

print("10 / 20 =");
result = calc(10, 20, '/');
print(result);

print("10 ^ 20 = (unknown operator returns first value)");
result = calc(10, 20, '^');
print(result);

----------
----------
File: demo/bubble.vp
int[] func bubble(int[] arr) {
    for(int i = 0; i < len(arr); i++) {
        int limit = len(arr) - i - 1;
        for(int j = 0; j < limit; j++) {
            if(arr[j] > arr[j+1]) {
                int temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
                print("Step:");
                print(arr);
            }
        }
    }

    return arr;
}


int[] target = [4, 5, 10, 1, 5, 3];
print("Initial:");
print(target);

int[] sort = bubble(target);
print("Final:");
print(sort);
----------
----------
File: demo/binse.vp
int func binary_search(int []arr, int x) {
    int low = 0;
    int high = len(arr);
    int mid = 0;

    while(low <= high) {
        mid = (high + low)/2;
        if(arr[mid] < x) {
            low = mid + 1;
        } else if(arr[mid] > x) {
            high = mid - 1;
```

```
        }
        else {
            return mid;
        }
    }
    return -1;
}

nah func print_result(int loc) {
    if (loc != -1) {
        print("Found! Location:");
        print(loc);
    } else {
        print("Not found.");
    }
}

int target = 4;
int []present = [1,2,3,4,5];
int []absent = [1,2,3,5];

print("Looking for 4 in:");
print(present);
print_result(binary_search(present, 4));

print();

print("Looking for 4 in:");
print(absent);
print_result(binary_search(absent, 4));

----------
----------
File: demo/fizzbuzz.vp
bool func isDivisible(int x, int div) => (x % div == 0);

for(int i = 1; i <= 100; i+=1) {
    if(isDivisible(i, 15)) {
        print("fizzbuzz");
    }
    else if(isDivisible(i, 3)) {
        print("fizz");
    } else if(isDivisible(i, 5)) {
        print("buzz");
    } else {
        print(i);
    }
}

----------
----------
File: demo/wordcount.vp
```

```
[string: int] func count_words(string[] str) {
    [string: int] word_counts = [];
    for(string x in str) {
        if(word_counts.contains(x) == 1) {
            word_counts.add(x, word_counts[x] + 1);
        } else {
            word_counts.add(x, 1);
        }
    }
    return word_counts;
}

string [] test = ["Hello", "world", "am", "Viper", "Hello", "world", "am", "world",
"Viper", "Viper", "Viper", "ratghav"];

[string: int] counts = count_words(test);
string[] unique_keys = counts.keys();

print("Word list:");
print(test);
print();

print("Counts of keys:");
for(string key in unique_keys) {
    print("Key:");
    print(key);
    print("Count:");
    print(counts[key]);
    print();
}

----------
----------
File: buildcontainer.sh
#!/bin/sh
# Builds the MicroC Docker container, which has the dependencies needed for Viper's
compiler.

echo "Entering MicroC container, run 'exit' to leave."
docker run --rm -it -v `pwd`:/home/microc -w=/home/microc columbiasedwards/plt
----------
----------
File: test/tests/condition.vp
print(5 > 0);
print(0 < 5);
print(5 >= 5);
print(5 <= 5);
print(true && true);
print(true || false);
print(false && true);
print(false && false || true);
```

```
if (true) {
    print(true);
} else {
    print(false);
}

if (false) {
    print(false);
} else {
    print(true);
}
----------
----------
File: test/tests/contains.vp
int[] ints = [1];
print(ints.contains(1));

string[] names = ["ratghav"];
print(names.contains("ratghav"));

float[] floats = [1.2];
print(floats.contains(1.2));

char[] chars = ['A'];
print(chars.contains('A'));
----------
----------
File: test/tests/skip_abort.vp
int[] list = [1,2,3,4,5];
for (int i in list){
    if (i == 3){ skip; }
    else if (i == 4){ abort; }
}

----------
----------
File: test/tests/matching.vp
int func positive(int x) {
    int test = ??
        x < 0 : 0
      | x > 0: 10
    ?? 0;
    return test;
}

print(positive(-3));
print(positive(10));

----------
----------
File: test/tests/modulo.vp
```

```
int ten = 10;
int twenty = 20;
int one = 1;
int two = 2;


/* 10 */
int a = ten % twenty;
print(a);

/* 0 */
int b = twenty % ten;
print(b);

/* 1 */
int c = one % two;
print(c);

/* 0 */
int d = two % one;
print(d);
----------
----------
File: test/tests/function_call.vp
int func test(int x) {
    return x * x;
}

int func sumOfList(int[] l) {
    int sum = 0;
    int i;
    for (i = 0; i < len(l); i+=1) {
        sum += l[i];
    }
    return sum;
}

bool func isIndexOfGreatest(int index, int[] l) {
    if (len(l) == 0) return false;
    int i;
    for (i = 0; i < len(l); i+=1) {
        if (l[i] > l[index]) {
            return false;
        }
    }
    return true;
}

print(test(10));

int[] fib = [1, 1, 2, 3, 5, 8, 13, 21];
print(sumOfList(fib));
```

```
int[] t = [1, 2, 3, 4, 5, 6, 7, 8];
print(isIndexOfGreatest(7, t));
print(isIndexOfGreatest(2, t));
----------
----------
File: test/tests/nestedloop.vp.out
0
1
2
3
4
5
6
7
8
9
0
1
2
3
4
5
6
7
8
9
0
1
2
3
4
5
6
7
8
9
0
1
2
3
4
5
6
7
8
9
0
1
2
3
4
```

```
7
8
9

----------
----------
File: test/tests/printchar-seq.vp
print('r');
print('a');
print('t');

----------
----------
File: test/tests/matching.vp.out
0
10
----------
----------
File: test/tests/printfloat.vp.out
3
38264.1
0
-99.9
----------
----------
File: test/tests/modulo.vp.out
10
0
1
0

----------
----------
File: test/tests/condition.vp.out
1
1
1
1
1
1
0
1
1
1
----------
----------
File: test/tests/indexed_function.vp.out
4
3
----------
----------
File: test/tests/for.vp.out
```

```
Outer:
0
Inner:
0
Inner:
1
Inner:
2
Inner:
3
Inner:
4
Inner:
5
Inner:
6
Inner:
7
Inner:
8
Inner:
9
Outer:
1
Inner:
1
Inner:
2
Inner:
3
Inner:
4
Inner:
5
Inner:
6
Inner:
7
Inner:
8
Inner:
9
Outer:
2
Inner:
2
Inner:
3
Inner:
4
Inner:
5
```

```
Inner:
6
Inner:
7
Inner:
8
Inner:
9
Outer:
3
Inner:
3
Inner:
4
Inner:
5
Inner:
6
Inner:
7
Inner:
8
Inner:
9
Outer:
4
Inner:
4
Inner:
5
Inner:
6
Inner:
7
Inner:
8
Inner:
9
----------
----------
File: test/tests/list-append-access.vp
string[] names = ["trey"];
print(names[0]);

names.append("ratghav");
print(names[1]);

int[] nums = [1];
print(nums[0]);

nums.append(2);
print(nums[1]);
```

```
char[] chars = ['A'];
print(chars[0]);

chars.append('B');
print(chars[1]);

float[] nums2 = [1.0];
print(nums2[0]);

nums2.append(2.0);
print(nums2[1]);
----------
----------
File: test/tests/incr-decr.vp.out
1
2
1
----------
----------
File: test/tests/skip.vp.out
0
1
2
3
4
bad
6
7
8
9

0
skipping
1
did not skip
2
did not skip
3
did not skip
4
skipping
5
did not skip
6
did not skip
7
did not skip
8
skipping
9
did not skip
```

```
10
did not skip
11
did not skip
12
skipping
13
did not skip
14
did not skip
15
did not skip
16
skipping
17
did not skip
18
did not skip
19
did not skip
20
skipping
21
did not skip
22
did not skip
23
did not skip
24
skipping
25
did not skip
26
did not skip
27
did not skip
28
skipping
29
did not skip
30
did not skip
31
did not skip
32
skipping
33
did not skip
34
did not skip
35
did not skip
```

```
36
skipping
37
did not skip
38
did not skip
39
did not skip
40
skipping
41
did not skip
42
did not skip
43
did not skip
44
skipping
45
did not skip
46
did not skip
47
did not skip
48
skipping
49
did not skip
50
did not skip
51
did not skip
52
did not skip
53
did not skip
54
did not skip
55
did not skip
56
did not skip
57
did not skip
58
did not skip
59
did not skip
60
did not skip
61
did not skip
```

```
62
did not skip
63
did not skip
64
did not skip
65
did not skip
66
did not skip
67
did not skip
68
did not skip
69
did not skip
70
did not skip
71
did not skip
72
did not skip
73
did not skip
74
did not skip
75
did not skip
76
did not skip
77
did not skip
78
did not skip
79
did not skip
80
did not skip
81
did not skip
82
did not skip
83
did not skip
84
did not skip
85
did not skip
86
did not skip
87
did not skip
```

```
88
did not skip
89
did not skip
90
did not skip
91
did not skip
92
did not skip
93
did not skip
94
did not skip
95
did not skip
96
did not skip
97
did not skip
98
did not skip
99
did not skip
----------
----------
File: test/tests/helloworld.vp
print("Hello World!");
----------
----------
File: test/tests/for.vp
int j;
for (int i = 0; i < 5; i+=1) {
    print("Outer:");
    print(i);
    for (j = i; j < 10; j+=1) {
        print("Inner:");
        print(j);
    }
}

/*
for(int f in arr) {
    j = j + f;
}

int g;

for(g in arr) {
    j = j + g;
}
```

```
for((int, bool) tup in list) {
    if (tup[1] == true) {
        j += tup[0];
    }
}

for((int iter, int index) in list) {
    j += index;
    j += iter;
}
*/

----------
----------
File: test/tests/pow.vp.out
4
16
767376
----------
----------
File: test/tests/skip_abort.vp.out

----------
----------
File: test/tests/incr-decr.vp
int a = 1;
print(a);

a++;
print(a);

a--;
print(a);

----------
----------
File: test/tests/helloworld.vp.out
Hello World!

----------
----------
File: test/tests/cast.vp.out
9
99
888
false
1
0
8
d
h
----------
```

```
----------
File: test/tests/printbools.vp.out
1
0
----------
----------
File: test/tests/arrowfunction_call.vp
int func sum (int c, int d) => c + d;
bool func between (int i, int lowerb, int upperb) => (i > lowerb && i < upperb);


print(sum(10, 40));
print(between(5, 1, 9));

----------
----------
File: test/tests/arrays.vp.out
1
2
3
4
5
a
b
c
d
----------
----------
File: test/tests/strings.vp
string test = "bla";
print(test);
string test2 = "hi";
string test3 = test2;
print(test3);

test = "hello";
print(test);


print("@!#$*(YW(EF*H))");
print("\rwow\twow\nwow\rwow\twow\nwow");
print("11234569097654");
----------
----------
File: test/tests/index_access.vp
int[] d = [1,2,3,4];
char[] c = ['a'];

print(d[2]);
print(c[0]);

/* throws error
```

```
print(c[-1]);
*/

----------
----------
File: test/tests/attribute_call.vp.out
testing with no parameters:
42
testing with many parameters:
0
----------
----------
File: test/tests/printbools.vp
print(true);
print(false);

----------
----------
File: test/tests/variable_decl.vp.out
10
20
10
20
a
1
0
1.1
2.2
3.3

----------
----------
File: test/tests/cast.vp
print(toInt(9.34));
print(toInt('c'));
print(toString(888));
print(toString(false));
print(toBool(400));
print(toBool(""));
print(toFloat(8));
print(toChar(100));
print(toChar("hi"));

----------
----------
File: test/tests/list-append-access.vp.out
trey
ratghav
1
2
A
B
```

```
1
2
----------
----------
File: test/tests/pow.vp
float a = pow2(2.0);
print(a);

float b = pow2(a);

print(b);

float c = pow2(876.0);
print(c);



----------
----------
File: test/tests/shorthands.vp
int a = 10;
int b = 20;
int c = 0;

/* 0 */
print(c);

/* 10 */
c += a;
print(c);

/* -10 */
c -= b;
print(c);

/* 190 */
c += a * b;
print(c);

/* -10 */
c -= a * b;
print(c);

/* 100 */
c *= c;
print(c);
----------
----------
File: test/tests/shorthands.vp.out
0
10
-10
```

```
190
-10
100

----------
----------
File: test/tests/arrowfunction_call.vp.out
50
1
----------
----------
File: test/tests/attribute_call.vp
int func tester(int x) {
  return 42;
}

bool func least(int a, int b, int d, int e, int f) {
  return a > b;
}

int z = 42.tester();
print("testing with no parameters:");
print(z);

print("testing with many parameters:");
int isLeast = 5;
print(isLeast.least(6, 3, -9, 88));
----------
----------
File: test/tests/arrays.vp

int[] s = [1, 2, 3, 4, 5];
char[] c = ['a', 'b', 'c', 'd'];
int[] e;

print(s[0]);
print(s[1]);
print(s[2]);
print(s[3]);
print(s[4]);

print(c[0]);
print(c[1]);
print(c[2]);
print(c[3]);



----------
----------
File: test/tests/skip.vp
for (int i = 0; i < 10; i += 1){
```

```
    if (i == 5) {
        int x = 444;
        print("bad");
        skip;
    }
    print(i);
}
print();
for (int i = 0; i < 100; i+= 1) {
    print(i);
    if (i < 50) {
        if (i % 4 == 0){
            print("skipping");
            skip;
        }
    }
    print("did not skip");
}

----------
----------
File: test/tests/printnum-seq.vp
print(3);
print(4);
print(5);

----------
----------
File: test/tests/printfloat.vp
print(3.0);
print(38264.1);
print(0.0);
print(-99.9);

----------
----------
File: test/tests/indexed_function.vp
int[] func returnArray() {
    return [1,2,3,4];
}

print(returnArray()[3]);

[char: int] func returnDict() {
    return ['A': 1, 'B': 2, 'C': 3];
}

print(returnDict()['C']);
----------
----------
File: test/tests/abort.vp.out
testing skip and abort together:
```

```
0
1
2
3
5


---------

testing abort in a while loop:
0
1
2
3

testing skip and abort with nested loops:
aborted inner for loop
1
0
aborted inner for loop
2
0
2
1
aborted inner for loop
4
0
4
1
4
2
4
3
aborted inner for loop
5
0
5
1
5
2
5
3
5
4
aborted inner for loop
6
0
6
1
6
2
6
```

```
3
6
4
6
5
aborted inner for loop
7
0
7
1
7
2
7
3
7
4
7
5
7
6
aborted inner for loop
8
0
8
1
8
2
8
3
8
4
8
5
8
6
8
7
aborted inner for loop
9
0
9
1
9
2
9
3
9
4
9
5
9
6
```

```
9
7
9
8
aborted inner for loop
10
0
10
1
10
2
10
3
10
4
10
5
10
6
10
7
10
8
10
9
aborted inner for loop
terminated outer while loop
----------
----------
File: test/tests/printnum-seq.vp.out
3
4
5

----------
----------
File: test/tests/abort.vp
print("testing skip and abort together:");
for (int i = 0; i < 10; i += 1) {
    if (i == 4) {
        skip;
    }
    if (i > 5) {
        abort;
    }
    print(i);
}

print();
print();
print("---------");
print();
```

```
print("testing abort in a while loop:");
int a = -1;
while (a += 1 < 80) {
    if (a > 3) {
        abort;
    }
    print(a);
}

/* Throws error
if (a > 9) {
    abort;
} */

print();
print("testing skip and abort with nested loops:");
int x = -1;
while (x < 10) {
    x += 1;
    if (x == 3) {
        skip;
    }
    int i = 0;
    for (i; i < 10; i += 1) {
        if (x > i) {
            print(x);
            print(i);
        } else {
            abort;
        }
    }
    print("aborted inner for loop");
}
print("terminated outer while loop");




----------
----------
File: test/tests/variable_decl.vp
int a = 10;
print(a);
int b = 20;
print(b);
int c;
c = 10;
print(c);
int d = c + 10;
print(d);

char e = 'a';
```

```
print(e);

bool x = true;
print(x);
bool y = false;
print(y);

float xx = 1.1;
print(xx);
float zz = 2.2;
print(zz);
float aa = xx + zz;
print(aa);

----------
----------
File: test/tests/printchar-seq.vp.out
r
a
t

----------
----------
File: test/tests/index_access.vp.out
3
a

----------
----------
File: test/tests/strings.vp.out
bla
hi
hello
@!#$*(YW(EF*H))
\rwow\twow\nwow\rwow\twow\nwow
11234569097654
----------
----------
File: test/tests/function_call.vp.out
100
54
1
0

----------
----------
File: test/tests/nestedloop.vp
for(int i = 0; i < 10; i++) {
    for(int j = 0; j < 10; j++) {
        print(j);
    }
}
```

```
----------
----------
File: test/tests/contains.vp.out
1
1
1
1
----------
----------
File: test/tests/ternary.vp.out

----------
----------
File: contents.py
import os

IGNORE = [".github", "_build", "misc", "src/_build"]

import glob

top = os.getcwd()

def _get_pf_name(x):
    return x[29:]

def _check_is_exclude(x):
    for s in IGNORE:
        if x.startswith(s):
            return True
    return False

def _fetch_contents(fname):
    return open(fname, mode='r').read()

for filename in glob.iglob(os.path.join(top, '**/**'), recursive=True):
    if os.path.isfile(filename):
        printable_f_name = _get_pf_name(filename)
        if not _check_is_exclude(printable_f_name):
            print("-"*10)
            print("File: {}".format(printable_f_name))
            print(_fetch_contents(filename))
            print("-"*10)

----------
----------
File: contents.txt
----------
File: demo/factorial.vp
int func factorial(int target) {
    if(target == 1) {
        print(target);
```

```
        return target;
    }

    int res = target * factorial(target-1);
    print(res);
    return res;
}

print("Factorial of 10:");
factorial(10);

----------
----------
File: demo/calc.vp
int func calc(int a, int b, char op) {
    int res = ??
        op == '+' : (a + b)
      | op == '-' : (a - b)
      | op == '*' : (a * b)
      | op == '/' : (a / b)
    ?? a ;

    return res;
}

int result = 0;

print("10 + 20 =");
result = calc(10, 20, '+');
print(result);

print("10 - 20 =");
result = calc(10, 20, '-');
print(result);

print("10 * 20 =");
result = calc(10, 20, '*');
print(result);

print("10 / 20 =");
result = calc(10, 20, '/');
print(result);

print("10 ^ 20 = (unknown operator returns first value)");
result = calc(10, 20, '^');
print(result);

----------
----------
File: demo/bubble.vp
int[] func bubble(int[] arr) {
    for(int i = 0; i < len(arr); i++) {
```

```
        int limit = len(arr) - i - 1;
        for(int j = 0; j < limit; j++) {
            if(arr[j] > arr[j+1]) {
                int temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
                print("Step:");
                print(arr);
            }
        }
    }

    return arr;
}


int[] target = [4, 5, 10, 1, 5, 3];
print("Initial:");
print(target);

int[] sort = bubble(target);
print("Final:");
print(sort);
----------
----------
File: demo/binse.vp
int func binary_search(int []arr, int x) {
    int low = 0;
    int high = len(arr);
    int mid = 0;

    while(low <= high) {
        mid = (high + low)/2;
        if(arr[mid] < x) {
            low = mid + 1;
        } else if(arr[mid] > x) {
            high = mid - 1;
        }
        else {
            return mid;
        }
    }
    return -1;
}

nah func print_result(int loc) {
    if (loc != -1) {
        print("Found! Location:");
        print(loc);
    } else {
        print("Not found.");
    }
```

```
}

int target = 4;
int []present = [1,2,3,4,5];
int []absent = [1,2,3,5];

print("Looking for 4 in:");
print(present);
print_result(binary_search(present, 4));

print();

print("Looking for 4 in:");
print(absent);
print_result(binary_search(absent, 4));

----------
----------
File: demo/fizzbuzz.vp
bool func isDivisible(int x, int div) => (x % div == 0);

for(int i = 1; i <= 100; i+=1) {
    if(isDivisible(i, 15)) {
        print("fizzbuzz");
    }
    else if(isDivisible(i, 3)) {
        print("fizz");
    } else if(isDivisible(i, 5)) {
        print("buzz");
    } else {
        print(i);
    }
}

----------
----------
File: demo/wordcount.vp
[string: int] func count_words(string[] str) {
    [string: int] word_counts = [];
    for(string x in str) {
        if(word_counts.contains(x) == 1) {
            word_counts.add(x, word_counts[x] + 1);
        } else {
            word_counts.add(x, 1);
        }
    }
    return word_counts;
}

string [] test = ["Hello", "world", "am", "Viper", "Hello", "world", "am", "world",
"Viper", "Viper", "Viper", "ratghav"];
```

```
[string: int] counts = count_words(test);
string[] unique_keys = counts.keys();

print("Word list:");
print(test);
print();

print("Counts of keys:");
for(string key in unique_keys) {
    print("Key:");
    print(key);
    print("Count:");
    print(counts[key]);
    print();
}

----------
----------
File: buildcontainer.sh
#!/bin/sh
# Builds the MicroC Docker container, which has the dependencies needed for Viper's
compiler.

echo "Entering MicroC container, run 'exit' to leave."
docker run --rm -it -v `pwd`:/home/microc -w=/home/microc columbiasedwards/plt
----------
----------
File: test/tests/condition.vp
print(5 > 0);
print(0 < 5);
print(5 >= 5);
print(5 <= 5);
print(true && true);
print(true || false);
print(false && true);
print(false && false || true);


if (true) {
    print(true);
} else {
    print(false);
}

if (false) {
    print(false);
} else {
    print(true);
}
----------
----------
File: test/tests/contains.vp
```

```
int[] ints = [1];
print(ints.contains(1));

string[] names = ["ratghav"];
print(names.contains("ratghav"));

float[] floats = [1.2];
print(floats.contains(1.2));

char[] chars = ['A'];
print(chars.contains('A'));
----------
----------
File: test/tests/skip_abort.vp
int[] list = [1,2,3,4,5];
for (int i in list){
    if (i == 3){ skip; }
    else if (i == 4){ abort; }
}

----------
----------
File: test/tests/matching.vp
int func positive(int x) {
    int test = ??
        x < 0 : 0
      | x > 0: 10
    ?? 0;
    return test;
}

print(positive(-3));
print(positive(10));

----------
----------
File: test/tests/modulo.vp
int ten = 10;
int twenty = 20;
int one = 1;
int two = 2;


/* 10 */
int a = ten % twenty;
print(a);

/* 0 */
int b = twenty % ten;
print(b);

/* 1 */
```

```
int c = one % two;
print(c);

/* 0 */
int d = two % one;
print(d);
----------
----------
File: test/tests/function_call.vp
int func test(int x) {
    return x * x;
}

int func sumOfList(int[] l) {
    int sum = 0;
    int i;
    for (i = 0; i < len(l); i+=1) {
        sum += l[i];
    }
    return sum;
}

bool func isIndexOfGreatest(int index, int[] l) {
    if (len(l) == 0) return false;
    int i;
    for (i = 0; i < len(l); i+=1) {
        if (l[i] > l[index]) {
            return false;
        }
    }
    return true;
}

print(test(10));

int[] fib = [1, 1, 2, 3, 5, 8, 13, 21];
print(sumOfList(fib));

int[] t = [1, 2, 3, 4, 5, 6, 7, 8];
print(isIndexOfGreatest(7, t));
print(isIndexOfGreatest(2, t));
----------
----------
File: test/tests/nestedloop.vp.out
0
1
2
3
4
5
6
7
```

```
8
9
0
1
2
3
4
5
6
7
8
9
0
1
2
3
4
5
6
7
8
9
0
1
2
3
4
5
6
7
8
9
0
1
2
3
4
5
6
7
8
9
```

```
0
1
2
3
4
5
6
7
8
9
0
1
2
3
4
5
6
7
8
9
0
1
2
3
4
5
6
7
8
9
0
1
2
3
4
5
6
7
8
9

----------
----------
File: test/tests/printchar-seq.vp
print('r');
print('a');
print('t');

----------
----------
File: test/tests/matching.vp.out
0
```

```
10
----------
----------
File: test/tests/printfloat.vp.out
3
38264.1
0
-99.9
----------
----------
File: test/tests/modulo.vp.out
10
0
1
0

----------
----------
File: test/tests/condition.vp.out
1
1
1
1
1
1
0
1
1
1
----------
----------
File: test/tests/indexed_function.vp.out
4
3
----------
----------
File: test/tests/for.vp.out
Outer:
0
Inner:
0
Inner:
1
Inner:
2
Inner:
3
Inner:
4
Inner:
5
Inner:
```

```
6
Inner:
7
Inner:
8
Inner:
9
Outer:
1
Inner:
1
Inner:
2
Inner:
3
Inner:
4
Inner:
5
Inner:
6
Inner:
7
Inner:
8
Inner:
9
Outer:
2
Inner:
2
Inner:
3
Inner:
4
Inner:
5
Inner:
6
Inner:
7
Inner:
8
Inner:
9
Outer:
3
Inner:
3
Inner:
4
Inner:
```

```
5
Inner:
6
Inner:
7
Inner:
8
Inner:
9
Outer:
4
Inner:
4
Inner:
5
Inner:
6
Inner:
7
Inner:
8
Inner:
9
----------
----------
File: test/tests/list-append-access.vp
string[] names = ["trey"];
print(names[0]);

names.append("ratghav");
print(names[1]);

int[] nums = [1];
print(nums[0]);

nums.append(2);
print(nums[1]);

char[] chars = ['A'];
print(chars[0]);

chars.append('B');
print(chars[1]);

float[] nums2 = [1.0];
print(nums2[0]);

nums2.append(2.0);
print(nums2[1]);
----------
----------
File: test/tests/incr-decr.vp.out
```

```
1
2
1
----------
----------
File: test/tests/skip.vp.out
0
1
2
3
4
bad
6
7
8
9

0
skipping
1
did not skip
2
did not skip
3
did not skip
4
skipping
5
did not skip
6
did not skip
7
did not skip
8
skipping
9
did not skip
10
did not skip
11
did not skip
12
skipping
13
did not skip
14
did not skip
15
did not skip
16
skipping
17
```

```
did not skip
18
did not skip
19
did not skip
20
skipping
21
did not skip
22
did not skip
23
did not skip
24
skipping
25
did not skip
26
did not skip
27
did not skip
28
skipping
29
did not skip
30
did not skip
31
did not skip
32
skipping
33
did not skip
34
did not skip
35
did not skip
36
skipping
37
did not skip
38
did not skip
39
did not skip
40
skipping
41
did not skip
42
did not skip
43
```

```
did not skip
44
skipping
45
did not skip
46
did not skip
47
did not skip
48
skipping
49
did not skip
50
did not skip
51
did not skip
52
did not skip
53
did not skip
54
did not skip
55
did not skip
56
did not skip
57
did not skip
58
did not skip
59
did not skip
60
did not skip
61
did not skip
62
did not skip
63
did not skip
64
did not skip
65
did not skip
66
did not skip
67
did not skip
68
did not skip
69
```

```
did not skip
70
did not skip
71
did not skip
72
did not skip
73
did not skip
74
did not skip
75
did not skip
76
did not skip
77
did not skip
78
did not skip
79
did not skip
80
did not skip
81
did not skip
82
did not skip
83
did not skip
84
did not skip
85
did not skip
86
did not skip
87
did not skip
88
did not skip
89
did not skip
90
did not skip
91
did not skip
92
did not skip
93
did not skip
94
did not skip
95
```

```
did not skip
96
did not skip
97
did not skip
98
did not skip
99
did not skip
----------
----------
File: runtests.sh
#!/bin/bash
verbose=0
type="sast"

while getopts v:t: flag
do
    case "${flag}" in
        v) verbose=${OPTARG};;
        t) type=${OPTARG};;
    esac
done

llvm_tests() {
    cd test/tests
    echo "Beginning LLVM tests"
    for i in *.vp; do
        if [ ! -e "${i}.out" ]
        then
            echo "Skipping test as no file found for: ${i}.out in ${PWD}"
            continue;
        fi
        echo "Running LLVM test on: $i"
        ../../viper.native $i > a.ll
        llc -relocation-model=pic a.ll > a.s
        cc -o a.exe a.s ../../src/library.o -lm
        output=$(./a.exe)
        expectedoutput=$(cat "${i}.out")
        rm a.ll
        if [ $verbose -eq 1 ];
        then
            echo "Output: ${output}"
            echo "Expected Output: ${expectedoutput}"
        fi
        if [[ "$output" == "$expectedoutput" ]]
        then
            echo "PASSED"
        else
            echo "FAILURE: $i"
            exit 1
        fi
```

```
        echo "_____________________________________"
    done
}

sast_tests() {
    cd test/tests
    echo "Beginning SAST tests"
    for i in *.vp; do
        echo "Running SAST test on: $i"
        if [ $verbose -eq 1 ];
        then
            ../../viper.native -a $i
        else
            ../../viper.native -a $i >/dev/null 2>&1
        fi
        if [ $? -eq 0 ]
        then
            echo "PASSED"
        else
            echo "FAILURE: $i"
            exit 1
        fi
        echo "_____________________________________"
    done
}

cd src
make clean
make
mv ./viper.native ../viper.native
cd ..

echo "Test type: ${type}"
echo "Verbose enabled: ${verbose}"

if [ $type == "llvm" ]
then
    llvm_tests
else
   sast_tests
fi

----------
----------
File: rundemos.sh
#!/bin/bash
verbose=0
type="sast"

while getopts v:t: flag
do
    case "${flag}" in
```

```
        v) verbose=${OPTARG};;
        t) type=${OPTARG};;
    esac
done

llvm_tests() {
    cd demo/
    echo "Beginning LLVM tests"
    for i in *.vp; do
        if [ ! -e "${i}.out" ]
        then
            echo "Skipping test as no file found for: ${i}.out in ${PWD}"
            continue;
        fi
        echo "Running LLVM test on: $i"
        ../viper.native $i > a.ll
        llc -relocation-model=pic a.ll > a.s
        cc -o a.exe a.s ../src/library.o -lm
        output=$(./a.exe)
        expectedoutput=$(cat "${i}.out")
        rm a.ll
        if [ $verbose -eq 1 ];
        then
            echo "Output: ${output}"
            echo "Expected Output: ${expectedoutput}"
        fi
        if [[ "$output" == "$expectedoutput" ]]
        then
            echo "PASSED"
        else
            echo "FAILURE: $i"
            exit 1
        fi
        echo "_____________________________________"
    done
}

sast_tests() {
    cd demo/
    echo "Beginning SAST tests"
    for i in *.vp; do
        echo "Running SAST test on: $i"
        if [ $verbose -eq 1 ];
        then
            ../viper.native -a $i
        else
            ../viper.native -a $i >/dev/null 2>&1
        fi
        if [ $? -eq 0 ]
        then
            echo "PASSED"
        else
```

```
            echo "FAILURE: $i"
            exit 1
        fi
        echo "________________________________________"
    done
}

cd src
make clean
make
mv ./viper.native ../viper.native
cd ..

echo "Test type: ${type}"
echo "Verbose enabled: ${verbose}"

if [ $type == "llvm" ]
then
    llvm_tests
else
   sast_tests
fi

----------
----------
File: README.md
# viper
An amalgamation of our favourite programming language traits

See misc/proposal.md for more details

## Samples
Code samples for common Viper programs may be found under `examples/`

## Testing
Viper runs integration tests in order to ensure that nothing's horribly broken.

### Running Tests
* `./runtests.sh` for a general overview of LLVM tests
* `./runtests.sh -t ast` for tests that specifically check AST generation
* `./runtests.sh -v 1` for verbose outputs

### Case Creation
* Create a new test case in `test/tests`
* Create a corresponding output file in `test/tests` as well. For example, if your
case is `testo.vp`, your output file for the same would be `testo.vp.out`
* If these steps are correctly implemented, then `runtests.sh` should pick your test
and the output case up, and compare the two. Empty outputs are permitted, and are
represented by an empty `.out` file.

----------
----------
```

```
File: exec.sh
#!/bin/bash
#./runtests.sh
./viper.native $1 > a.ll

if [ $# -eq 2 ] && [ $2 = "-v" ];
then
    cat a.ll
fi

llc -relocation-model=pic a.ll > a.s
cc -o a.exe a.s src/library.o -lm
output=$(./a.exe)
echo "$output"

----------
----------
File: src/ast.ml
(* Abstract Syntax Tree and functions for printing it *)

type op = Add | Sub | Mult | Div | Mod | Equal | Neq | Less | Leq | Greater | Geq |
          And | Or | Has

type uop = Neg | Not | Incr | Decr

type typ =
    Int
  | Bool
  | Nah
  | Char
  | Float
  | String
  | Array of typ
  | Function of typ
  | Group of typ list
  | Dictionary of typ * typ

type bind = typ * string

type expr =
    IntegerLiteral of int
  | CharacterLiteral of char
  | BoolLit of bool
  | FloatLiteral of float
  | StringLiteral of string
  | ListLit of expr list
  | DictElem of expr * expr
  | DictLit of expr list

  | Id of string
  | Binop of expr * op * expr
  | Unop of uop * expr
```

```
  | Ternop of expr * expr * expr

  | Assign of string * expr
  | Deconstruct of bind list * expr
  | OpAssign of string * op * expr
  | DecAssign of typ * string * expr
  | Access of expr * expr
  | AccessAssign of expr * expr * expr

  | MatchPattern of expr list * expr
  | ConditionalPattern of expr * expr
  | PatternMatch of string * expr
  | DecPatternMatch of typ * string * expr

  | Call of string * expr list
  | AttributeCall of expr * string * expr list

  | Noexpr

type stmt =
    Block of stmt list
  | PretendBlock of stmt list
  | Expr of expr
  | Dec of typ * string
  | Return of expr
  | Skip of expr
  | Abort of expr
  | Panic of expr
  | If of expr * stmt * stmt
  | For of expr * expr * expr * stmt
  | ForIter of string * expr * stmt
  | DecForIter of typ * string * expr * stmt
  | DeconstForIter of bind list * expr * stmt
  | While of expr * stmt * stmt

type func_decl = {
    typ : typ;
    fname : string;
    formals : bind list;
    body : stmt list;
    autoreturn: bool;
  }

type program = stmt list * func_decl list

(* Pretty-printing functions *)

let string_of_op = function
    Add -> "+"
  | Sub -> "-"
  | Mult -> "*"
  | Div -> "/"
```

```
  | Mod -> "%"
  | Equal -> "=="
  | Neq -> "!="
  | Less -> "<"
  | Leq -> "<="
  | Greater -> ">"
  | Geq -> ">="
  | And -> "&&"
  | Or -> "||"
  | Has -> "has"

let string_of_uop = function
    Neg -> "-"
  | Not -> "!"
  | Incr -> "++"
  | Decr -> "--"

let rec string_of_typ = function
    Int -> "int"
  | Bool -> "bool"
  | Nah -> "nah"
  | Char -> "char"
  | Float -> "float"
  | String -> "string"
  | Array(t) -> string_of_typ t ^ "[]"
  | Function(t) -> string_of_typ t ^ " func"
  | Group(t) -> "(" ^ String.concat ", " (List.map string_of_typ t) ^ ")"
  | Dictionary(t1, t2) -> "[" ^ string_of_typ t1 ^ ":" ^ string_of_typ t2 ^ "]"

let rec string_of_expr = function
    IntegerLiteral(l) -> string_of_int l
  | CharacterLiteral(l) -> "'" ^ Char.escaped l ^ "'"
  | BoolLit(true) -> "true"
  | BoolLit(false) -> "false"
  | FloatLiteral(l) -> string_of_float l
  | StringLiteral(s) -> "\"" ^ s ^ "\""
  | ListLit(lst) -> "[" ^ String.concat ", " (List.map string_of_expr lst) ^ "]"
  | DictElem(e1, e2) -> string_of_expr e1 ^ ": " ^ string_of_expr e2
  | DictLit(lst) ->  "[" ^ String.concat ", " (List.map string_of_expr lst) ^ "]"

  | Id(s) -> s
  | Binop(e1, o, e2) ->
      string_of_expr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_expr e2
  | OpAssign(v, o, e) -> v ^ " " ^ string_of_op o ^ "= " ^ string_of_expr e
  | Ternop(e1, e2, e3) -> string_of_expr e1 ^ " ? " ^ string_of_expr e2 ^ " : " ^
string_of_expr e3
  | Unop(o, e) -> string_of_uop o ^ string_of_expr e
  | Assign(v, e) -> v ^ " = " ^ string_of_expr e
  | Access(e, l) -> string_of_expr e ^ "[" ^ string_of_expr l ^ "]" (*List.fold_left
(fun s e -> s ^ "[" ^ string_of_expr e ^ "]") "" l*)
  | DecAssign(t, v, e) -> string_of_typ t ^ " " ^ v ^ " = " ^ string_of_expr e
  | Deconstruct(v, e) ->  "(" ^ String.concat ", " (List.map snd v) ^ ") = " ^
```

```
string_of_expr e

  | AccessAssign(i, idx, e) -> string_of_expr i ^ "[" ^ string_of_expr idx ^ "]" ^ " =
" ^ string_of_expr e

  | ConditionalPattern(c, r) -> string_of_expr c ^ " : " ^ string_of_expr r
  | MatchPattern(c, b) -> "?? " ^ String.concat " | " (List.map string_of_expr c) ^ "
?? " ^ string_of_expr b
  | PatternMatch(s, e) -> s ^ " = " ^ string_of_expr e
  | DecPatternMatch(t, s, e) -> string_of_typ t ^ " " ^ s ^ " = " ^ string_of_expr e

  | Call(f, el) -> f ^ "(" ^ String.concat ", " (List.map string_of_expr el) ^ ")"
  | AttributeCall(e, f, el) -> string_of_expr e ^ "." ^ f ^ "(" ^ String.concat ", "
(List.map string_of_expr el) ^ ")"

  | Noexpr -> ""

let rec string_of_stmt = function
    Block(stmts) ->
      "{\n" ^ String.concat "" (List.map string_of_stmt stmts) ^ "}\n"
  | PretendBlock(stmts) -> "\n" ^ String.concat "" (List.map string_of_stmt stmts) ^
"\n"
  | Expr(expr) -> string_of_expr expr ^ ";\n";
  | Dec(t, v) -> string_of_typ t ^ " " ^ v ^ ";\n";
  | Return(expr) -> "return " ^ string_of_expr expr ^ ";\n";
  | Skip(expr) -> "skip " ^ string_of_expr expr ^ ";\n";
  | Abort(expr) -> "abort " ^ string_of_expr expr ^ ";\n";
  | Panic(expr) -> "panic " ^ string_of_expr expr ^ ";\n";
  | If(e, s, Block([])) -> "if (" ^ string_of_expr e ^ ")\n" ^ string_of_stmt s
  | If(e, s1, s2) ->  "if (" ^ string_of_expr e ^ ")\n" ^
      string_of_stmt s1 ^ "else\n" ^ string_of_stmt s2
  | For(e1, e2, e3, s) ->
      "for (" ^ string_of_expr e1  ^ " ; " ^ string_of_expr e2 ^ " ; " ^
      string_of_expr e3  ^ ") " ^ string_of_stmt s
  | ForIter(name, e2, s) ->
      "for (" ^ name ^ " in " ^ string_of_expr e2 ^ ") " ^ string_of_stmt s
  | DecForIter(t, name, e2, s) ->
      "for (" ^ string_of_typ t ^ " " ^ name ^ " in " ^ string_of_expr e2 ^ ") " ^
string_of_stmt s
  | DeconstForIter(p, expr, s) -> "for ((" ^ String.concat ", " (List.map snd p) ^ ")
in " ^ string_of_expr expr ^ ") " ^ string_of_stmt s
  | While(e, s, _) -> "while (" ^ string_of_expr e ^ ") " ^ string_of_stmt s

let string_of_vdecl (t, id) = string_of_typ t ^ " " ^ id ^ ";\n"

let string_of_fdecl fdecl =
  string_of_typ fdecl.typ ^ " " ^
  fdecl.fname ^ "(" ^ String.concat ", " (List.map snd fdecl.formals) ^
  ")\n{\n" ^ (if fdecl.autoreturn then "return " else "") ^
  String.concat "" (List.map string_of_stmt fdecl.body) ^
  "}\n"
```

```
let string_of_program (sts, funcs) =
  (* Do we reverse to pretty-print? Is upside down stuff an indicator of the AST *)
  String.concat "" (List.map string_of_fdecl funcs) ^ "\n" ^
  String.concat "\n" (List.map string_of_stmt (List.rev sts))


----------
----------
File: src/rhandlhand.ml
(*
Comparison checks to make sure the LHS and RHS of an expression work out
*)

open Sast
open Ast

let check_decassign ty1 s expr1 =
    match expr1 with
        (* Allows declarations of empty list and dictionary literals *)
        (Array(Nah), SListLiteral([])) -> (ty1, SDecAssign(ty1, s, expr1))
      | (Dictionary(Nah, Nah), SDictLiteral([])) -> (ty1, SDecAssign(ty1, s, expr1))
      | (Nah, SDictLiteral([])) -> (ty1, SDecAssign(ty1, s, (ty1, SDictLiteral([]))))
      | (ty2, _) when ty1 = ty2 -> (ty1, SDecAssign(ty1, s, expr1))
      | (ty2, _) -> raise (Failure ("lvalue " ^ string_of_typ ty1 ^ " not equal to
rvalue " ^ string_of_typ ty2))
  (*| DecAssign(ty, string, expr1) -> check_assign ty string (expr expr1)*)

let check_assign lvaluet rvaluet  =
      if lvaluet = rvaluet then lvaluet else raise (Failure "wrong assignment")


----------
----------
File: src/sast.ml
(* Semantically-checked Abstract Syntax Tree and functions for printing it *)

open Ast

type sexpr = typ * sx
and sx =
    SIntegerLiteral of int
  | SCharacterLiteral of char
  | SBoolLiteral of bool
  | SFloatLiteral of float
  | SStringLiteral of string
  | SListLiteral of sexpr list
  | SDictElem of sexpr * sexpr
  | SDictLiteral of sexpr list

  | SId of string
  | SBinop of sexpr * op * sexpr
  | SUnop of uop * sexpr
```

```
  | SAssign of string * sexpr
  | SDeconstruct of bind list * sexpr
  | SOpAssign of string * op * sexpr
  | SDecAssign of typ * string * sexpr
  | SAccess of sexpr * sexpr
  | SAccessAssign of sexpr * sexpr * sexpr

  | SCall of string * sexpr list
  | SAttributeCall of sexpr * string * sexpr list

  | SNoexpr

type sstmt =
    SBlock of sstmt list
  | SExpr of sexpr
  | SDec of typ * string
  | SReturn of sexpr
  | SSkip of sexpr
  | SAbort of sexpr
  | SPanic of sexpr
  | SIf of sexpr * sstmt * sstmt
  | SWhile of sexpr * sstmt * sstmt

type sfunc_decl = {
  styp : typ;
  sfname : string;
  sformals : bind list;
  sbody : sstmt list;
}

type sprogram = sstmt list * sfunc_decl list

(* Pretty-printing functions *)

let rec string_of_sexpr (t, e) =
  "(" ^ string_of_typ t ^ " : " ^ (match e with
    SIntegerLiteral(l) -> string_of_int l
  | SCharacterLiteral(l) -> "'" ^ Char.escaped l ^ "'"
  | SBoolLiteral(true) -> "true"
  | SBoolLiteral(false) -> "false"
  | SFloatLiteral(l) -> string_of_float l
  | SStringLiteral(s) -> "\"" ^ s ^ "\""
  | SListLiteral(list) -> "[" ^ String.concat ", " (List.map string_of_sexpr list) ^
"]"
  | SDictElem(e1, e2) -> string_of_sexpr e1 ^ ": " ^ string_of_sexpr e2
  | SDictLiteral(list) -> "[" ^ String.concat ", " (List.map string_of_sexpr list) ^
"]"

  | SId(s) -> s
  | SUnop(o, e) -> string_of_uop o ^ string_of_sexpr e
  | SBinop(e1, o, e2) ->
```

```
      string_of_sexpr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_sexpr e2
  | SOpAssign(v, o, e) -> v ^ " " ^ string_of_op o ^ "= " ^ string_of_sexpr e
  | SAssign(v, e) -> v ^ " = " ^ string_of_sexpr e
  | SAccess(e, l) -> string_of_sexpr e ^ "[" ^ string_of_sexpr l ^ "]"
  | SAccessAssign(i, index, e) -> string_of_sexpr i ^ "[" ^ string_of_sexpr index ^ "]
= " ^ string_of_sexpr e
  | SDecAssign(t, v, e) -> string_of_typ t ^ " " ^ v ^ " = " ^ string_of_sexpr e
  | SDeconstruct(v, e) -> "(" ^ String.concat ", " (List.map snd v) ^ ") = " ^
string_of_sexpr e

  | SCall(f, el) -> f ^ "(" ^ String.concat ", " (List.map string_of_sexpr el) ^ ")"
  | SAttributeCall(e, f, el) -> string_of_sexpr e ^ "." ^ f ^ "(" ^ String.concat ", "
(List.map string_of_sexpr el) ^ ")"

  | SNoexpr -> "")
    ^ ")"

let rec string_of_sstmt = function
    SBlock(stmts) -> "{\n" ^ String.concat "" (List.map string_of_sstmt stmts) ^ "}\n"
  | SExpr(expr) -> string_of_sexpr expr ^ ";\n"
  | SDec(t, v) -> string_of_typ t ^ " " ^ v ^ ";\n"
  | SReturn(expr) -> "return " ^ string_of_sexpr expr ^ ";\n"
  | SSkip(expr) -> "skip " ^ string_of_sexpr expr ^ ";\n"
  | SAbort(expr) -> "abort " ^ string_of_sexpr expr ^ ";\n"
  | SPanic(expr) -> "panic " ^ string_of_sexpr expr ^ ";\n"
  | SIf(e, s, SBlock([])) -> "if (" ^ string_of_sexpr e ^ ")\n" ^ string_of_sstmt s
  | SIf(e, s1, s2) ->  "if (" ^ string_of_sexpr e ^ ")\n" ^ string_of_sstmt s1 ^
"else\n" ^ string_of_sstmt s2
  | SWhile(e, s, _) -> "while (" ^ string_of_sexpr e ^ ") " ^ string_of_sstmt s

let string_of_sfdecl fdecl =
  string_of_typ fdecl.styp ^ " " ^
  fdecl.sfname ^ "(" ^ String.concat ", " (List.map snd fdecl.sformals) ^
  ")\n{\n" ^
  String.concat "" (List.map string_of_sstmt fdecl.sbody) ^
  "}\n"

let string_of_sprogram (sts, funcs) =
  String.concat "" (List.map string_of_sfdecl funcs) ^ "\n" ^
  String.concat "\n" (List.map string_of_sstmt (List.rev sts))

----------
----------
File: src/scanner.mll
(* Ocamllex scanner for viper, adapted from that of the MicroC compiler. Ref:
https://github.com/cwabbott0/microc-llvm *)

{ open Parser }

rule token = parse
  [' ' '\t' '\r' '\n'] { token lexbuf } (* Whitespace *)
| "/*"     { comment lexbuf }           (* Comments *)
```

```
| '('      { LPAREN }
| ')'      { RPAREN }
| '{'      { LBRACE }
| '}'      { RBRACE }
| '['      { ARROPEN }
| ']'      { ARRCLOSE }
| ';'      { SEMI }
| ','      { COMMA }
| '+'      { PLUS }
| '-'      { MINUS }
| '*'      { TIMES }
| '/'      { DIVIDE }
| '%'      { MODULO }
| '='      { ASSIGN }
| "+="     { PLUS_ASSIGN }
| "-="     { MINUS_ASSIGN }
| "*="     { TIMES_ASSIGN }
| "/="     { DIVIDE_ASSIGN }
| "=="     { EQ }
| "!="     { NEQ }
| '<'      { LT }
| "<="     { LEQ }
| ">"      { GT }
| ">="     { GEQ }
| "&&"     { AND }
| "||"     { OR }
| "!"      { NOT }
| "if"     { IF }
| "else"   { ELSE }
| "for"    { FOR }
| "while"  { WHILE }
| "return" { RETURN }
| "func"   { FUNC }
| "in"     { IN }
| "has"    { HAS }
| "int"    { INT }
| "char"   { CHAR }
| "bool"   { BOOL }
| "float"  { FLOAT }
| "string" { STRING }
| "nah"    { NAH }
| "=>"     { ARROW }
| "true"   { TRUE }
| "false"  { FALSE }
| "skip"   { SKIP }
| "abort"  { ABORT }
| "panic"  { PANIC }
| '?'      { QUESTION }
| "??"     { MATCH }
| '|'      { BAR }
| '.'      { DOT }
| ':'      { COLON }
```

```
| ['0'-'9']+['.']['0'-'9']+ as lxm { FLOATLIT(float_of_string lxm) }
| ['0'-'9']+ as lxm { INTLIT(int_of_string lxm) }
| ['\''](['\x20'-'\x7E'] as lxm)['\''] { CHARLIT(lxm) }
| ['\"'](['\x20'-'\x21' '\x23' - '\x7E']* as lxm)['\"'] { STRLIT(lxm) }
| ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']* as lxm { ID(lxm) }
| eof { EOF }
| _ as char { raise (Failure("illegal character " ^ Char.escaped char)) }

and comment = parse
  "*/" { token lexbuf }
| _    { comment lexbuf }

----------
----------
File: src/boolcheck.ml
(*
One function that just checks if an if-condition or while loops have a bool predicate
*)

open Ast
open Sast
open Decs

let check_bool e =
    match e with
        (ty, l) -> if ty = Bool then (ty, l) else raise (Failure "value must be a
bool")

----------
----------
File: src/Makefile
# Make sure ocamlbuild can find opam-managed packages: first run
#
# eval `opam config env`

# Easiest way to build: using ocamlbuild, which in turn uses ocamlfind

.PHONY: all
all : viper.native library.o

.PHONY: viper.native
viper.native :
    ocamlbuild -use-ocamlfind -pkgs llvm,llvm.analysis,str -cflags -w,+a-4-42-45-27-
33-11-26-39-8-10-29-40 \
        viper.native

# "make clean" removes all generated files

.PHONY : clean
clean :
    ocamlbuild -clean
    rm -rf testall.log *.diff viper scanner.ml parser.ml parser.mli
```

```
    rm -rf printbig
    rm -rf *.cmx *.cmi *.cmo *.cmx *.o *.s *.ll *.out *.exe library

# More detailed: build using ocamlc/ocamlopt + ocamlfind to locate LLVM

OBJS = ast.cmx parser.cmx scanner.cmx viper.cmx semant.cmx

viper : $(OBJS)
    ocamlfind ocamlopt -linkpkg -package llvm -package llvm.analysis $(OBJS) -o viper

scanner.ml : scanner.mll
    ocamllex scanner.mll

parser.ml parser.mli : parser.mly
    ocamlyacc parser.mly

%.cmo : %.ml
    ocamlc -c $<

%.cmi : %.mli
    ocamlc -c $<

%.cmx : %.ml
    ocamlfind ocamlopt -c -package llvm $<

### Generated by "ocamldep *.ml *.mli" after building scanner.ml and parser.ml
ast.cmo :
ast.cmx :
viper.cmo : scanner.cmo parser.cmi ast.cmo
viper.cmx : scanner.cmx parser.cmx ast.cmx
parser.cmo : ast.cmo parser.cmi
parser.cmx : ast.cmx parser.cmi
scanner.cmo : parser.cmi
scanner.cmx : parser.cmx
semant.cmo : ast.cmo
semant.cmx : ast.cmx
parser.cmi : ast.cmo

### C Standard Library

library : library.c
    cc -o library -DBUILD_TEST library.c -lm

----------
----------
File: src/viper.ml
(*
Top-level of the Viper compiler: scan & parse the input, check the resulting AST,
generate LLVM IR, and dump the module
REF: https://github.com/cwabbott0/microc-llvm/blob/master/microc.ml
*)
```

```
type action = Ast | Sast | LLVM_IR | Compile

let _ =
  let action = ref Compile in
  let input = ref "" in
  let set_action a () = action := a in
  let speclist = [
    ("-a", Arg.Unit (set_action Ast), "Pretty print the AST");
    ("-s", Arg.Unit (set_action Sast), "Pretty print the SAST");
    ("-l", Arg.Unit (set_action LLVM_IR), "Print the generated LLVM IR");
    ("-c", Arg.Unit (set_action Compile),
     "Check and print the generated LLVM IR (default)");
  ] in
  let usage_msg = "usage: ./viper.native [-a|-l] [file]" in
  Arg.parse speclist (fun s -> input := s) usage_msg;
  let channel = if !input = "" then
      stdin
    else
      open_in !input
  in
  let lexbuf = Lexing.from_channel channel in
  let ast = Parser.program Scanner.token lexbuf in
  (* this is sast, currently not used so replace _ with sast when used *)
  let desugared = Desugar.desugar ast in
  let sast = Semantdriver.check desugared in
  match !action with
    Ast -> print_string (Ast.string_of_program desugared)
  | Sast -> print_string (Sast.string_of_sprogram sast)
  | LLVM_IR -> print_string (Llvm.string_of_llmodule (Codegen.translate sast))
  | Compile -> let m = Codegen.translate sast in
    Llvm_analysis.assert_valid_module m;
    print_string (Llvm.string_of_llmodule m)

----------
----------
File: src/desugar.ml
(* Desugar module to desugar an AST *)

open Ast
let placeholderCheck ast = ast

exception SemanticException of string

let rec clean_pattern_rec e base = match e with
    ConditionalPattern(cond, exp) :: [] -> Ternop(cond, exp, base)
  | ConditionalPattern(cond, exp) :: tail -> Ternop(cond, exp, (clean_pattern_rec tail
base))
  | _ -> Noexpr


let rec clean_expression expr = match expr with
    MatchPattern(p, b) -> clean_expression (clean_pattern_rec p b)
```

```
  | PatternMatch(s, e) -> Assign(s, clean_expression e)
  | DecPatternMatch(t, s, e) -> DecAssign(t, s, clean_expression e)
  | Ternop(e, e1, e2) -> Ternop((clean_expression e), (clean_expression e1),
(clean_expression e2))

  | Binop(e1, op, e2) -> Binop((clean_expression e1), op, (clean_expression e2))
  | Unop(op, e) -> Unop(op, (clean_expression e))

  | Assign(n, e) -> Assign(n, (clean_expression e))
  | OpAssign(n, o, e) -> OpAssign(n, o, (clean_expression e))
  | DecAssign(t, n, e) -> DecAssign(t, n, (clean_expression e))
  | AccessAssign(e1, e2, e3) -> AccessAssign((clean_expression e1), (clean_expression
e2), (clean_expression e3))

  | Call(n, l) -> Call(n, (List.map clean_expression l))
  | AttributeCall(e, f, args) -> Call(f, List.map clean_expression (e::args))

  | _ -> expr

let decompose_deconstforiter n e s =
    let comparison = Binop(Id("dfi_tmp_idx"), Less, Call("len", [e]))
    in
    let (idx_t, idx_name) = List.hd n
    in
    let (el_t, el_name) = List.hd (List.rev n)
    in
    let exec = Block([ Expr(DecAssign(idx_t, idx_name, Id("dfi_tmp_idx")));
Expr(DecAssign(el_t, el_name, Access(e, Id("dfi_tmp_idx")))); s; Expr(Unop(Incr,
Id("dfi_tmp_idx")))  ])
    in
    While(comparison, exec, Expr(Noexpr))

let decompose_foriter n e s =
    let comparison = Binop(Id("dfi_tmp_idx"), Less, Call("len", [e]))
    in
    let (iterator : Ast.stmt) = Expr(Unop(Incr, Id("dfi_tmp_idx"))) in
    let exec = Block([ Expr(Assign(n, Access(e, Id("dfi_tmp_idx")))); s; iterator])
    in
    While(comparison, exec, iterator)

let decompose_decforiter t n e s =
    let comparison = Binop(Id("dfi_tmp_idx"), Less, Call("len", [e]))
    in
    let exec = Block([ Expr(DecAssign(t, n, Access(e, Id("dfi_tmp_idx")))); s;
Expr(Unop(Incr, Id("dfi_tmp_idx")))  ])
    in
    While(comparison, exec, Expr(Noexpr))

let clean_statements stmts =
    let rec clean_statement stmt = match stmt with
        Block(s) -> Block(List.map clean_statement s)
      | Expr(expr) -> Expr(clean_expression expr)
```

```
      | For(e1, e2, e3, s) -> Block([Expr(e1); While(e2, Block([ (clean_statement s);
Expr(e3)]), Expr(e3))])
      | ForIter(name, e2, s) -> Block([ Expr(DecAssign(Int, "dfi_tmp_idx",
IntegerLiteral(0))); decompose_foriter name e2 (clean_statement s)])
      | DecForIter(t, name, e2, s) -> Block([ Expr(DecAssign(Int, "dfi_tmp_idx",
IntegerLiteral(0))); decompose_decforiter t name e2 (clean_statement s)])
      | DeconstForIter(p, e, s) ->  Block([ Expr(DecAssign(Int, "dfi_tmp_idx",
IntegerLiteral(0))); decompose_deconstforiter p e (clean_statement s)])
      | While(e, s, iterator) -> While(clean_expression e, clean_statement s,
clean_statement iterator)
      | If(cond, t, f) -> If(clean_expression cond, clean_statement t, clean_statement
f)
      | _ -> stmt
    in
    (List.map clean_statement stmts)

let reshape_arrow_function fdecl = {
    typ=fdecl.typ;
    formals=fdecl.formals;
    fname=fdecl.fname;
    body = (match List.hd fdecl.body with
        Expr(e) -> [Return(e)]
      | _ -> [Return(Noexpr)]
      );
    autoreturn=false;
}
let clean_normal_function fdecl = { typ=fdecl.typ; formals=fdecl.formals;
fname=fdecl.fname; body = (clean_statements fdecl.body); autoreturn=fdecl.autoreturn;
}

let clean_function fdecl = if fdecl.autoreturn then reshape_arrow_function fdecl else
clean_normal_function fdecl

(*
let rec eliminate_ternaries stmts =
  let new_functions = [] in

  let rec eliminate_ternaries_from_expr expr = match expr with
      Ternop(e, e1, e2) -> (* Do something here, return append_function_call + do
recursively *)
    | _ -> expr
  in

  let rec clean_statement = match stmt with
    Expr(e) -> eliminate_ternaries_from_expr e
  | _ -> stmt
  in

  let cleaned_stmts = List.map clean_statement stmts in
  (cleaned_stmts, new_functions)
*)
```

```
let rec sanitize_expr e = match e with
    Binop(e1, op, e2) -> Binop((sanitize_expr e1), op, (sanitize_expr e2))
  | Unop(op, e) -> Unop(op, (sanitize_expr e))

  | Assign(n, e) -> Assign(n, (sanitize_expr e))
  | OpAssign(n, o, e) -> OpAssign(n, o, (sanitize_expr e))
  | DecAssign(t, n, e) -> DecAssign(t, n, (sanitize_expr e))
  | AccessAssign(e1, e2, e3) -> AccessAssign((sanitize_expr e1), (sanitize_expr e2),
(sanitize_expr e3))

  | Call(n, l) -> Call(n, (List.map sanitize_expr l))
  | AttributeCall(e, f, args) -> Call(f, List.map sanitize_expr (e::args))
  | Ternop(e, e1, e2) -> Id("ternop_tempvar")
  | _ -> e

let rec decompose_nested_ternary e e1 e2 = match e2 with
    Ternop(e', e1', e2') -> If(e, Expr(Assign("ternop_tempvar", e1)),
(decompose_nested_ternary e' e1' e2'))
  | _ -> If(e, Expr(Assign("ternop_tempvar", e1)), Expr(Assign("ternop_tempvar", e2)))

let rec check_for_ternary e t = match e with
    DecAssign(t, n, e) -> check_for_ternary e t
  | Ternop(e, e1, e2) -> (true, decompose_nested_ternary e e1 e2, "ternary_tmpvar", t)
  | _ -> (false, PretendBlock([]), "ternary_tempvar_none", Int)

let sanitize_ternaries stmts =
    let generate_pb_if_needed e =
        let (to_sanitize, required_stmt, name, t) = (check_for_ternary e Int) in
        if to_sanitize then PretendBlock([ Dec(t, "ternop_tempvar"); required_stmt;
Expr(sanitize_expr e) ]) else Expr(e)
    in

    (* int a = TERNARY *)
    let check_stmt stmt = match stmt with
    Expr(e) -> generate_pb_if_needed e
  | _ -> stmt

    in List.map check_stmt stmts

let sanitize_ternaries_f fdecl = { typ=fdecl.typ; formals=fdecl.formals;
fname=fdecl.fname; body = (sanitize_ternaries fdecl.body);
autoreturn=fdecl.autoreturn; }

let desugar (stmts, functions) =
  let cleaned_statements = clean_statements stmts in
  let cleaned_functions = List.map clean_function functions in
  let adjusted_statements = sanitize_ternaries cleaned_statements in
  let adjusted_functions = List.map sanitize_ternaries_f cleaned_functions in
  (adjusted_statements, adjusted_functions)

----------
----------
```

File: src/codegen.ml

```
(*
Codegen module to handle all the LLVM
*)

module L = Llvm
module A = Ast
module Str = Str
open Cast
open Sast

exception Error of string

module StringMap = Map.Make(String)

(* translate : Sast.program -> Llvm.module
   a viper program consists of statements and function defs
*)
let translate (_, functions) =
  let context    = L.global_context () in

  (* Create the LLVM compilation module into which
     we will generate code *)
  let the_module = L.create_module context "Viper" in

  (* define llytype variables *)
  let i64_t      = L.i64_type    context
  and i32_t      = L.i32_type    context
  and i16_t      = L.i16_type    context
  and i8_t       = L.i8_type     context
  and i1_t       = L.i1_type     context
  and float_t    = L.float_type  context
  and double_t   = L.double_type context
  and void_t     = L.void_type   context
  and struct_t   = L.struct_type context in
  let str_t      = L.pointer_type i8_t
  in

  (* STRUCT CODE HERE *)

  (* create hashtbl for the structs we use in standard library *)
  let struct_types:(string, L.lltype) Hashtbl.t = Hashtbl.create 3 in

  (* function to lookup struct type *)
  let find_struct_type name = try Hashtbl.find struct_types name
    with Not_found -> raise (Error "Invalid struct name")
  in

  (* declare struct type in llvm and retain its lltype in a map *)
  let declare_struct_typ name =
    let struct_type = L.named_struct_type context name in
    Hashtbl.add struct_types name struct_type
```

```
  in

  (* build struct body by passing lltypes its body contains*)
  let define_struct_body name lltypes =
    let struct_type = find_struct_type name in
    L.struct_set_body struct_type lltypes false
  in

  (* declare list struct*)
  let _ = declare_struct_typ "list"
  and _ = define_struct_body "list" [| L.pointer_type (L.pointer_type i8_t); i32_t;
i32_t; L.pointer_type i8_t |]

  (* declare dict_elem struct*)
  and _ = declare_struct_typ "dict_elem"
  and _ = define_struct_body "dict_elem" [| L.pointer_type i8_t; L.pointer_type i8_t
|]

  (* declare dict struct*)
  and _ = declare_struct_typ "dict"
  and _ = define_struct_body "dict" [| (find_struct_type "list"); L.pointer_type i8_t;
L.pointer_type i8_t |]
  in

  (* Return the LLVM lltype for a Viper type *)
  let rec ltype_of_typ = function
      A.Int                     -> i32_t
    | A.Bool                    -> i1_t
    | A.Nah                     -> void_t
    | A.Char                    -> i8_t
    | A.Float                   -> float_t
    | A.String                  -> str_t
    | A.Array(_)                -> (L.pointer_type (find_struct_type "list"))
    | A.Dictionary(_, _)        -> (L.pointer_type (find_struct_type "dict"))
    | A.Function(_)             -> raise (Error "Function lltype not implemented")
    | A.Group(_)                -> raise (Error "Group lltype not implemented")
  in

  (* Return an initial value for a declaration *)
  let rec lvalue_of_typ typ = function
      A.Int | A.Bool | A.Nah | A.Char -> L.const_int (ltype_of_typ typ) 0
    | A.Float                         -> L.const_float (ltype_of_typ typ) 0.0
    | A.String                        -> L.const_pointer_null (ltype_of_typ typ)
    | A.Array(_)                      -> L.const_pointer_null (find_struct_type
"list")
    | A.Dictionary(_, _)              -> L.const_pointer_null (find_struct_type
"dict")
    | A.Function(_)                   -> raise (Error "Function lltype not
implemented")
    | A.Group(_)                      -> raise (Error "Group llvalue not implemented")
  in
```

```
  (* BUILT-IN FUNCTIONS HERE*)
  (* These functions are either built-in C functions or defined in our library.c file
*)

  (* PRINT FUNCTIONS HERE *)

  (* Prints a char[] list *)
  let print_char_list_t : L.lltype =
    L.function_type (ltype_of_typ A.Nah) [| (L.pointer_type (find_struct_type "list"))
|] in
  let print_char_list_func : L.llvalue =
    L.declare_function "print_char_list" print_char_list_t the_module in

  (* Prints a string[] list *)
  let print_str_list_t : L.lltype =
    L.function_type (ltype_of_typ A.Nah) [| (L.pointer_type (find_struct_type "list"))
|] in
  let print_str_list_func : L.llvalue =
    L.declare_function "print_str_list" print_str_list_t the_module in

  (* Prints an int[] list *)
  let print_int_list_t : L.lltype =
    L.function_type (ltype_of_typ A.Nah) [| (L.pointer_type (find_struct_type "list"))
|] in
  let print_int_list_func : L.llvalue =
    L.declare_function "print_int_list" print_int_list_t the_module in

  (* C STANDARD LIBRARY FUNCTIONS HERE *)

  (* C's printf function *)
  let printf_t : L.lltype =
    L.var_arg_function_type i32_t [| L.pointer_type i8_t |] in
  let printf_func : L.llvalue =
    L.declare_function "printf" printf_t the_module in

  (* Raises a float to the power of 2 *)
  let pow2_t : L.lltype =
    L.function_type float_t [| float_t |] in
  let pow2_func : L.llvalue =
    L.declare_function "pow2" pow2_t the_module in

  (* BUILTIN LIST FUNCTIONS HERE*)

  (* Creates an empty list given a type string *)
  let create_list_t : L.lltype =
    L.function_type (L.pointer_type (find_struct_type "list")) [| L.pointer_type i8_t
|] in
  let create_list_func : L.llvalue =
    L.declare_function "create_list" create_list_t the_module in

  (* given a pointer to list, returns length attribute out of struct *)
  let listlen_t : L.lltype =
```

```
    L.function_type (ltype_of_typ A.Int) [| (L.pointer_type (find_struct_type "list"))
|] in
  let listlen_func : L.llvalue =
    L.declare_function "listlen" listlen_t the_module in

  (* ACCESS LIST FUNCTIONS HERE*)

  (* takes in a char[] list and an int and returns the value at the index *)
  let access_char_t : L.lltype =
    L.function_type (ltype_of_typ A.Char) [| (L.pointer_type (find_struct_type
"list")); (ltype_of_typ A.Int) |] in
  let access_char_func : L.llvalue =
    L.declare_function "access_char" access_char_t the_module in

  (* takes in a int[] list and an int and returns the value at the index *)
  let access_int_t : L.lltype =
    L.function_type (ltype_of_typ A.Int) [| (L.pointer_type (find_struct_type
"list")); (ltype_of_typ A.Int)|] in
  let access_int_func : L.llvalue =
    L.declare_function "access_int" access_int_t the_module in

  (* takes in a string[] list and an int and returns the value at the index *)
  let access_str_t : L.lltype =
    L.function_type (ltype_of_typ A.String) [| (L.pointer_type (find_struct_type
"list")); (ltype_of_typ A.Int)|] in
  let access_str_func : L.llvalue =
    L.declare_function "access_str" access_str_t the_module in

  (* takes in a float[] list and an int and returns the value at the index*)
  let access_float_t : L.lltype =
    L.function_type (ltype_of_typ A.Float) [| (L.pointer_type (find_struct_type
"list")); (ltype_of_typ A.Int)|] in
  let access_float_func : L.llvalue =
    L.declare_function "access_float" access_float_t the_module in

  (* APPEND LIST FUNCTIONS HERE *)

  (* appends a char to a char[] list *)
  let append_char_t : L.lltype =
    L.function_type (ltype_of_typ A.Nah) [| (L.pointer_type (find_struct_type
"list")); (ltype_of_typ A.Char) |] in
  let append_char_func : L.llvalue =
    L.declare_function "append_char" append_char_t the_module in

  (* appends an int to a int[] list *)
  let append_int_t : L.lltype =
    L.function_type (ltype_of_typ A.Nah) [| (L.pointer_type (find_struct_type
"list")); (ltype_of_typ A.Int) |] in
  let append_int_func : L.llvalue =
    L.declare_function "append_int" append_int_t the_module in

  (* appends a string to a string[] list *)
```

```
  let append_str_t : L.lltype =
    L.function_type (ltype_of_typ A.Nah) [| (L.pointer_type (find_struct_type
"list")); (ltype_of_typ A.String) |] in
  let append_str_func : L.llvalue =
    L.declare_function "append_str" append_str_t the_module in

  (* appends a float to a float[] list *)
  let append_float_t : L.lltype =
    L.function_type (ltype_of_typ A.Nah) [| (L.pointer_type (find_struct_type
"list")); (ltype_of_typ A.Float) |] in
  let append_float_func : L.llvalue =
    L.declare_function "append_float" append_float_t the_module in

  (* CONTAINS LIST FUNCTIONS HERE *)

  let contains_char_t : L.lltype =
    L.function_type (ltype_of_typ A.Int) [| (L.pointer_type (find_struct_type
"list")); (ltype_of_typ A.Char) |] in
  let contains_char_func : L.llvalue =
    L.declare_function "contains_char" contains_char_t the_module in

  let contains_int_t : L.lltype =
    L.function_type (ltype_of_typ A.Int) [| (L.pointer_type (find_struct_type
"list")); (ltype_of_typ A.Int) |] in
  let contains_int_func : L.llvalue =
    L.declare_function "contains_int" contains_int_t the_module in

  let contains_str_t : L.lltype =
    L.function_type (ltype_of_typ A.Int) [| (L.pointer_type (find_struct_type
"list")); (ltype_of_typ A.String) |] in
  let contains_str_func : L.llvalue =
    L.declare_function "contains_str" contains_str_t the_module in

  let contains_float_t : L.lltype =
    L.function_type (ltype_of_typ A.Int) [| (L.pointer_type (find_struct_type
"list")); (ltype_of_typ A.Float) |] in
  let contains_float_func : L.llvalue =
    L.declare_function "contains_float" contains_float_t the_module in

  (* ASSIGN LIST FUNCTIONS HERE *)

  let assign_int_t : L.lltype =
    L.function_type (ltype_of_typ A.Int) [| (L.pointer_type (find_struct_type
"list")); (ltype_of_typ A.Int); (ltype_of_typ A.Int) |] in
  let assign_int_func : L.llvalue =
    L.declare_function "assign_int" assign_int_t the_module in

  (* BUILTIN DICT FUNCTIONS HERE *)

  (* takes in a char pointer to a string of the type *)
  let create_dict_t : L.lltype =
    L.function_type (L.pointer_type (find_struct_type "dict")) [| L.pointer_type i8_t;
```

```
L.pointer_type i8_t |] in
  let create_dict_func : L.llvalue =
    L.declare_function "create_dict" create_dict_t the_module in

  (* takes in a dict pointer and a void pointer to key and val and adds pair to dict
*)
  let add_keyval_t : L.lltype =
    L.function_type (ltype_of_typ A.Nah) [| (L.pointer_type (find_struct_type
"dict")); (L.pointer_type i8_t); (L.pointer_type i8_t) |] in
  let add_keyval_func : L.llvalue =
    L.declare_function "add_keyval" add_keyval_t the_module in

  (* DICT ACCESS FUNCTIONS HERE *)

  (* takes in a dict and a char key and returns a void pointer to the value *)
  let access_char_key_t : L.lltype =
    L.function_type (L.pointer_type i8_t) [| (L.pointer_type (find_struct_type
"dict")); (ltype_of_typ A.Char) |] in
  let access_char_key_func : L.llvalue =
    L.declare_function "access_char_key" access_char_key_t the_module in

  let access_str_key_t : L.lltype =
    L.function_type (L.pointer_type i8_t) [| (L.pointer_type (find_struct_type
"dict")); (L.pointer_type (ltype_of_typ A.Char)) |] in
  let access_str_key_func : L.llvalue =
    L.declare_function "access_str_key" access_str_key_t the_module in

  (* TYPE ALLOC FUNCTIONS HERE *)

  let int_alloc_t : L.lltype =
    L.function_type (L.pointer_type i8_t) [| (ltype_of_typ A.Int) |] in
  let int_alloc_func : L.llvalue =
    L.declare_function "int_alloc_zone" int_alloc_t the_module in

  let char_alloc_t : L.lltype =
    L.function_type (L.pointer_type i8_t) [| (ltype_of_typ A.Char) |] in
  let char_alloc_func : L.llvalue =
    L.declare_function "char_alloc_zone" char_alloc_t the_module in

  let str_alloc_t : L.lltype =
    L.function_type (L.pointer_type i8_t) [| (ltype_of_typ A.String) |] in
  let str_alloc_func : L.llvalue =
    L.declare_function "str_alloc_zone" str_alloc_t the_module in

  (* CONTAINS DICT FUNCTIONS HERE *)

  let contains_str_key_t : L.lltype =
    L.function_type (ltype_of_typ A.Int) [| (L.pointer_type (find_struct_type
"dict")); (L.pointer_type (ltype_of_typ A.Char)) |] in
  let contains_str_key_func : L.llvalue =
    L.declare_function "contains_str_key" contains_str_key_t the_module in
```

```
  (* GET DICT FUNCTIONS *)

  let get_str_keys_t : L.lltype =
    L.function_type (L.pointer_type (find_struct_type "list")) [| (L.pointer_type
(find_struct_type "dict")) |] in
  let get_str_keys_func: L.llvalue =
    L.declare_function "get_str_keys" get_str_keys_t the_module in

  (* Define each function (arguments and return type) so we can
     call it even before we've created its body *)
  let function_decls : (L.llvalue * sfunc_decl) StringMap.t =
    let function_decl m fdecl =
      let name = fdecl.sfname
      and formal_types =
        Array.of_list (List.map (fun (t,_) -> ltype_of_typ t) fdecl.sformals)
      in let ftype = L.function_type (ltype_of_typ fdecl.styp) formal_types in
      StringMap.add name (L.define_function name ftype the_module, fdecl) m in
    List.fold_left function_decl StringMap.empty functions in

  (* Fill in the body of the given function *)
  let build_function_body fdecl =
    let (the_function, _) = try StringMap.find fdecl.sfname function_decls
      with Not_found -> raise (Error "function definition not found")
    in
    let builder = L.builder_at_end context (L.entry_block the_function) in

    (* define global stringptr's for format str *)
    let char_format_str = L.build_global_stringptr "%c\n" "fmt" builder
    and int_format_str = L.build_global_stringptr "%d\n" "fmt" builder
    and str_format_str = L.build_global_stringptr "%s\n" "fmt" builder
    and float_format_str = L.build_global_stringptr "%g\n" "fmt" builder
    in

    (* create empty local_vars Hashtbl to track vars llvalue *)
    let local_vars:(string, L.llvalue) Hashtbl.t = Hashtbl.create 50 in

    (* function takes in a formal binding and parameter values to add to locals map *)
    let add_formal (t, n) p =
      L.set_value_name n p;
      let local = L.build_alloca (ltype_of_typ t) n builder in
      ignore (L.build_store p local builder);
      Hashtbl.add local_vars n local;
    in

    (* iterate over the list of formal bindings and their values to add to local_vars
*)
    let _ = List.iter2 add_formal fdecl.sformals (Array.to_list (L.params
the_function)) in

    (* Return the llvalue for a variable or formal argument *)
    let lookup n = try Hashtbl.find local_vars n
      with Not_found -> raise (Error ("variable " ^ n ^ " not found in locals map"))
```

```
    in

    (* LLVM insists each basic block end with exactly one "terminator"
       instruction that transfers control.  This function runs "instr builder"
       if the current block does not already have a terminator.  Used,
       e.g., to handle the "fall off the end of the function" case. *)
    let add_terminal builder instr =
      match L.block_terminator (L.insertion_block builder) with
        Some _ -> ()
      | None -> ignore (instr builder) in

    (* maps a Viper type to a string for use in create array/dict functions *)
    let rec get_type_string e_type = match e_type with
      | A.Array (li_t)              -> get_type_string li_t
      | A.Dictionary (key_t, val_t) ->
        let key_t_str = get_type_string key_t in
        let val_t_str = get_type_string val_t in
        key_t_str ^ " " ^ val_t_str
      | A.Char    -> "char"
      | A.Int     -> "int"
      | A.Nah     -> "nah"
      | A.String  -> "string"
      | A.Float   -> "float"
      | _         -> raise (Error "type string map not implemented")
    in

    (* returns an llvalue for an expression, builds necessary instructions for
evaluation *)
    let rec expr builder ((e_type, e) : sexpr) = match e with
        SIntegerLiteral(num)      -> L.const_int (ltype_of_typ A.Int) num
      | SCharacterLiteral(chr)    -> L.const_int (ltype_of_typ A.Char) (Char.code chr)
      | SBoolLiteral(bln)         -> L.const_int (ltype_of_typ A.Bool) (if bln then 1
else 0)
      | SFloatLiteral(flt)        -> L.const_float (ltype_of_typ A.Float) flt
      | SStringLiteral(str)       -> L.build_global_stringptr str "" builder

      | SListLiteral(list)        ->
        let type_string = (get_type_string e_type) in
        let type_string_ptr = expr builder (A.String, SStringLiteral(type_string)) in
        (* create empty list llvalue *)
        let li = L.build_call create_list_func [| type_string_ptr |] "create_list"
builder in
        (* map over elements to add to list *)
        let rec append_func typ = match typ with
            A.Int         -> append_int_func
          | A.Char        -> append_char_func
          | A.String      -> append_str_func
          | A.Float       -> append_float_func
          | A.Array(arr)  -> (append_func arr)
          | A.Nah         -> raise (Error "No such thing as nah append function")
          | _             -> raise (Error "list append function not defined for type")
        in
```

```
        let appender c = L.build_call (append_func e_type) [| li; (expr builder c) |]
"" builder in
        (List.map appender list); li

      | SDictLiteral(dict_elem_list)        ->
        let dict_type_string_tup  = Str.split (Str.regexp " ") (get_type_string
e_type) in
        let key_type_string       = List.hd dict_type_string_tup in
        let key_type_string_ptr   = expr builder (A.String,
SStringLiteral(key_type_string)) in
        let val_type_string       = List.nth dict_type_string_tup 1 in
        let val_type_string_ptr   = expr builder (A.String,
SStringLiteral(val_type_string)) in
        (* create empty dict llvalue *)
        let dict = L.build_call create_dict_func [| key_type_string_ptr;
val_type_string_ptr |] "create_dict" builder in
        (* takes in a SDictElem, adds its key and value to the dict above*)
        let adder (dict_elem_t, dict_elem) = (match dict_elem with
            | SDictElem(key, value) ->
              (* takes in (typ,sx) and returns a build_call to allocate space for
values in dict*)
              let void_alloc ((z_t, z_x) as z) = (match z_t with
                  | A.Int       -> L.build_call int_alloc_func [| (expr builder z) |]
"int_alloc" builder
                  | A.Char      -> L.build_call char_alloc_func [| (expr builder z) |]
"char_alloc" builder
                  | A.String    -> L.build_call str_alloc_func [| (expr builder z) |]
"str_alloc" builder
                  | A.Array(_)  ->
                    let list_ptr = expr builder z in
                    L.build_bitcast list_ptr (L.pointer_type i8_t) (L.value_name
list_ptr) builder
                  | A.Dictionary(_, _)  ->
                    let dict_ptr = expr builder z in
                    L.build_bitcast dict_ptr (L.pointer_type i8_t) (L.value_name
dict_ptr) builder
                  | _       -> raise (Error "No alloc function for type"))
              in
              (* call alloc functions for the key and value in the dict element*)
              let key_ll = (void_alloc key) in
              let val_ll = (void_alloc value) in
              (* call add_keyval_func *)
              L.build_call add_keyval_func [| dict; key_ll; val_ll |] "" builder

            | _             -> raise(Error "Dictionary should never add non dict-
elements. How did you get here???")
          ) in

        (* map over list of dict elements to add them to dict *)
        (List.map adder dict_elem_list); dict

      | SDictElem(e1, e2)         -> raise (Error "Dict Elements should never be
```

```
encountered. How did you get here???")

      | SId s                     -> L.build_load (lookup s) s builder

      | SBinop ((A.Float,_ ) as e1, op, e2) ->
        let e1' = expr builder e1
        and e2' = expr builder e2 in
        (match op with
           A.Add     -> L.build_fadd
         | A.Sub     -> L.build_fsub
         | A.Mult    -> L.build_fmul
         | A.Div     -> L.build_fdiv
         | A.Equal   -> L.build_fcmp L.Fcmp.Oeq
         | A.Neq     -> L.build_fcmp L.Fcmp.One
         | A.Less    -> L.build_fcmp L.Fcmp.Olt
         | A.Leq     -> L.build_fcmp L.Fcmp.Ole
         | A.Greater -> L.build_fcmp L.Fcmp.Ogt
         | A.Geq     -> L.build_fcmp L.Fcmp.Oge
         | A.Mod     -> L.build_frem
         | A.And | A.Or | A.Has ->
           raise (Error "Invalid expressions for Binary operator between floats,
semant should have stopped this")
        ) e1' e2' "tmp" builder

      | SBinop (e1, op, e2) ->
        let e1' = expr builder e1
        and e2' = expr builder e2 in
        (match op with
           A.Add     -> L.build_add
         | A.Sub     -> L.build_sub
         | A.Mult    -> L.build_mul
         | A.Div     -> L.build_sdiv
         | A.And     -> L.build_and
         | A.Or      -> L.build_or
         | A.Equal   -> L.build_icmp L.Icmp.Eq
         | A.Neq     -> L.build_icmp L.Icmp.Ne
         | A.Less    -> L.build_icmp L.Icmp.Slt
         | A.Leq     -> L.build_icmp L.Icmp.Sle
         | A.Greater -> L.build_icmp L.Icmp.Sgt
         | A.Geq     -> L.build_icmp L.Icmp.Sge
         | A.Mod     -> L.build_srem
         | A.Has     -> raise (Error "Has should have been desugared. How did you get
here???")
        ) e1' e2' "tmp" builder

      | SUnop(op, ((t, var) as e)) ->
        let e' = expr builder e in
        (match op with
           A.Neg when t = A.Float -> L.build_fneg e' "tmp" builder
         | A.Neg                  -> L.build_neg e' "tmp" builder
         | A.Not                  -> L.build_not e' "tmp" builder
         | A.Incr ->
```

```
           let added = L.build_nsw_add e' (L.const_int (ltype_of_typ t) 1)
(L.value_name e') builder in
           let var_name vr = (match vr with
               | SId(s)  -> s
               | _       -> raise (Error "Incr should only be used with variables."))
           in
           L.build_store added (lookup (var_name var)) builder

         | A.Decr ->
           let added = L.build_nsw_add e' (L.const_int (ltype_of_typ t) (-1))
(L.value_name e') builder in
           let var_name vr = (match vr with
               | SId(s)  -> s
               | _       -> raise (Error "Decr should only be used with variables."))
           in
           L.build_store added (lookup (var_name var)) builder
        )

      | SAssign (s, e)            ->
        let e' = expr builder e in
        ignore(L.build_store e' (lookup s) builder); e'
      | SDeconstruct(v, e)        -> raise (Error "Deconstruct should have been
removed")
      | SOpAssign(v, o, e)        ->
        (* compute value to assign *)
        let value = expr builder (A.Nah, SBinop( (A.Nah, SId(v)), o, e)) in
        (* assign result to variable*)
        ignore(L.build_store value (lookup v) builder); value
      | SDecAssign(t, s, e)       ->
        let local_var = L.build_alloca (ltype_of_typ t) s builder in
        Hashtbl.add local_vars s local_var;
        let e' = expr builder e in
        ignore(L.build_store e' (lookup s) builder); e'

      (* typ will be a dictionary or a dict *)
      | SAccess((typ, _) as e, l) ->
        let index       = expr builder l in
        let li          = expr builder e in
        let rec access_func typ = match typ with
            A.Int         -> access_int_func
          | A.Char        -> access_char_func
          | A.String      -> access_str_func
          | A.Float       -> access_float_func
          | A.Array(arr)  -> (access_func arr)
          | A.Dictionary(key_t, key_v) -> (match key_t with
              | A.Char    -> access_char_key_func
              | A.String  -> access_str_key_func
              | _     -> raise (Error "dictionary access function not defined for key
type")
            )
          | A.Nah         -> raise (Error "No such thing as nah access function")
          | _             -> raise (Error "list access function not defined for type")
```

```
        in
        (match typ with
         (* Dictionary access requires pointer bitcasting *)
         | A.Dictionary(_, val_t) ->
           let void_ptr = L.build_call (access_func typ) [| li; index |] "access"
builder in
           (match val_t with
            | A.Int ->
              let int_ptr = L.build_bitcast void_ptr (L.pointer_type (ltype_of_typ
A.Int)) (L.value_name void_ptr) builder in
              L.build_load int_ptr (L.value_name int_ptr) builder
            | A.Char -> raise (Error "val is char")
            | A.Dictionary(_, _) ->
              L.build_bitcast void_ptr (L.pointer_type (find_struct_type "dict"))
(L.value_name void_ptr) builder
            | _       -> raise (Error "idk what this dict val type is chief"))
         (* Array access is a simple function call*)
         | A.Array(_)        -> L.build_call (access_func typ) [| li; index |]
"access" builder
         | _                 -> raise (Error "SAccess on something that isn't a list
or a dictionary not supported"))

      | SAccessAssign(i, idx, e) ->
        let li = expr builder i in
        let index = expr builder idx in
        let value = expr builder e in
        L.build_call assign_int_func [| li; index; value |] "assign_int" builder

      (* USER-EXPOSED BUILT-INS: Must be added to check in decs.ml *)

      (* print *)
      | SCall("print", []) -> let newline = expr builder (String, SStringLiteral(""))
        in L.build_call printf_func [| str_format_str ; newline |] "printf" builder

      | SCall("print", [((p_t, p_v) as params)]) -> (match p_t with
          | A.Dictionary(_,_) -> raise (Error "Printing dictionary not supported
currently")
          | A.Array(arr) -> (match arr with
              | A.Char   -> L.build_call print_char_list_func [| (expr builder params)
|] "" builder
              | A.String -> L.build_call print_str_list_func [| (expr builder params)
|] "" builder
              | A.Int    -> L.build_call print_int_list_func [| (expr builder params)
|] "" builder
              | _ -> raise (Error "Print not supported for array type"))
          | _ ->
            let print_value = (match p_t with
                | A.Float -> L.build_fpext (expr builder params) double_t "ext"
builder
                | _       -> expr builder params)
            in
            let format_str = (match p_t with
```

```
                  A.Int     -> int_format_str
                | A.Char    -> char_format_str
                | A.String  -> str_format_str
                | A.Float   -> float_format_str
                | A.Bool    -> int_format_str
                | _ -> raise (Error "print passed an invalid type"))
            in
            L.build_call printf_func [| format_str ; print_value |] "printf" builder)

      | SCall("print", params) -> raise (Error "print not supported with multiple
params currently")

      (* casts *)
      | SCall("toChar", params) -> expr builder (Cast.to_char params)
      | SCall("toInt", params) -> expr builder (Cast.to_int params)
      | SCall("toFloat", params) -> expr builder (Cast.to_float params)
      | SCall("toBool", params) -> expr builder (Cast.to_bool params)
      | SCall("toString", params) -> expr builder (Cast.to_string params)
      | SCall("toNah", params) -> expr builder (Cast.to_nah params)

      (* pow2 *)
      | SCall ("pow2", [params])    -> let value = expr builder params in
        L.build_call pow2_func [| value |] "pow2" builder

      (* append *)
      | SCall ("append", params)  ->
        let li_e = List.hd params in
        let p_e = List.nth params 1 in
        let li = expr builder li_e in
        let p = expr builder p_e in
        let append_func = (match p_e with
            | (A.Char, _)    -> append_char_func
            | (A.String, _)  -> append_str_func
            | (A.Int, _)     -> append_int_func
            | (A.Float, _)   -> append_float_func
            | _          -> raise (Error "Append parameter type invalid"))
        in
        L.build_call append_func [| li; p |] "" builder

      (* len *)
      | SCall ("len", params)  ->
        let li = expr builder (List.hd params) in
        L.build_call listlen_func [| li |] "len" builder

      (* contains *)
      | SCall ("contains", params) ->
        let typ = (fst (List.hd params)) in
        let li = expr builder (List.hd params) in
        let p = expr builder (List.nth params 1) in
        let rec contains_func typ = match typ with
            A.Int         -> contains_int_func
          | A.Char        -> contains_char_func
```

```
          | A.Float       -> contains_float_func
          | A.String      -> contains_str_func
          | A.Nah         -> raise (Error "No such thing as nah contains function")
          | A.Array(arr)  -> (contains_func arr)
          | A.Dictionary(_,_) -> contains_str_key_func
          | _             -> raise (Error "contains function not defined for type")
        in
        L.build_call (contains_func typ) [| li; p |] "contains" builder

      (* takes in a dict pointer and returns a list pointer of the keys in the dict *)
      | SCall ("keys", [params]) ->
        let typ = fst params in
        (match typ with
         | A.Dictionary(_,_) ->
           let dict = expr builder params in
           L.build_call get_str_keys_func [| dict |] "get_str_keys" builder
         | _                 -> raise (Error "Keys only supported for Dictionaries"))

      (* add is a wrapper over keyval, takes in dict, void * key, void * to val *)
      | SCall ("add", params) ->
        let ds = expr builder (List.hd params) in
        let key = (List.nth params 1) in
        let value = (List.nth params 2) in

        let void_alloc ((z_t, z_x) as z) = (match z_t with
            | A.Int       -> L.build_call int_alloc_func [| (expr builder z) |]
"int_alloc" builder
            | A.Char      -> L.build_call char_alloc_func [| (expr builder z) |]
"char_alloc" builder
            | A.String    -> L.build_call str_alloc_func [| (expr builder z) |]
"str_alloc" builder
            | A.Array(_)  ->
              let list_ptr = expr builder z in
              L.build_bitcast list_ptr (L.pointer_type i8_t) (L.value_name list_ptr)
builder
            (* cast to void pointer here *)
            | A.Dictionary(_, _)  ->
              let dict_ptr = expr builder z in
              L.build_bitcast dict_ptr (L.pointer_type i8_t) (L.value_name dict_ptr)
builder
            (* cast to void pointer here *)
            | _       -> raise (Error "No alloc function for type"))
        in
        (* cast key and value pointers to void pointers*)
        let key_void_ptr = (void_alloc key) in
        let value_void_ptr = (void_alloc value) in
        L.build_call add_keyval_func [| ds; key_void_ptr; value_void_ptr |] "" builder

      (* SCall for user defined functions *)
      | SCall (f, args)           ->
        let (fdef, fdecl) = try StringMap.find f function_decls
          with Not_found -> raise (Error "User defined function call not found")
```

```
        in
        let llargs = List.rev (List.map (expr builder) (List.rev args)) in
        let result = (match fdecl.styp with
              A.Nah -> ""
            | _ -> f ^ "_result") in
        L.build_call fdef (Array.of_list llargs) result builder

      (* clever desugar *)
      | SAttributeCall(e, f, el)  -> expr builder (A.Nah, SCall(f, e::el))
      | SNoexpr                   -> L.const_int i32_t 0
      | _ -> raise (Error "Expression match not implemented")
    in

    let rec stmt builder = function
      | SBlock sl                               -> List.fold_left stmt builder sl
      | SExpr e                                 -> ignore(expr builder e); builder
      | SDec (t, n)                             ->
        let local_var = L.build_alloca (ltype_of_typ t) n builder
        in Hashtbl.add local_vars n local_var; builder
      | SReturn e -> ignore(match fdecl.styp with
          (* Special "return nothing" instr *)
            A.Nah -> L.build_ret_void builder
          (* Build return statement *)
          | _ -> L.build_ret (expr builder e) builder );
        builder

      | SIf (predicate, then_stmt, else_stmt)   ->
        let bool_val = expr builder predicate in
        let merge_bb = L.append_block context "merge" the_function in
        let build_br_merge = L.build_br merge_bb in (* partial function *)

        let then_bb = L.append_block context "then" the_function in
        add_terminal (stmt (L.builder_at_end context then_bb) then_stmt)
          build_br_merge;

        let else_bb = L.append_block context "else" the_function in
        add_terminal (stmt (L.builder_at_end context else_bb) else_stmt)
          build_br_merge;

        ignore(L.build_cond_br bool_val then_bb else_bb builder);
        L.builder_at_end context merge_bb

      | SWhile (predicate, body, increment) ->
        let rec loop_stmt loop_bb exit_bb builder = (function
              SBlock(sl) -> List.fold_left (fun b s -> loop_stmt loop_bb exit_bb b s)
builder sl
            | SIf (predicate, then_stmt, else_stmt)   ->
              let bool_val = expr builder predicate in
              let merge_bb = L.append_block context "merge" the_function in
              let build_br_merge = L.build_br merge_bb in (* partial function *)

              let then_bb = L.append_block context "then" the_function in
```

```
              add_terminal (loop_stmt loop_bb exit_bb (L.builder_at_end context
then_bb) then_stmt)
                build_br_merge;

              let else_bb = L.append_block context "else" the_function in
              add_terminal (loop_stmt loop_bb exit_bb (L.builder_at_end context
else_bb) else_stmt)
                build_br_merge;

              ignore(L.build_cond_br bool_val then_bb else_bb builder);
              L.builder_at_end context merge_bb
            | SSkip _ ->
              let skip_bb = L.append_block context "skip" the_function in
              ignore (L.build_br skip_bb builder);
              let skip_builder = (L.builder_at_end context skip_bb) in
              add_terminal (loop_stmt loop_bb exit_bb skip_builder increment)
(L.build_br loop_bb);
              builder
            | SAbort _ -> ignore(L.build_br exit_bb builder); builder
            | _ as e -> stmt builder e) in

        let pred_bb = L.append_block context "while" the_function
        and merge_bb = L.append_block context "merge" the_function in

        ignore(L.build_br pred_bb builder);
        let body_bb = L.append_block context "while_body" the_function in
        add_terminal (loop_stmt pred_bb merge_bb (L.builder_at_end context body_bb)
body)
          (L.build_br pred_bb);

        let pred_builder = L.builder_at_end context pred_bb in
        let bool_val = expr pred_builder predicate in

        ignore(L.build_cond_br bool_val body_bb merge_bb pred_builder);
        L.builder_at_end context merge_bb
      | SSkip _     -> raise (Failure "Error: skip occurs outside of loop")
      | SAbort _    -> raise (Failure "Error: abort occurs outside of loop")
      | SPanic expr -> raise (Error "Panic statement not implemented")
      | _           -> raise (Error "Statement match for stmt builder not
implemented")
    in

    (* Build the code for each statement in the function *)
    let builder = stmt builder (SBlock fdecl.sbody) in

    (* Add a return if the last block falls off the end *)
    add_terminal builder (match fdecl.styp with
          A.Nah -> L.build_ret_void
        | A.Float -> L.build_ret (L.const_float float_t 0.0)
        | t -> L.build_ret (L.const_int (ltype_of_typ t) 0))
  in
```

```
  (* build all function bodies *)
  let _ = List.map build_function_body functions in

  (* return the LLVM module *)
  the_module
----------
----------
File: src/decs.ml
(*
   This file semantically checks all variable and function declarations in the AST.
   Duplicate variables, nah variable declarations, and function declarations
   with the same parameters throw errors.
   A scoped symbol table mapping variable names to their types is returned, along with
   a mapping of functions to their scoped formal and local variables.
*)

open Ast

module StringMap = Map.Make(String)

(* Data structure that holds scoped variables *)
type scope_table = {
  variables : typ StringMap.t;
  parent : scope_table option;
}

let rec string_of_scope scope =
  let print_bind name ty str = ("(" ^ name ^ ": " ^ (string_of_typ ty) ^ ")\n" ^ str)
in
  let this_scope = StringMap.fold (fun k t s -> print_bind k t s) scope.variables ""
in
  let parent_scope = match scope.parent with
      Some(parent) -> string_of_scope parent
    | None -> ""
  in "{" ^ this_scope ^ "}\n|\nV\n{" ^ parent_scope ^ "}"


(* Data structure that stores function declarations *)
type func_table = {
  formals : scope_table;
  locals : scope_table;
  ret_typ : typ;
}

(* Names of built-in Viper functions *)
(* User-defined functions cannot have any name in this list *)
let illegal_func_names = ["print"; "len"; "int"; "char"; "float"; "bool"; "string";
"nah"; "pow2"; "append"]

(* Retrieves a type from an ID name *)
let rec toi scope s =
  if StringMap.mem s scope.variables then
```

```
    StringMap.find s scope.variables
  else match scope.parent with
      Some(parent) -> toi parent s
    | _ -> raise (Failure ("Variable " ^ s ^ " not found"))

(* Builds a comma-separated string out of a list of parameters *)
(* For example, [int, char, bool] becomes "(int, char, bool)" *)
let rec string_of_params params = match params with
    (typ, _) :: [] -> string_of_typ typ
  | (typ, _) ::  p -> string_of_typ typ ^ ", " ^ string_of_params p
  | _              -> ""

(* Creates a key string used in mapping functions to their declarations *)
(* Strings include function name and a list of parameters to allow overloaded
functions *)
and key_string name params = name ^ " (" ^ string_of_params params ^ ")"

let rec string_of_params_built_in params = match params with
  | typ :: []   -> string_of_typ typ
  | typ :: p    -> string_of_typ typ ^ ", " ^ string_of_params_built_in p
  | _           -> ""

and key_string_built_in_functions name params = name ^ " (" ^
string_of_params_built_in params ^ ")"

(* Checks to see if a variable name is a duplicate *)
let rec is_valid_dec name scope =
  if StringMap.mem name scope.variables then
    raise (Failure ("Error: variable " ^ name ^ " is already defined"))
  else match scope.parent with
      Some(parent) -> is_valid_dec name parent
    | _ -> true

(* Adds a (name, type) pair to a symbol table *)
let add_symbol name ty scope = match ty with
    Nah -> raise (Failure ("Error: variable " ^ name ^ " declared with type nah"))
  | _  -> if is_valid_dec name scope then {
      variables = StringMap.add name ty scope.variables;
      parent = scope.parent;
    } else scope (* Never enters else clause, but still needed to avoid type error *)

let add_symbol_driver name ty scope = match ty with
    Nah -> raise (Failure ("Error: variable " ^ name ^ " declared with type nah"))
  | _  ->  {
      variables = StringMap.add name ty scope.variables;
      parent = scope.parent;
    }

(* Adds declarations from a bind into a symbol table *)
let get_bind_decs scope bind =
  let ty, name = bind in add_symbol name ty scope
```

```
(* Adds declarations from expressions into a symbol table *)
let rec get_expr_decs scope expr =
  match expr with
    Binop(e1, _, e2) ->
    let expr_list = [e1; e2] in List.fold_left get_expr_decs scope expr_list
  | Unop(_, e) -> get_expr_decs scope e
  | Ternop(e1, e2, e3) ->
    let expr_list = [e1; e2; e3] in List.fold_left get_expr_decs scope expr_list
  | OpAssign(_, _, e)
  | Assign(_, e) -> get_expr_decs scope e
  | DecAssign(ty, name, e) ->
    let updated_scope = get_expr_decs scope e in add_symbol name ty updated_scope
  | Deconstruct(b_list, e) ->
    let updated_scope = List.fold_left get_bind_decs scope b_list in get_expr_decs
updated_scope e
  | Access(arr, index) ->
    let expr_list = [arr; index] in List.fold_left get_expr_decs scope expr_list
  | AccessAssign(e1, e2, e3) ->
    let expr_list = [e1; e2; e3] in List.fold_left get_expr_decs scope expr_list
  | Call(_, expr_list) -> List.fold_left get_expr_decs scope expr_list
  | AttributeCall(e1, _, e_list) ->
    let expr_list = e1 :: e_list in List.fold_left (get_expr_decs) scope expr_list
  | _ -> scope

(* Adds declarations from statements into a symbol table *)
let rec get_stmt_decs scope stmt =
  let new_scope = {
    variables = StringMap.empty;
    parent = Some(scope);
  } in match stmt with
    Block(s_list) ->
    let _ = List.fold_left get_stmt_decs new_scope s_list in scope
  | Expr(e) -> get_expr_decs scope e
  | Dec(ty, name) -> add_symbol name ty scope
  | If(cond, then_s, else_s) ->
    let cond_scope = get_expr_decs new_scope cond in
    let _ = (get_stmt_decs cond_scope then_s, get_stmt_decs cond_scope else_s) in
scope
  | For(e1, e2, e3, s) ->
    let expr_list = [e1; e2; e3] in
    let for_scope = List.fold_left get_expr_decs new_scope expr_list in
    let _ = get_stmt_decs for_scope s in scope
  | DecForIter(ty, name, e, s) ->
    let iter_scope = add_symbol name ty new_scope in
    let for_scope = get_expr_decs iter_scope e in
    let _ = get_stmt_decs for_scope s in scope
  | While(e, s, _) ->
    let while_scope = get_expr_decs new_scope e in
    let _ = get_stmt_decs while_scope s in scope
  | PretendBlock(s_list) ->
    let _ = List.fold_left get_stmt_decs new_scope s_list in scope
  | _ -> scope
```

```
(* Driver for getting declarations from a list of statements *)
let get_vars scope s_list = List.fold_left get_stmt_decs scope s_list

(* Ensure that no functions are duplicated *)
(* Functions are invalid if they share the name with a built-in Viper function, or it
   two user-defined functions have the same name and list of formal arguments *)
let valid_func_name fd map =
  let rec unused_name name illegals = match illegals with
      [] -> ()
    | illegal_name :: _ when name = illegal_name ->
      raise (Failure ("Error: illegal function name " ^ name))
    | _ :: tail -> unused_name name tail
  in let _ = unused_name fd.fname illegal_func_names in
  let key = key_string fd.fname fd.formals in
  if StringMap.mem key map then
    raise (Failure("Error: function " ^ fd.fname ^ " is already defined with formal
arguments (" ^
                   (string_of_params fd.formals) ^ ")"))
  else key

(* Builds a function table for a function declaration *)
let build_func_table global_scope (fd : func_decl) map =
  let key = valid_func_name fd map in
  let formals_scope = List.fold_left get_bind_decs {
      variables = StringMap.empty;
      parent = None;
    } fd.formals in
  let updated_scope = {
    variables = formals_scope.variables;
    parent = Some(global_scope); } in
  let locals_scope = get_vars {
      variables = StringMap.empty;
      parent = Some(updated_scope);
    } fd.body in StringMap.add key {
    formals = updated_scope;
    locals = locals_scope;
    ret_typ = fd.typ;
  } map

(* Driver for getting declarations *)
let get_decs (s_list, f_list) =
  let globals = get_vars {
      variables = StringMap.empty;
      parent = None;
    } (List.rev s_list) in

  (* Collect declarations for Viper's built-in functions *)
  let built_in_funcs =
    let build_built_in_func_table map (name, param_typ, typ) =
      let args = List.fold_left
          (fun m f -> let param_name = ("p" ^ string_of_int (StringMap.cardinal
```

```
m.variables)) in add_symbol param_name f m) {
          variables = StringMap.empty;
          parent = None;
        } param_typ in
      let key = (key_string_built_in_functions name param_typ)
      in StringMap.add key {
        formals = args;
        locals = {
          variables = StringMap.empty;
          parent = None;
        };
        ret_typ = typ;
      } map
    in List.fold_left build_built_in_func_table StringMap.empty [
      ("print", [], Nah);
      ("print", [Int], Nah);
      ("print", [String], Nah);
      ("print", [Char], Nah);
      ("print", [Bool], Nah);
      ("print", [Float], Nah);
      ("print", [Array(Char)], Nah);
      ("print", [Array(String)], Nah);
      ("print", [Array(Int)], Nah);

      ("len", [Array(Int)], Int);
      ("len", [Array(Float)], Int);
      ("len", [Array(Bool)], Int);
      ("len", [Array(String)], Int);
      ("len", [Array(Char)], Int);

      ("append", [Array(Char); Char], Nah);
      ("append", [Array(String); String], Nah);
      ("append", [Array(Int); Int], Nah);
      ("append", [Array(Float); Float], Nah);

      ("contains", [Array(Char); Char], Int);
      ("contains", [Array(String); String], Int);
      ("contains", [Array(Float); Float], Int);
      ("contains", [Array(Int); Int], Int);
      ("contains", [Dictionary(String, Int); String], Int);

      ("add", [String; Int], Nah);
      ("add", [Dictionary(String, Int); String; Int], Nah);
      ("add", [Dictionary(Char, Int); Char; Int], Nah);

      ("keys", [Dictionary(String, Int)], Array(String));

      ("toInt", [Float], Int);
      ("toInt", [String], Int);
      ("toInt", [Char], Int);
      ("toInt", [Bool], Int);
      ("toInt", [Int], Int);
```

```
      ("toChar", [Int], Char);
      ("toChar", [String], Char);
      ("toChar", [Char], Char);
      ("toFloat", [Int], Float);
      ("toFloat", [String], Float);
      ("toFloat", [Char], Float);
      ("toFloat", [Float], Float);
      ("toBool", [Int], Bool);
      ("toBool", [String], Bool);
      ("toBool", [Char], Bool);
      ("toBool", [Bool], Bool);
      ("toString", [Int], String);
      ("toString", [Float], String);
      ("toString", [Bool], String);
      ("toString", [String], String);
      ("toNah", [Int], Nah);
      ("toNah", [String], Nah);
      ("toNah", [Char], Nah);
      ("toNah", [Float], Nah);
      ("toNah", [Bool], Nah);
      ("pow2", [Float], Float);
      ("pow2", [Int], Float);
    ]
  in

  (* Collects function tables for a list of function declarations *)
  let get_funcs f_list =
    List.fold_left (fun m f -> build_func_table globals f m) built_in_funcs f_list

  in (globals, get_funcs f_list)

----------
----------
File: src/library.c
/*
Viper's C standard library
*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>

struct list
{
    void **data;
    int size;
    int mem;
    char *type;
};

struct dict_elem
```

```
{
    void *key;
    void *val;
};

struct dict
{
    struct list *pairs;
    char *key_type;
    char *val_type;
};

struct list *create_list(char *type)
{
    struct list *inlist = malloc(sizeof(struct list));
    char *toadd = malloc(strlen(type));
    strcpy(toadd, type);
    inlist->type = toadd;
    inlist->size = 0;
    inlist->mem = 64;
    inlist->data = malloc(64);
    return inlist;
}

void realloc_check(struct list *inlist)
{
    if (inlist->mem <= (inlist->size * 8))
    {
        void **newdata = realloc(inlist->data, inlist->mem + 64);
        if (newdata == NULL)
        {
            printf("Failure to reallocate data");
            return;
        }
        inlist->data = newdata;
    }
}

void append_pair(struct list *inlist, struct dict_elem *pair)
{
    if (strcmp(inlist->type, "dict"))
    {
        printf("Can only append %s, not dict\n", inlist->type);
        return;
    }

    realloc_check(inlist);
    inlist->data[(inlist->size)] = pair;
    inlist->size = inlist->size + 1;
}

void append_str(struct list *inlist, char *str)
```

```
{
    if (strcmp(inlist->type, "string"))
    {
        printf("Can only append %s, not string\n", inlist->type);
        return;
    }

    realloc_check(inlist);
    char *toadd = malloc(strlen(str));
    strcpy(toadd, str);
    inlist->data[(inlist->size)] = toadd;
    inlist->size = inlist->size + 1;
}

void append_char(struct list *inlist, char chr)
{
    if (strcmp(inlist->type, "char"))
    {
        printf("Can only append %s, not char\n", inlist->type);
        return;
    }

    realloc_check(inlist);
    char *toadd = malloc(1);
    *toadd = chr;
    inlist->data[(inlist->size)] = toadd;
    inlist->size = inlist->size + 1;
}

void append_int(struct list *inlist, int num)
{
    if (strcmp(inlist->type, "int"))
    {
        printf("Can only append %s, not int\n", inlist->type);
        return;
    }

    realloc_check(inlist);
    int *toadd = malloc(4);
    *toadd = num;
    inlist->data[(inlist->size)] = toadd;
    inlist->size = inlist->size + 1;
}

void append_float(struct list *inlist, float flt)
{
    if (strcmp(inlist->type, "float"))
    {
        printf("Can only append %s, not float\n", inlist->type);
        return;
    }
```

```
    realloc_check(inlist);
    float *toadd = malloc(4);
    *toadd = flt;
    inlist->data[(inlist->size)] = toadd;
    inlist->size = inlist->size + 1;
}

void append_list(struct list *inlist, struct list *outlist)
{
    if (strcmp(inlist->type, "list"))
    {
        printf("Can only append %s, not list\n", inlist->type);
        return;
    }

    realloc_check(inlist);
    inlist->data[(inlist->size)] = outlist;
    inlist->size = inlist->size + 1;
}

void *access(struct list *inlist, int index)
{
    return inlist->data[index];
}

int access_int(struct list *inlist, int index)
{
    if (strcmp(inlist->type, "int"))
    {
        printf("Can only access %s, not int\n", inlist->type);
        return 0;
    }

    int *num = (int *)access(inlist, index);
    if (num == NULL)
    {
        printf("Illegal index accessed\n");
        return 0;
    }
    return *num;
}

char access_char(struct list *inlist, int index)
{
    if (strcmp(inlist->type, "char"))
    {
        printf("Can only access %s, not char\n", inlist->type);
        return 0;
    }

    char *chr = (char *)access(inlist, index);
    if (chr == NULL)
```

```
    {
        printf("Illegal index accessed\n");
        return 0;
    }
    return *chr;
}

float access_float(struct list *inlist, int index)
{
    if (strcmp(inlist->type, "float"))
    {
        printf("Can only access %s, not float\n", inlist->type);
        return 0;
    }

    float *flt = (float *)access(inlist, index);
    if (flt == NULL)
    {
        printf("Illegal index accessed\n");
        return 0;
    }
    return *flt;
}

char *access_str(struct list *inlist, int index)
{
    if (strcmp(inlist->type, "string"))
    {
        printf("Can only access %s, not string\n", inlist->type);
        return 0;
    }

    char *str = (char *)access(inlist, index);
    if (str == NULL)
    {
        printf("Illegal index accessed\n");
        return NULL;
    }
    return str;
}

int contains_int(struct list *inlist, int tocheck)
{
    if (strcmp(inlist->type, "int"))
    {
        printf("Can only check membership for %s, not int\n", inlist->type);
        return 0;
    }

    int i;
    for (i = 0; i < inlist->size; i++)
    {
```

```
        int *num = (int *)access(inlist, i);
        if (*num == tocheck)
        {
            return 1;
        }
    }
    return 0;
}

int contains_char(struct list *inlist, char tocheck)
{
    if (strcmp(inlist->type, "char"))
    {
        printf("Can only check membership for %s, not char\n", inlist->type);
        return 0;
    }

    int i;
    for (i = 0; i < inlist->size; i++)
    {
        char *chr = (char *)access(inlist, i);
        if (*chr == tocheck)
        {
            return 1;
        }
    }
    return 0;
}

int contains_float(struct list *inlist, float tocheck)
{
    if (strcmp(inlist->type, "float"))
    {
        printf("Can only check membership for %s, not float\n", inlist->type);
        return 0;
    }

    int i;
    for (i = 0; i < inlist->size; i++)
    {
        float *flt = (float *)access(inlist, i);
        if (*flt == tocheck)
        {
            return 1;
        }
    }
    return 0;
}

int contains_str(struct list *inlist, char *tocheck)
{
    if (strcmp(inlist->type, "string"))
```

```
    {
        printf("Can only check membership for %s, not string\n", inlist->type);
        return 0;
    }

    int i;
    for (i = 0; i < inlist->size; i++)
    {
        char *str = (char *)access(inlist, i);
        if (!strcmp(str, tocheck))
        {
            return 1;
        }
    }
    return 0;
}

struct list *access_list(struct list *inlist, int index)
{
    if (strcmp(inlist->type, "list"))
    {
        printf("Can only access %s, not list\n", inlist->type);
        return 0;
    }

    struct list *lst = (struct list *)access(inlist, index);
    if (lst == NULL)
    {
        printf("Illegal index accessed\n");
        return NULL;
    }
    return lst;
}

struct dict_elem *access_pair(struct list *inlist, int index)
{
    if (strcmp(inlist->type, "dict"))
    {
        printf("Can only access %s, not dict\n", inlist->type);
        return 0;
    }

    struct dict_elem *lst = (struct dict_elem *)access(inlist, index);
    if (lst == NULL)
    {
        printf("Illegal index accessed\n");
        return NULL;
    }
    return lst;
}

int assign_int(struct list *inlist, int index, int toAssign)
```

```
{
    if (strcmp(inlist->type, "int"))
    {
        printf("Can only access %s, not int\n", inlist->type);
        return 0;
    }

    if (index >= inlist->size || index < 0)
    {
        printf("Illegal index accessed\n");
        return 0;
    }

    *((int *)inlist->data[index]) = toAssign;

    return toAssign;
}

char assign_char(struct list *inlist, int index, char toAssign)
{
    if (strcmp(inlist->type, "char"))
    {
        printf("Can only access %s, not char\n", inlist->type);
        return 0;
    }

    if (index >= inlist->size || index < 0)
    {
        printf("Illegal index accessed\n");
        return 0;
    }

    *((char *)inlist->data[index]) = toAssign;

    return toAssign;
}

float assign_float(struct list *inlist, int index, float toAssign)
{
    if (strcmp(inlist->type, "float"))
    {
        printf("Can only access %s, not float\n", inlist->type);
        return 0;
    }

    if (index >= inlist->size || index < 0)
    {
        printf("Illegal index accessed\n");
        return 0;
    }

    *((float *)inlist->data[index]) = toAssign;
```

```
    return toAssign;
}

char *assign_str(struct list *inlist, int index, char *toAssign)
{
    if (strcmp(inlist->type, "string"))
    {
        printf("Can only access %s, not string\n", inlist->type);
        return 0;
    }

    if (index >= inlist->size || index < 0)
    {
        printf("Illegal index accessed\n");
        return 0;
    }

    inlist->data[index] = toAssign;

    return toAssign;
}

struct list *assign_list(struct list *inlist, int index, struct list *toAssign)
{
    if (strcmp(inlist->type, "list"))
    {
        printf("Can only access %s, not list\n", inlist->type);
        return 0;
    }

    if (index >= inlist->size || index < 0)
    {
        printf("Illegal index accessed\n");
        return 0;
    }

    inlist->data[index] = toAssign;

    return toAssign;
}

char *get_type(struct list *inlist)
{
    return inlist->type;
}

int listlen(struct list *inlist)
{
    return inlist->size;
}
```

```
struct dict *create_dict(char *ktype, char *vtype)
{
    struct dict *indict = malloc(sizeof(struct dict));
    char *ktoadd = malloc(strlen(ktype));
    strcpy(ktoadd, ktype);
    char *vtoadd = malloc(strlen(vtype));
    strcpy(vtoadd, vtype);
    indict->key_type = ktoadd;
    indict->val_type = vtoadd;
    indict->pairs = create_list("dict");
    return indict;
}

void *int_alloc_zone(int input)
{
    int *toadd = malloc(4);
    *toadd = input;
    return (void *)toadd;
}

void *char_alloc_zone(char input)
{
    char *toadd = malloc(1);
    *toadd = input;
    return (void *)toadd;
}

void *float_alloc_zone(float input)
{
    float *toadd = malloc(4);
    *toadd = input;
    return (void *)toadd;
}

void *str_alloc_zone(char *input)
{
    char *toadd = malloc(strlen(input));
    strcpy(toadd, input);
    return (void *)toadd;
}

void add_keyval(struct dict *indict, void *key, void *val)
{
    struct dict_elem *pair = malloc(sizeof(struct dict_elem));
    pair->key = key;
    pair->val = val;
    append_pair(indict->pairs, pair);
}

void *access_str_key(struct dict *indict, char *key)
{
    void *toret = NULL;
```

```
    for (int i = 0; i < indict->pairs->size; i++)
    {
        if (!strcmp((char *)access_pair(indict->pairs, i)->key, key))
        {
            toret = access_pair(indict->pairs, i)->val;
        }
    }
    return toret;
}

void *access_char_key(struct dict *indict, char key)
{
    void *toret = NULL;
    for (int i = 0; i < indict->pairs->size; i++)
    {
        if (*((char *)access_pair(indict->pairs, i)->key) == key)
        {
            toret = access_pair(indict->pairs, i)->val;
        }
    }
    return toret;
}

void print_char_list(struct list *inlist)
{
    if (strcmp(inlist->type, "char"))
    {
        printf("Can only access %s, not char\n", inlist->type);
    }

    printf("[");

    for (int i = 0; i < inlist->size; i++)
    {
        printf("\'%c\'", access_char(inlist, i));

        if (i < inlist->size - 1)
        {
            printf(", ");
        }
    }

    printf("]\n");
}

void print_int_list(struct list *inlist)
{
    if (strcmp(inlist->type, "int"))
    {
        printf("Can only access %s, not int\n", inlist->type);
    }
```

```
    printf("[");

    for (int i = 0; i < inlist->size; i++)
    {
        printf("%d", access_int(inlist, i));

        if (i < inlist->size - 1)
        {
            printf(", ");
        }
    }

    printf("]\n");
}

void print_str_list(struct list *inlist)
{
    if (strcmp(inlist->type, "string"))
    {
        printf("Can only access %s, not string\n", inlist->type);
    }

    printf("[");

    for (int i = 0; i < inlist->size; i++)
    {
        printf("\"%s\"", access_str(inlist, i));

        if (i < inlist->size - 1)
        {
            printf(", ");
        }
    }

    printf("]\n");
}

int contains_str_key(struct dict *indict, char *key)
{
    for (int i = 0; i < indict->pairs->size; i++)
    {
        if (!strcmp((char *)access_pair(indict->pairs, i)->key, key))
        {
            return 1;
        }
    }
    return 0;
}

int contains_char_key(struct dict *indict, char key)
{
    for (int i = 0; i < indict->pairs->size; i++)
```

```
    {
        if (*((char *)access_pair(indict->pairs, i)->key) != key)
        {
            return 1;
        }
    }
    return 0;
}

struct list *get_char_keys(struct dict *indict)
{
    if (strcmp(indict->key_type, "char"))
    {
        printf("Can only access %s, not char\n", indict->key_type);
        return 0;
    }

    struct list *keys = create_list("char");
    for (int i = 0; i < indict->pairs->size; i++)
    {
        char toadd = *((char *)access_pair(indict->pairs, i)->key);
        if (!contains_char(keys, toadd)){
            append_char(keys, toadd);
        }
    }
    return keys;
}

struct list *get_str_keys(struct dict *indict)
{
    if (strcmp(indict->key_type, "string"))
    {
        printf("Can only access %s, not string\n", indict->key_type);
        return 0;
    }

    struct list *keys = create_list("string");
    for (int i = 0; i < indict->pairs->size; i++)
    {
        char *toadd = (char *)access_pair(indict->pairs, i)->key;
        if (!contains_str(keys, toadd)){
            append_str(keys, toadd);
        }
    }
    return keys;
}

void remove_str_key(struct dict *indict, char *key)
{
    int index_to_remove = -1;
    for (int i = 0; i < indict->pairs->size; i++)
    {
```

```
        if (!strcmp((char *)access_pair(indict->pairs, i)->key, key))
        {
            access_pair(indict->pairs, i)->key = NULL;
            index_to_remove = i;
        }
    }

    for (int i = index_to_remove; i < indict->pairs->size - 1; i++)
    {
        access_pair(indict->pairs, i)->key = access_pair(indict->pairs, i + 1)->key;
        access_pair(indict->pairs, i)->val = access_pair(indict->pairs, i + 1)->val;
    }

    indict->pairs->size -= 1;
}

void remove_char_key(struct dict *indict, char key)
{
    int index_to_remove = -1;
    for (int i = 0; i < indict->pairs->size; i++)
    {
        if (*((char *)access_pair(indict->pairs, i)->key) != key)
        {
            access_pair(indict->pairs, i)->key = NULL;
            index_to_remove = i;
        }
    }

    for (int i = index_to_remove; i < indict->pairs->size - 1; i++)
    {
        access_pair(indict->pairs, i)->key = access_pair(indict->pairs, i + 1)->key;
        access_pair(indict->pairs, i)->val = access_pair(indict->pairs, i + 1)->val;
    }

    indict->pairs->size -= 1;
}

int void_to_int(void *v_ptr)
{
    return *((int *)v_ptr);
}

float pow2(float base)
{
    return pow(base, 2);
}

int imax(int first, int second)
{
    return (int)fmax(first, second);
}
```

```
char cmax(char first, char second)
{
    return (char)fmax(first, second);
}

int imin(int first, int second)
{
    return (int)fmin(first, second);
}

char cmin(char first, char second)
{
    return (char)fmin(first, second);
}

float ptrunc(float input, int decs)
{
    return floor(pow(10, decs) * input) / pow(10, decs);
}

// TODO: Remove these
void testy()
{
    int a = 0;
    a++;
    a--;
}

void floaty()
{
}

void listy()
{
    struct list *chrlist = create_list("char");
    append_char(chrlist, 'a');
    append_char(chrlist, 'b');
    append_char(chrlist, 'c');
    access_char(chrlist, 0);
    // print_char_list(chrlist);
}

void dicty()
{
    struct dict *chardict = create_dict("char", "int");
    // void* test1 = char_alloc_zone('A');
    // void* test2 = int_alloc_zone(0);
    add_keyval(chardict, char_alloc_zone('A'), int_alloc_zone(3));
    // printf("access_char_key(chrdict, 'A'): %d\n",
void_to_int(access_char_key(testdict, 'A')));
    // printf("access_char_key(chrdict, 'A'): %d\n", *((int *)
(access_char_key(testdict, 'A'))));
```

```
    struct dict *testdict = create_dict("char", "dict");
    add_keyval(testdict, char_alloc_zone('Z'), (void *)chardict);
    struct dict *char_dict_ptr = (struct dict *)access_char_key(testdict, 'Z');
    printf("access_char_key(chrdict, 'A'): %d\n", *((int
*)access_char_key(char_dict_ptr, 'A')));
}

#ifdef BUILD_TEST
int main(void)
{
    //dicty();
    printf("sqrt(100) = %f\n", sqrt(100));
    printf("sqrt(100.7) = %f\n", sqrt(100.7));
    printf("pow(100, 3) = %f\n", pow(100, 3));
    printf("pow(1.7, 12) = %f\n", pow(1.7, 12));
    printf("pow2(3) = %f\n", pow2(3));
    printf("floor(2.4) = %f\n", floor(2.4));
    printf("ceil(2.4) = %f\n", ceil(2.4));
    printf("round(2.4) = %f\n", round(2.4));
    printf("round(2.4) = %f\n", round(2.6));
    printf("imax(2, 7) = %d\n", imax(2, 7));
    printf("fmax(2.6, 3.5) = %f\n", fmax(2.6, 3.5));
    printf("cmax('k', 'm') = %c\n", cmax('k', 'm'));
    printf("trunc(5.121212) = %f\n", trunc(5.121212));
    printf("ptrunc(5.121212, 3) = %f\n", ptrunc(5.121212, 3));

    struct list *charlist = create_list("char");
    append_char(charlist, 'a');
    append_char(charlist, 'b');
    append_char(charlist, 'c');

    struct list *mylist = create_list("int");
    append_int(mylist, 0);
    append_str(mylist, "yooo");
    append_int(mylist, 1);
    append_int(mylist, 2);
    append_int(mylist, 3);
    append_int(mylist, 4);
    append_int(mylist, 5);
    append_int(mylist, 6);
    append_int(mylist, 7);
    append_int(mylist, 8);
    append_float(mylist, 4.5);
    append_char(mylist, 'c');
    struct list *otherlist = create_list("list");
    append_int(otherlist, 1);
    append_list(otherlist, mylist);

    printf("Type: %s\n", get_type(mylist));
    printf("Type: %s\n", get_type(otherlist));
    printf("Length: %d\n", listlen(mylist));
```

```
    printf("Length: %d\n", listlen(otherlist));

    printf("@ index 0: %d\n", access_int(mylist, 0));
    printf("@ index 5: %d\n", access_int(mylist, 5));
    printf("@ Illegal Index: %d\n", access_int(mylist, 1000));

    struct list *accessed = access_list(otherlist, 0);
    printf("@ index 5: %d\n", access_int(accessed, 5));

    printf("contains_int(mylist, 5): %d\n", contains_int(mylist, 5));
    printf("contains_int(mylist, 10): %d\n", contains_int(mylist, 10));

    int test = assign_int(mylist, 1, 7);
    printf("Test assign return val: %d\n", test);
    printf("@ index 1: %d\n", access_int(mylist, 1));
    printf("contains_int(mylist, 1): %d\n", contains_int(mylist, 1));

    struct list *evillist = create_list("int");
    append_int(evillist, 100);
    append_int(evillist, 200);
    append_int(evillist, 300);
    struct list *testo = assign_list(otherlist, 0, evillist);
    printf("Test assign return list: %d\n", access_int(evillist, 0));
    struct list *sameto = access_list(otherlist, 0);
    printf("Test accessed list: %d\n", access_int(evillist, 1));

    // struct dict *testdict = create_dict("string", "int");
    // //void* test1 = str_alloc_zone("yo");
    //void* test2 = int_alloc_zone(1);
    //add_keyval(testdict, str_alloc_zone("yo"), int_alloc_zone(1));
    //add_keyval(testdict, str_alloc_zone("gabba"), int_alloc_zone(12));
    //printf("access_str_key(testdict, \"yo\"): %d\n", *((int
*)access_str_key(testdict, "yo")));
    //printf("access_str_key(testdict, \"yo\"): %d\n", *((int
*)access_str_key(testdict, "gabba")));

    struct dict *chrdict = create_dict("char", "int");
    add_keyval(chrdict, char_alloc_zone('A'), int_alloc_zone(7));
    add_keyval(chrdict, char_alloc_zone('B'), int_alloc_zone(8));
    struct list *chrkeys = get_char_keys(chrdict);
    print_char_list(chrkeys);
    printf("access_char_key(chrdict, 'A'): %d\n", *((int *)access_char_key(chrdict,
'A')));
    printf("access_char_key(chrdict, 'B'): %d\n", *((int *)access_char_key(chrdict,
'B')));
    printf("contains_char_key(chrdict, 'A'): %d\n", contains_char_key(chrdict, 'A'));

    struct dict *strdict = create_dict("string", "int");
    add_keyval(strdict, str_alloc_zone("AAA"), int_alloc_zone(7));
    add_keyval(strdict, str_alloc_zone("BBB"), int_alloc_zone(8));
    add_keyval(strdict, str_alloc_zone("CCC"), int_alloc_zone(8));
    struct list *strkeys = get_str_keys(strdict);
```

```
    print_str_list(strkeys);
    printf("contains_str_key(strdict, 'CCC'): %d\n", contains_str_key(strdict,
"CCC"));
    remove_str_key(strdict, "CCC");
    struct list *strkeys2 = get_str_keys(strdict);
    print_str_list(strkeys2);
    printf("contains_str_key(strdict, 'CCC'): %d\n", contains_str_key(strdict,
"CCC"));

    // struct list *intlist = create_list("int");
    // append_int(intlist, 1);
    // append_int(intlist, 2);
    // append_int(intlist, 3);
    // print_int_list(intlist);

    // struct list *strlist = create_list("string");
    // append_str(strlist, "aaa");
    // append_str(strlist, "bbb");
    // append_str(strlist, "ccc");
    // print_str_list(strlist);
}
#endif

----------
----------
File: src/cast.ml
(*
This file handles Viper's type casting, defining what types can and cannot be cast to
another.
*)

open Ast
open Sast

let verify_params params func =
  if List.length params != 1 then
    raise (Failure ("Error: " ^ func ^ "() requires one argument"))
  else List.hd params

let to_char params =
  match verify_params params "char" with
      (_, (SCharacterLiteral(_) as c)) -> (Char, c)
    | (_, SIntegerLiteral(i)) when i > -1 && i < 256 -> (Char,
SCharacterLiteral(char_of_int i))
    | (_, SIntegerLiteral(i)) -> raise (Failure("Error: integer " ^ string_of_int i ^
" cannot be cast to char,
        it must have a value between 0 and 255"))
    | (_, SStringLiteral("")) -> raise (Failure("Error: empty string cannot be cast to
char"))
    | (_, SStringLiteral(s)) -> (Char, SCharacterLiteral(String.get s 0))
    | (typ, _) -> raise (Failure ("Error: type " ^ string_of_typ typ ^ " cannot be
cast to char"))
```

```
let to_int params =
  match verify_params params "int" with
      (_, (SIntegerLiteral(_) as i)) -> (Int, i)
    | (_, SCharacterLiteral(c)) -> (Int, SIntegerLiteral(int_of_char c))
    | (_, SFloatLiteral(f)) -> (Int, SIntegerLiteral(int_of_float f))
    | (_, SBoolLiteral(true)) -> (Int, SIntegerLiteral(1))
    | (_, SBoolLiteral(false)) -> (Int, SIntegerLiteral(0))
    | (_, SStringLiteral(s)) -> (
        try (Int, SIntegerLiteral(int_of_string s))
        with Failure _ -> raise (Failure ("Error: string \"" ^ s ^ "\" cannot be cast
to int")))
    | (typ, _) -> raise (Failure ("Error: type " ^ string_of_typ typ ^ " cannot be
cast to int"))

let to_float params =
  match verify_params params "float" with
      (_, (SFloatLiteral(_) as f)) -> (Float, f)
    | (_, SIntegerLiteral(i)) -> (Float, SFloatLiteral(float_of_int i))
    | (_, SCharacterLiteral(c)) -> (Float, SFloatLiteral(float_of_int (int_of_char
c)))
    | (_, SStringLiteral(s)) -> (
      try (Int, SFloatLiteral(float_of_string s))
      with Failure _ -> raise (Failure ("Error: string \"" ^ s ^ "\" cannot be cast to
float")))
    | (typ, _) -> raise (Failure ("Error: type " ^ string_of_typ typ ^ " cannot be
cast to float"))

let to_bool params =
  match verify_params params "bool" with
      (_, (SBoolLiteral(_) as b)) -> (Bool, b)
    | (_, SCharacterLiteral(c)) when c = '\x00' -> (Bool, SBoolLiteral(false))
    | (_, SIntegerLiteral(i)) when i = 0 -> (Bool, SBoolLiteral(false))
    | (_, SFloatLiteral(f)) when f = 0.0 -> (Bool, SBoolLiteral(false))
    | (_, SStringLiteral(s)) when s = "" -> (Bool, SBoolLiteral(false))
    | (Nah, _) -> (Bool, SBoolLiteral(false))
    | (_, SCharacterLiteral(_))
    | (_, SIntegerLiteral(_))
    | (_, SFloatLiteral(_))
    | (_, SStringLiteral(_)) -> (Bool, SBoolLiteral(true))
    | (typ, _) -> raise (Failure ("Error: type " ^ string_of_typ typ ^ " cannot be
cast to bool"))

let to_string params =
  match verify_params params "string" with
      (_, (SStringLiteral(_) as s)) -> (String, s)
    | (_, SCharacterLiteral(c)) -> (String, SStringLiteral(String.make 1 c))
    | (_, SIntegerLiteral(i)) -> (String, SStringLiteral(string_of_int i))
    | (_, SFloatLiteral(f)) -> (String, SStringLiteral(string_of_float f))
    | (_, SBoolLiteral(true)) -> (String, SStringLiteral("true"))
    | (_, SBoolLiteral(false)) -> (String, SStringLiteral("false"))
    | (Nah, _) -> (String, SStringLiteral("nah"))
```

```
    | (typ, _) -> raise (Failure ("Error: type " ^ string_of_typ typ ^ " cannot be
cast to string"))

let to_nah params =
  match verify_params params "nah" with
      _ -> (Nah, SNoexpr)

----------
----------
File: src/parser.mly
%{
open Ast
%}

%token SEMI LPAREN RPAREN LBRACE RBRACE COMMA ARROPEN ARRCLOSE DOT
%token PLUS MINUS TIMES DIVIDE ASSIGN NOT PLUS_ASSIGN MINUS_ASSIGN TIMES_ASSIGN
DIVIDE_ASSIGN MODULO HAS QUESTION COLON
%token EQ NEQ LT LEQ GT GEQ TRUE FALSE AND OR MATCH BAR
%token RETURN IF ELSE FOR WHILE INT CHAR BOOL FLOAT STRING NAH FUNC IN ARROW PANIC
%token SKIP ABORT
%token <int> INTLIT
%token <char> CHARLIT
%token <float> FLOATLIT
%token <string> STRLIT
%token <string> ID
%token EOF

%nonassoc NOELSE
%nonassoc ELSE
%right ASSIGN
%left QUESTION COLON MATCH
%left BAR
%left OR
%left AND
%left EQ NEQ
%left LT GT LEQ GEQ
%right HAS
%left PLUS MINUS PLUS_ASSIGN MINUS_ASSIGN
%left TIMES DIVIDE MODULO TIMES_ASSIGN DIVIDE_ASSIGN
%nonassoc INCR DECR
%right NOT NEG
%left ARROPEN ARRCLOSE DOT

%start program
%type <Ast.program> program

%%

program:
  decls EOF { $1 }

decls:
```

```
   /* nothing */ { [], [] }
 | decls fdecl { fst $1, ($2 :: snd $1) }
 | decls stmt { ($2 :: fst $1), snd $1 }

fdecl:
   typ ID LPAREN formals_opt RPAREN LBRACE stmt_list RBRACE
     { { typ = $1;
       fname = $2;
       formals = $4;
     body = List.rev $7;
     autoreturn = false } }
    | typ ID LPAREN formals_opt RPAREN ARROW stmt
     { { typ = $1;
     fname = $2;
     formals = $4;
     body = [$7];
     autoreturn = true } }

formals_opt:
    /* nothing */ { [] }
  | formal_list   { List.rev $1 }

formal_list:
    typ ID                   { [($1,$2)] }
  | formal_list COMMA typ ID { ($3,$4) :: $1 }

typ:
    INT { Int }
  | BOOL { Bool }
  | NAH { Nah }
  | CHAR { Char }
  | FLOAT { Float }
  | STRING { String }
  | typ FUNC { Function($1) }
  | typ ARROPEN ARRCLOSE { Array($1) }
  | LPAREN type_list RPAREN { Group($2) }
  | ARROPEN typ COLON typ ARRCLOSE { Dictionary($2, $4) }

type_list:
    typ            { [$1] }
  | typ COMMA type_list { $1 :: $3 }

stmt_list:
    /* nothing */  { [] }
  | stmt_list stmt { $2 :: $1 }

stmt:
    expr SEMI { Expr $1 }
  | typ ID SEMI { Dec($1, $2) }
  | RETURN SEMI { Return Noexpr }
  | RETURN expr SEMI { Return $2 }
  | SKIP SEMI { Skip Noexpr }
```

```
  | ABORT SEMI { Abort Noexpr }
  | PANIC expr SEMI { Panic $2 }
  | LBRACE stmt_list RBRACE { Block(List.rev $2) }
  | IF LPAREN expr RPAREN stmt %prec NOELSE { If($3, $5, Block([])) }
  | IF LPAREN expr RPAREN stmt ELSE stmt    { If($3, $5, $7) }
  | FOR LPAREN expr_opt SEMI expr SEMI expr_opt RPAREN stmt
     { For($3, $5, $7, $9) }
  | FOR LPAREN ID IN expr RPAREN stmt { ForIter($3, $5, $7) }
  | FOR LPAREN typ ID IN expr RPAREN stmt { DecForIter($3, $4, $6, $8) }
  | FOR LPAREN LPAREN formal_list RPAREN IN expr RPAREN stmt { DeconstForIter($4, $7,
$9) }
  | WHILE LPAREN expr RPAREN stmt { While($3, $5, Expr(Noexpr)) }

expr_opt:
    /* nothing */ { Noexpr }
  | expr          { $1 }

expr:
    INTLIT          { IntegerLiteral($1) }
  | CHARLIT         { CharacterLiteral($1) }
  | FLOATLIT        { FloatLiteral($1) }
  | STRLIT          { StringLiteral($1) }
  | TRUE             { BoolLit(true) }
  | FALSE            { BoolLit(false) }
  | ID               { Id($1) }
  | list_exp         { $1 }
  | dict_exp         { $1 }

  | expr PLUS   expr { Binop($1, Add,   $3) }
  | expr MINUS  expr { Binop($1, Sub,   $3) }
  | expr TIMES  expr { Binop($1, Mult,  $3) }
  | expr DIVIDE expr { Binop($1, Div,   $3) }
  | expr MODULO expr { Binop($1, Mod, $3) }

  | ID PLUS_ASSIGN expr { OpAssign($1, Add, $3) }
  | ID MINUS_ASSIGN expr { OpAssign($1, Sub, $3) }
  | ID TIMES_ASSIGN expr { OpAssign($1, Mult, $3) }
  | ID DIVIDE_ASSIGN expr { OpAssign($1, Div, $3) }

  | expr EQ     expr { Binop($1, Equal, $3) }
  | expr NEQ    expr { Binop($1, Neq,   $3) }
  | expr LT     expr { Binop($1, Less,  $3) }
  | expr LEQ    expr { Binop($1, Leq,   $3) }
  | expr GT     expr { Binop($1, Greater, $3) }
  | expr GEQ    expr { Binop($1, Geq,   $3) }
  | expr AND    expr { Binop($1, And,   $3) }
  | expr OR     expr { Binop($1, Or,    $3) }
  | expr HAS     expr { Binop($1, Has,    $3) }

  | expr QUESTION expr COLON expr { Ternop($1, $3, $5) }

  | MINUS expr %prec NEG { Unop(Neg, $2) }
```

```
  | NOT expr         { Unop(Not, $2) }
  | expr PLUS PLUS %prec INCR   { Unop(Incr, $1) }
  | expr MINUS MINUS %prec DECR { Unop(Decr, $1) }

  | typ ID ASSIGN expr { DecAssign($1, $2, $4) }
  | ID ASSIGN expr   { Assign($1, $3) }
  | LPAREN formal_list RPAREN ASSIGN expr { Deconstruct($2, $5) }

  | expr ARROPEN expr ARRCLOSE { Access($1, $3) }
  | expr ARROPEN expr ARRCLOSE ASSIGN expr { AccessAssign($1, $3, $6) }

  | typ ID ASSIGN MATCH pattern { DecPatternMatch($1, $2, $5) }
  | ID ASSIGN MATCH pattern { PatternMatch($1, $4) }

  | ID LPAREN actuals_opt RPAREN { Call($1, $3) }
  | expr DOT ID LPAREN actuals_opt RPAREN { AttributeCall($1, $3, $5) }

  | LPAREN expr RPAREN { $2 }

pattern:
    c_pattern MATCH expr { MatchPattern($1, $3) }

c_pattern:
    expr COLON expr { [ConditionalPattern($1, $3)] }
  | expr COLON expr BAR c_pattern { ConditionalPattern($1, $3) :: $5 }

dict_exp:
    ARROPEN dict_elems ARRCLOSE { DictLit($2) }

dict_elems:
    dict_elem   { [$1] }
    | dict_elem COMMA dict_elems  { $1 :: $3 }

dict_elem:
    expr COLON expr { DictElem($1, $3) }

list_exp:
  ARROPEN list_elems ARRCLOSE { ListLit($2) }

list_elems:
  /* nothing */           { [] }
  | expr                  { [$1] }
  | expr COMMA list_elems { $1 :: $3 }

actuals_opt:
    /* nothing */ { [] }
  | actuals_list  { List.rev $1 }

actuals_list:
    expr                    { [$1] }
  | actuals_list COMMA expr { $3 :: $1 }
```

```
/* MARK: Stuff that we aren't using, but we might use (lol) */

/*
tuple_exp:
    LPAREN tuple_elems RPAREN   { TupleLit($2) }

tuple_elems:
    expr           { [$1] }
  | expr COMMA tuple_elems  { $1 :: $3 }
*/

/*
    | typ LPAREN formals_opt RPAREN ARROW stmt
      { { typ = $1;
      fname = "anon";
      formals = $4;
      body = [$7];
      autoreturn = true } }
    | typ LPAREN formals_opt RPAREN ARROW LBRACE stmt_list RBRACE
      { { typ = $1;
      fname = "anon";
      formals = $4;
      body = List.rev $8;
      autoreturn = false } }
 */

----------
----------
File: src/listcheck.ml
(*
Checking for array types
*)

open SAST

let eval_list = List.map (expr scope deepscope) l in

let rec check_types = function
        (t1, _) :: [] -> (t1, SListLiteral(eval_list))
      | ((t1,_) :: (t2,_) :: _) when n1 != n2 ->
      raise (Failure "List types are inconsistent")
      | _ :: t -> check_types t
    in check_types eval_list
----------
----------
File: src/semantdriver.ml
(*
Module to generate a SAST given a desugared AST
*)
open Ast
open Sast
```

```
open Boolcheck
open Rhandlhand
open Decs

let check (statements, functions) =

(* Gets symbol table of statement scope, and a list of symbol tables with each
function's scope *)
let symbol_table = get_decs (statements, functions) in
let global_scope = fst symbol_table in
let function_scopes = snd symbol_table in

(* Verifies that a function has a valid return statement *)
let rec check_return slist ret = match slist with
    Return _ :: _ -> if ret != Nah then true else raise(Failure "Function of type Nah
should not have a return statement")
  | s :: ss -> check_return ss ret
  | [] -> if ret = Nah then true else raise (Failure "Function has an empty body at
the highest level but returns (?)") in


let check_expr_scope scope = function
    DecAssign(ty, s, _) -> add_symbol_driver s ty scope
  | _ -> scope in

let rec check_stmt_scope scope = function
    Expr(e) -> check_expr_scope scope e
  | Dec(ty, s) -> add_symbol_driver s ty scope
  | While(p, _, _) -> check_expr_scope scope p
  | If(p, _, _) -> check_expr_scope scope p
  | PretendBlock(sl) -> List.fold_left (fun m f -> check_stmt_scope m f) scope sl
  | _ -> scope in

(* Bug fix for function return type mismatching *)
let return_func = function
    Function(e) -> e
  | e           -> e
  | _           -> raise (Failure "function return type is flawed") in

let type_check t1 t2 =
  let type1 = match t1 with
  Group([ta; tb]) -> (ta, tb)
| _ -> (t1, t1) in
  let type2 = match t2 with
  Group([tc; td]) -> (tc, td)
| _ -> (t2, t2) in
(fst type1 = fst type2) && (snd type1 = snd type2)

in

(* Driver for semantically checking expressions *)
let rec expr scope deepscope e = match e with
```

```
    IntegerLiteral l -> (Int, SIntegerLiteral l)
  | CharacterLiteral l -> (Char, SCharacterLiteral l)
  | BoolLit l -> (Bool, SBoolLiteral l)
  | FloatLiteral l -> (Float, SFloatLiteral l)
  | StringLiteral l -> (String, SStringLiteral l)
  | Noexpr -> (Nah, SNoexpr)
  | ListLit l -> let eval_list = List.map (expr scope deepscope) l in
      let rec check_types = function
          [] -> (Nah, SDictLiteral([]))
        | (t1, _) :: [] -> (Array(t1), SListLiteral(eval_list))
        |   ((t1,_) :: (t2,_) :: _) when t1 != t2 ->
          raise (Failure ("Error: list types " ^ string_of_typ t1 ^ " and " ^
string_of_typ t2 ^ " are inconsistent"))
        | _ :: t -> check_types t
      in check_types eval_list
  | DictElem(l, s) -> let (t1, e1) = expr scope deepscope l in
      let (t2, e2) = expr scope deepscope s in
      (Group([t1; t2]), SDictElem((t1, e1), (t2, e2)))
  | DictLit l -> let eval_list = List.map (expr scope deepscope) l in
      let rec check_types = function
        | (Group([t1; t2]), _) :: [] -> (Dictionary(t1, t2), SDictLiteral(eval_list))
        |   ((Group([t1; t2]), _) :: (Group([t3; t4]), _) :: _) when not ((type_check
t1 t3)) || not ((type_check t2 t4)) (*t1 != t3 || t2 != t4 *)->
           raise (Failure (string_of_typ t1 ^ string_of_typ t2 ^ string_of_typ t3 ^
string_of_typ t4))
        | _ :: t -> check_types t
      in check_types eval_list
  | Id l -> (toi scope l, SId l)
  | Binop(e1, op, e2) as e ->
      let (t1, e1') = expr scope deepscope e1
      and (t2, e2') = expr scope deepscope e2 in
      (* All binary operators require operands of the same type *)
      let same = t1 = t2 in
      (* Determine expression type based on operator and operand types *)
      let ty = match op with
          Add | Sub | Mult | Div | Mod when same && t1 = Int   -> Int
        | Add | Sub | Mult | Div | Mod when same && t1 = Float -> Float
        | Equal | Neq                  when same               -> Bool
        | Less | Leq | Greater | Geq   when same && (t1 = Int || t1 = Float) -> Bool
        | And | Or | Has               when same && t1 = Bool -> Bool
        | _ -> raise (Failure ("illegal binary operator " ^
                      string_of_typ t1 ^ " " ^ string_of_op op ^ " " ^
                      string_of_typ t2 ^ " in " ^ string_of_expr e))
      in (ty, SBinop((t1, e1'), op, (t2, e2')))
  | Unop(uop, e) as ex ->
      let (t, e') = expr scope deepscope e in
      let ty = match uop with
          Neg | Incr | Decr when t = Int || t = Float -> t
        | Not when t = Bool -> Bool
        | _ -> raise (Failure ("illegal unary operator " ^
                               string_of_uop uop ^ string_of_typ t ^
                               " in " ^ string_of_expr ex))
```

```
      in (ty, SUnop(uop, (t, e')))
  | Assign(s, e) ->
      let lt = toi scope s
      and (rt, e') = expr scope deepscope e in
      (check_assign lt rt, SAssign(s, (rt, e')))
  | Deconstruct(l, e) -> (****** Work in progress, discrepancy between group and list
literals ******)
      let (e_typ, _) = expr scope deepscope e in
      let _ = match e_typ with
          Group(typs) -> typs
        | _ -> raise (Failure ("Error: deconstruct requires a Group, but was given " ^
string_of_typ e_typ ^ " " ^ string_of_expr e))
      in (Int, SDeconstruct(l, expr scope deepscope e))
  | OpAssign(s, op, e) -> let (t, e1) = expr scope deepscope e in
      if t = (toi scope s) then (t, SOpAssign(s, op, (t, e1))) else raise (Failure
"types not the same")
  | DecAssign(ty, l, expr1) -> (match ty, l, expr1 with
        (Array(t), n, ListLit([])) -> (Array(t), SDecAssign(Array(t), n, (Array(t),
SListLiteral([]))))
      | (Dictionary(t1, t2), n, DictLit([])) -> (Dictionary(t1, t2),
SDecAssign(Dictionary(t1, t2), n, (Dictionary(t1, t2), SDictLiteral([]))))
      | _ -> check_decassign ty l (expr scope deepscope expr1) )
  | Access(e1, e2) ->
      let (t1, e1') = expr scope deepscope e1
      and (t2, e2') = expr scope deepscope e2 in
      (match t1 with
          Array(t) when t2 = Int -> (t, SAccess((t1, e1'), (t2, e2')))
        | Array(_) -> raise (Failure ("Error: integer required for array access, given
type " ^ string_of_typ t2))
        | Dictionary((key_t, value_t)) when t2 = key_t -> (value_t, SAccess((t1, e1'),
(t2, e2')))
        | Dictionary((key_t, _)) -> raise (Failure ("Error: " ^ string_of_typ key_t ^
" required for dictionary access, given type " ^ string_of_typ t2))
        | _ -> raise (Failure ("Error: access not invalid for type " ^ string_of_typ
t1)))
  | AccessAssign(e1, e2, e3) ->
      let (t1, e1') = expr scope deepscope e1
      and (t2, e2') = expr scope deepscope e2
      and (t3, e3') = expr scope deepscope e3 in
      (match t1 with
          Dictionary((key_t, val_t)) when t3 = val_t ->
            if t2 = key_t then (t3, SAccessAssign((t1, e1'), (t2, e2'), (t3, e3')))
            else raise (Failure ("Error: key type " ^ string_of_typ key_t ^ " expected
for Dictionary access, but " ^
                                string_of_typ t2 ^ " given in expression " ^
string_of_expr e2))
        | Dictionary((_, val_t)) -> raise (Failure ("Error: value type " ^
string_of_typ t3 ^ " cannot be included in Dictionary " ^
                                           string_of_expr e1 ^ " with value type " ^
string_of_typ val_t))
        | Array(t) when t = t3 ->
            if t2 = Int then (t3, SAccessAssign((t1, e1'), (t2, e2'), (t3, e3')))
```

```
            else raise (Failure ("Error: integer expected for Array access, but " ^
string_of_typ t2 ^
                                " given in expression " ^ string_of_expr e2))
        | Array(t) -> raise (Failure ("Error: type " ^ string_of_typ t3 ^ " cannot be
included in Array " ^ string_of_expr e1 ^
                                     " with type " ^ string_of_typ t))
        | _ -> raise (Failure ("Error: expression " ^ string_of_expr e1 ^ " has type "
^ string_of_typ t1 ^
                                ", expected type Array")))
  | Call(fname, args) ->
      let eval_list = List.map (expr scope deepscope) args in
      let key_func = key_string fname eval_list in
      let fd = StringMap.find key_func function_scopes in
      let param_length = StringMap.cardinal fd.formals.variables in
      if List.length args != param_length then
      raise (Failure ("expecting " ^ string_of_int param_length ^
                " arguments in function call" ))
      else let check_call (_, ft) e =
      let (et, e') = expr scope deepscope e in
      (check_assign ft et, e')
      in
      let args' = List.map2 check_call (StringMap.bindings fd.formals.variables) args
      in (return_func fd.ret_typ, SCall(fname, args'))
  | AttributeCall(e, fname, args) ->
      let eval_list = List.map (expr scope deepscope) args in
      let key_func = key_string fname eval_list in
      let fd = StringMap.find key_func function_scopes in
      let param_length = StringMap.cardinal fd.formals.variables in
      if List.length args + 1 != param_length then raise (Failure ("expecting " ^
string_of_int param_length ^ " arguments in function call"))
      else let check_call (_, ft) e =
      let (et, e') = expr scope deepscope e in
      (check_assign ft et, e') in
      let args' = List.map2 check_call (StringMap.bindings fd.formals.variables) args
      in (return_func fd.ret_typ, SAttributeCall(expr scope deepscope e, fname,
args'))
  (*
  | Ternop(e1, e2, e3) ->
      let (e1t, e1') = expr scope deepscope e1 in
      if e1t != Bool then raise (Failure "Error: expected bool in first expression of
ternary operator") else
      let (e2t, e2') = expr scope deepscope e2
      and (e3t, e3') = expr scope deepscope e3 in
      if e2t != e3t then raise (Failure ("Error: ternary operator types " ^
string_of_typ e2t ^ " and " ^ string_of_typ e3t ^ " do not match"))
      else (e2t, STernop((e1t, e1'), (e2t, e2'), (e3t, e3'))) *)
  | _  -> raise (Failure "expression is not an expression")
in

(* Driver for semantically checking statements *)
let rec check_stmt scope inloop s =
  let new_scope = {
```

```
    variables = StringMap.empty;
    parent = Some(scope);
  } in match s with
    Expr e -> SExpr (expr scope inloop e)
  | Skip e -> if inloop then SSkip (expr scope inloop e) else raise (Failure "skip not
in a loop")
  | Abort e -> if inloop then SAbort (expr scope inloop e) else raise (Failure "abort
not in a loop")
  | Panic e -> SPanic (expr scope inloop e)
  | If(p, b1, b2) as i ->
      let scope = get_expr_decs scope p in
      let pred = check_bool (expr scope inloop p)
      and t = check_stmt scope inloop b1
      and f = check_stmt scope inloop b2 in SIf(pred, t, f)
  | While(p, s, inc) ->
      let scope = get_expr_decs scope p in
      let pred = check_bool (expr scope inloop p)
      and loop = check_stmt scope true s in SWhile(pred, loop, check_stmt scope inloop
inc)
  | For(e1, e2, e3, s) -> raise (Failure ("Error: nested for loops currently broken"))
  | Return _ -> raise (Failure "return outside a function")
  | Block sl ->
      let rec check_stmt_list blockscope = function
          [Return _ as s] -> [check_stmt blockscope inloop s]
        | Return _ :: _   -> raise (Failure "nothing may follow a return")
        (*| Block s :: ss   -> check_stmt_list blockscope (s @ ss)  Flatten blocks
        | PretendBlock sl :: ss -> check_stmt_list blockscope (sl @ ss)  Flatten
blocks *)
        | s :: ss         -> check_stmt blockscope inloop s :: check_stmt_list
blockscope ss
        | [] -> []
      in SBlock(check_stmt_list (List.fold_left (fun m f -> check_stmt_scope m f)
new_scope sl) sl)
  | PretendBlock sl ->
        SBlock(List.map (check_stmt scope inloop) sl)
  | Dec(ty, l) -> SDec(ty, l)
  | _ as s -> raise (Failure ("statement " ^ string_of_stmt s ^ " is not a
statement"))
in

(* Check statements within functions *)
let rec check_stmt_func scope inloop ret =
  let new_scope = {
    variables = StringMap.empty;
    parent = Some(scope);
  } in function
    Expr e -> SExpr (expr scope inloop e)
  | Skip e -> if inloop then SSkip (expr scope inloop e) else raise (Failure "skip not
in a loop")
  | Abort e -> if inloop then SAbort (expr scope inloop e) else raise (Failure "abort
not in a loop")
  | Panic e -> SPanic (expr scope inloop e)
```

```
  | If(p, b1, b2) -> SIf(check_bool (expr scope inloop p), check_stmt_func scope
inloop ret b1, check_stmt_func scope inloop ret b2)
  | While(p, s, inc) -> SWhile(check_bool (expr scope inloop p), check_stmt_func
new_scope true ret s, check_stmt scope inloop inc)
  | Return e -> let (t, e') = expr scope inloop e in
      if t = ret then SReturn (t, e')
      else raise (
      Failure ("return gives " ^ string_of_typ t ^ " expected " ^
        string_of_typ ret ^ " in " ^ string_of_expr e))
  | Block sl ->
      let rec check_stmt_list blockscope = function
          [Return _ as s] -> [check_stmt_func blockscope inloop ret s]
        | Return _ :: _   -> raise (Failure "nothing may follow a return")
        (*| Block sl :: ss  -> check_stmt_list blockscope (sl @ ss)  Flatten blocks
        | PretendBlock sl :: ss -> check_stmt_list blockscope (sl @ ss)  Flatten
blocks *)
        | s :: ss         -> check_stmt_func blockscope inloop ret s ::
check_stmt_list blockscope ss
        | []              -> []
      in SBlock(check_stmt_list (List.fold_left (fun m f -> check_stmt_scope m f)
new_scope sl) sl)
  | PretendBlock sl ->
      SBlock(List.map (check_stmt_func scope inloop ret) sl)
  | Dec(ty, l) -> SDec(ty, l)
  | _ as s -> raise (Failure ("statement " ^ string_of_stmt s ^ " is not a
statement"))
in

(* Check function declarations *)
let check_function (fd : func_decl) =
  if check_return fd.body (return_func fd.typ) then
    let key_func = key_string fd.fname fd.formals in
      let current_function = StringMap.find key_func function_scopes in
      { styp = return_func fd.typ;
        sfname = fd.fname;
        sformals = fd.formals;
        sbody = match check_stmt_func current_function.locals false (return_func
fd.typ) (Block fd.body) with
              SBlock(sl) -> sl
          | _ -> raise (Failure ("internal error: block didn't become a block?"))
    }
    else raise (Failure "there is not return statement at the highest level of the
function")
in

(* Collect the SAST of semantically-check statements and functions *)
let sstatements = List.map (check_stmt global_scope false) statements in
let sfuncs = List.map check_function functions in

(* Aggregate statements into an implicit main function if one isn't already defined *)
let rec has_main sfuncs = match sfuncs with
    [] -> false
```

```
  | sfd :: _ when sfd.sfname = "main" && sfd.styp = Int -> true
  | sfd :: _ when sfd.sfname = "main" -> raise (Failure ("Error: function main must
return type Int, not type " ^ string_of_typ sfd.styp))
  | _ :: tail -> has_main tail in

let updated_sfuncs = if has_main sfuncs then sfuncs else
  { styp = Int;
    sfname = "main";
    sformals = [];
    sbody = List.rev sstatements;
  } :: sfuncs in

(sstatements, updated_sfuncs)

----------
----------
File: demo/factorial.vp
int func factorial(int target) {
    if(target == 1) {
        print(target);
        return target;
    }

    int res = target * factorial(target-1);
    print(res);
    return res;
}

print("Factorial of 10:");
factorial(10);

----------
----------
File: demo/calc.vp
int func calc(int a, int b, char op) {
    int res = ??
        op == '+' : (a + b)
      | op == '-' : (a - b)
      | op == '*' : (a * b)
      | op == '/' : (a / b)
    ?? a ;

    return res;
}

int result = 0;

print("10 + 20 =");
result = calc(10, 20, '+');
print(result);

print("10 - 20 =");
```

```
result = calc(10, 20, '-');
print(result);

print("10 * 20 =");
result = calc(10, 20, '*');
print(result);

print("10 / 20 =");
result = calc(10, 20, '/');
print(result);

print("10 ^ 20 = (unknown operator returns first value)");
result = calc(10, 20, '^');
print(result);

----------
----------
File: demo/bubble.vp
int[] func bubble(int[] arr) {
    for(int i = 0; i < len(arr); i++) {
        int limit = len(arr) - i - 1;
        for(int j = 0; j < limit; j++) {
            if(arr[j] > arr[j+1]) {
                int temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
                print("Step:");
                print(arr);
            }
        }
    }

    return arr;
}


int[] target = [4, 5, 10, 1, 5, 3];
print("Initial:");
print(target);

int[] sort = bubble(target);
print("Final:");
print(sort);
----------
----------
File: demo/binse.vp
int func binary_search(int []arr, int x) {
    int low = 0;
    int high = len(arr);
    int mid = 0;

    while(low <= high) {
```

```
        mid = (high + low)/2;
        if(arr[mid] < x) {
            low = mid + 1;
        } else if(arr[mid] > x) {
            high = mid - 1;
        }
        else {
            return mid;
        }
    }
    return -1;
}

nah func print_result(int loc) {
    if (loc != -1) {
        print("Found! Location:");
        print(loc);
    } else {
        print("Not found.");
    }
}

int target = 4;
int []present = [1,2,3,4,5];
int []absent = [1,2,3,5];

print("Looking for 4 in:");
print(present);
print_result(binary_search(present, 4));

print();

print("Looking for 4 in:");
print(absent);
print_result(binary_search(absent, 4));

----------
----------
File: demo/fizzbuzz.vp
bool func isDivisible(int x, int div) => (x % div == 0);

for(int i = 1; i <= 100; i+=1) {
    if(isDivisible(i, 15)) {
        print("fizzbuzz");
    }
    else if(isDivisible(i, 3)) {
        print("fizz");
    } else if(isDivisible(i, 5)) {
        print("buzz");
    } else {
        print(i);
    }
```

```
}

----------
----------
File: demo/wordcount.vp
[string: int] func count_words(string[] str) {
    [string: int] word_counts = [];
    for(string x in str) {
        if(word_counts.contains(x) == 1) {
            word_counts.add(x, word_counts[x] + 1);
        } else {
            word_counts.add(x, 1);
        }
    }
    return word_counts;
}

string [] test = ["Hello", "world", "am", "Viper", "Hello", "world", "am", "world",
"Viper", "Viper", "Viper", "ratghav"];

[string: int] counts = count_words(test);
string[] unique_keys = counts.keys();

print("Word list:");
print(test);
print();

print("Counts of keys:");
for(string key in unique_keys) {
    print("Key:");
    print(key);
    print("Count:");
    print(counts[key]);
    print();
}

----------
----------
File: test/tests/condition.vp
print(5 > 0);
print(0 < 5);
print(5 >= 5);
print(5 <= 5);
print(true && true);
print(true || false);
print(false && true);
print(false && false || true);


if (true) {
    print(true);
} else {
```

```
    print(false);
}

if (false) {
    print(false);
} else {
    print(true);
}
----------
----------
File: test/tests/contains.vp
int[] ints = [1];
print(ints.contains(1));

string[] names = ["ratghav"];
print(names.contains("ratghav"));

float[] floats = [1.2];
print(floats.contains(1.2));

char[] chars = ['A'];
print(chars.contains('A'));
----------
----------
File: test/tests/skip_abort.vp
int[] list = [1,2,3,4,5];
for (int i in list){
    if (i == 3){ skip; }
    else if (i == 4){ abort; }
}

----------
----------
File: test/tests/matching.vp
int func positive(int x) {
    int test = ??
        x < 0 : 0
      | x > 0: 10
    ?? 0;
    return test;
}

print(positive(-3));
print(positive(10));

----------
----------
File: test/tests/modulo.vp
int ten = 10;
int twenty = 20;
int one = 1;
int two = 2;
```

```
/* 10 */
int a = ten % twenty;
print(a);

/* 0 */
int b = twenty % ten;
print(b);

/* 1 */
int c = one % two;
print(c);

/* 0 */
int d = two % one;
print(d);
----------
----------
File: test/tests/function_call.vp
int func test(int x) {
    return x * x;
}

int func sumOfList(int[] l) {
    int sum = 0;
    int i;
    for (i = 0; i < len(l); i+=1) {
        sum += l[i];
    }
    return sum;
}

bool func isIndexOfGreatest(int index, int[] l) {
    if (len(l) == 0) return false;
    int i;
    for (i = 0; i < len(l); i+=1) {
        if (l[i] > l[index]) {
            return false;
        }
    }
    return true;
}

print(test(10));

int[] fib = [1, 1, 2, 3, 5, 8, 13, 21];
print(sumOfList(fib));

int[] t = [1, 2, 3, 4, 5, 6, 7, 8];
print(isIndexOfGreatest(7, t));
print(isIndexOfGreatest(2, t));
```

```
----------
----------
File: test/tests/nestedloop.vp.out
0
1
2
3
4
5
6
7
8
9
0
1
2
3
4
5
6
7
8
9
0
1
2
3
4
5
6
7
8
9
0
1
2
3
4
5
6
7
8
9
0
1
2
3
4
5
6
7
8
```

```
----------
----------
File: test/tests/printchar-seq.vp
print('r');
print('a');
print('t');

----------
----------
File: test/tests/matching.vp.out
0
10
----------
----------
File: test/tests/printfloat.vp.out
3
38264.1
0
-99.9
----------
----------
File: test/tests/modulo.vp.out
10
0
1
0

----------
----------
File: test/tests/condition.vp.out
1
1
1
1
1
1
0
1
1
1
----------
----------
File: test/tests/indexed_function.vp.out
4
3
----------
----------
File: test/tests/for.vp.out
Outer:
0
Inner:
0
```

```
Inner:
1
Inner:
2
Inner:
3
Inner:
4
Inner:
5
Inner:
6
Inner:
7
Inner:
8
Inner:
9
Outer:
1
Inner:
1
Inner:
2
Inner:
3
Inner:
4
Inner:
5
Inner:
6
Inner:
7
Inner:
8
Inner:
9
Outer:
2
Inner:
2
Inner:
3
Inner:
4
Inner:
5
Inner:
6
Inner:
7
```

```
Inner:
8
Inner:
9
Outer:
3
Inner:
3
Inner:
4
Inner:
5
Inner:
6
Inner:
7
Inner:
8
Inner:
9
Outer:
4
Inner:
4
Inner:
5
Inner:
6
Inner:
7
Inner:
8
Inner:
9
----------
----------
File: test/tests/list-append-access.vp
string[] names = ["trey"];
print(names[0]);

names.append("ratghav");
print(names[1]);

int[] nums = [1];
print(nums[0]);

nums.append(2);
print(nums[1]);

char[] chars = ['A'];
print(chars[0]);
```

```
chars.append('B');
print(chars[1]);

float[] nums2 = [1.0];
print(nums2[0]);

nums2.append(2.0);
print(nums2[1]);
----------
----------
File: test/tests/incr-decr.vp.out
1
2
1
----------
----------
File: test/tests/skip.vp.out
0
1
2
3
4
bad
6
7
8
9

0
skipping
1
did not skip
2
did not skip
3
did not skip
4
skipping
5
did not skip
6
did not skip
7
did not skip
8
skipping
9
did not skip
10
did not skip
11
did not skip
```

```
12
skipping
13
did not skip
14
did not skip
15
did not skip
16
skipping
17
did not skip
18
did not skip
19
did not skip
20
skipping
21
did not skip
22
did not skip
23
did not skip
24
skipping
25
did not skip
26
did not skip
27
did not skip
28
skipping
29
did not skip
30
did not skip
31
did not skip
32
skipping
33
did not skip
34
did not skip
35
did not skip
36
skipping
37
did not skip
```

```
38
did not skip
39
did not skip
40
skipping
41
did not skip
42
did not skip
43
did not skip
44
skipping
45
did not skip
46
did not skip
47
did not skip
48
skipping
49
did not skip
50
did not skip
51
did not skip
52
did not skip
53
did not skip
54
did not skip
55
did not skip
56
did not skip
57
did not skip
58
did not skip
59
did not skip
60
did not skip
61
did not skip
62
did not skip
63
did not skip
```

```
64
did not skip
65
did not skip
66
did not skip
67
did not skip
68
did not skip
69
did not skip
70
did not skip
71
did not skip
72
did not skip
73
did not skip
74
did not skip
75
did not skip
76
did not skip
77
did not skip
78
did not skip
79
did not skip
80
did not skip
81
did not skip
82
did not skip
83
did not skip
84
did not skip
85
did not skip
86
did not skip
87
did not skip
88
did not skip
89
did not skip
```

```
90
did not skip
91
did not skip
92
did not skip
93
did not skip
94
did not skip
95
did not skip
96
did not skip
97
did not skip
98
did not skip
99
did not skip
----------
----------
File: test/tests/helloworld.vp
print("Hello World!");
----------
----------
File: test/tests/for.vp
int j;
for (int i = 0; i < 5; i+=1) {
    print("Outer:");
    print(i);
    for (j = i; j < 10; j+=1) {
        print("Inner:");
        print(j);
    }
}

/*
for(int f in arr) {
    j = j + f;
}

int g;

for(g in arr) {
    j = j + g;
}

for((int, bool) tup in list) {
    if (tup[1] == true) {
        j += tup[0];
    }
```

```
}

for((int iter, int index) in list) {
    j += index;
    j += iter;
}
*/

----------
----------
File: test/tests/pow.vp.out
4
16
767376
----------
----------
File: test/tests/skip_abort.vp.out

----------
----------
File: test/tests/incr-decr.vp
int a = 1;
print(a);

a++;
print(a);

a--;
print(a);

----------
----------
File: test/tests/helloworld.vp.out
Hello World!

----------
----------
File: test/tests/cast.vp.out
9
99
888
false
1
0
8
d
h
----------
----------
File: test/tests/printbools.vp.out
1
0
```

```
----------
----------
File: test/tests/arrowfunction_call.vp
int func sum (int c, int d) => c + d;
bool func between (int i, int lowerb, int upperb) => (i > lowerb && i < upperb);


print(sum(10, 40));
print(between(5, 1, 9));

----------
----------
File: test/tests/arrays.vp.out
1
2
3
4
5
a
b
c
d
----------
----------
File: test/tests/strings.vp
string test = "bla";
print(test);
string test2 = "hi";
string test3 = test2;
print(test3);

test = "hello";
print(test);


print("@!#$*(YW(EF*H))");
print("\rwow\twow\nwow\rwow\twow\nwow");
print("11234569097654");
----------
----------
File: test/tests/index_access.vp
int[] d = [1,2,3,4];
char[] c = ['a'];

print(d[2]);
print(c[0]);

/* throws error
print(c[-1]);
*/

----------
```

```
----------
File: test/tests/attribute_call.vp.out
testing with no parameters:
42
testing with many parameters:
0
----------
----------
File: test/tests/printbools.vp
print(true);
print(false);

----------
----------
File: test/tests/variable_decl.vp.out
10
20
10
20
a
1
0
1.1
2.2
3.3

----------
----------
File: test/tests/cast.vp
print(toInt(9.34));
print(toInt('c'));
print(toString(888));
print(toString(false));
print(toBool(400));
print(toBool(""));
print(toFloat(8));
print(toChar(100));
print(toChar("hi"));

----------
----------
File: test/tests/list-append-access.vp.out
trey
ratghav
1
2
A
B
1
2
----------
----------
```

```
File: test/tests/pow.vp
float a = pow2(2.0);
print(a);

float b = pow2(a);

print(b);

float c = pow2(876.0);
print(c);



----------
----------
File: test/tests/shorthands.vp
int a = 10;
int b = 20;
int c = 0;

/* 0 */
print(c);

/* 10 */
c += a;
print(c);

/* -10 */
c -= b;
print(c);

/* 190 */
c += a * b;
print(c);

/* -10 */
c -= a * b;
print(c);

/* 100 */
c *= c;
print(c);
----------
----------
File: test/tests/shorthands.vp.out
0
10
-10
190
-10
100
```

```
----------
----------
File: test/tests/arrowfunction_call.vp.out
50
1
----------
----------
File: test/tests/attribute_call.vp
int func tester(int x) {
  return 42;
}

bool func least(int a, int b, int d, int e, int f) {
  return a > b;
}

int z = 42.tester();
print("testing with no parameters:");
print(z);

print("testing with many parameters:");
int isLeast = 5;
print(isLeast.least(6, 3, -9, 88));
----------
----------
File: test/tests/arrays.vp

int[] s = [1, 2, 3, 4, 5];
char[] c = ['a', 'b', 'c', 'd'];
int[] e;

print(s[0]);
print(s[1]);
print(s[2]);
print(s[3]);
print(s[4]);

print(c[0]);
print(c[1]);
print(c[2]);
print(c[3]);



----------
----------
File: test/tests/skip.vp
for (int i = 0; i < 10; i += 1){
    if (i == 5) {
        int x = 444;
        print("bad");
        skip;
```

```
    }
    print(i);
}
print();
for (int i = 0; i < 100; i+= 1) {
    print(i);
    if (i < 50) {
        if (i % 4 == 0){
            print("skipping");
            skip;
        }
    }
    print("did not skip");
}

----------
----------
File: test/tests/printnum-seq.vp
print(3);
print(4);
print(5);

----------
----------
File: test/tests/printfloat.vp
print(3.0);
print(38264.1);
print(0.0);
print(-99.9);

----------
----------
File: test/tests/indexed_function.vp
int[] func returnArray() {
    return [1,2,3,4];
}

print(returnArray()[3]);

[char: int] func returnDict() {
    return ['A': 1, 'B': 2, 'C': 3];
}

print(returnDict()['C']);
----------
----------
File: test/tests/abort.vp.out
testing skip and abort together:
0
1
2
3
```

```
5


---------

testing abort in a while loop:
0
1
2
3

testing skip and abort with nested loops:
aborted inner for loop
1
0
aborted inner for loop
2
0
2
1
aborted inner for loop
4
0
4
1
4
2
4
3
aborted inner for loop
5
0
5
1
5
2
5
3
5
4
aborted inner for loop
6
0
6
1
6
2
6
3
6
4
6
```

```
5
aborted inner for loop
7
0
7
1
7
2
7
3
7
4
7
5
7
6
aborted inner for loop
8
0
8
1
8
2
8
3
8
4
8
5
8
6
8
7
aborted inner for loop
9
0
9
1
9
2
9
3
9
4
9
5
9
6
9
7
9
8
```

```
aborted inner for loop
10
0
10
1
10
2
10
3
10
4
10
5
10
6
10
7
10
8
10
9
aborted inner for loop
terminated outer while loop
----------
----------
File: test/tests/printnum-seq.vp.out
3
4
5

----------
----------
File: test/tests/abort.vp
print("testing skip and abort together:");
for (int i = 0; i < 10; i += 1) {
    if (i == 4) {
        skip;
    }
    if (i > 5) {
        abort;
    }
    print(i);
}

print();
print();
print("---------");
print();
print("testing abort in a while loop:");
int a = -1;
while (a += 1 < 80) {
    if (a > 3) {
```

```
        abort;
    }
    print(a);
}

/* Throws error
if (a > 9) {
    abort;
} */

print();
print("testing skip and abort with nested loops:");
int x = -1;
while (x < 10) {
    x += 1;
    if (x == 3) {
        skip;
    }
    int i = 0;
    for (i; i < 10; i += 1) {
        if (x > i) {
            print(x);
            print(i);
        } else {
            abort;
        }
    }
    print("aborted inner for loop");
}
print("terminated outer while loop");
```

```
----------
----------
File: test/tests/variable_decl.vp
int a = 10;
print(a);
int b = 20;
print(b);
int c;
c = 10;
print(c);
int d = c + 10;
print(d);

char e = 'a';
print(e);

bool x = true;
print(x);
```

```
bool y = false;
print(y);

float xx = 1.1;
print(xx);
float zz = 2.2;
print(zz);
float aa = xx + zz;
print(aa);

----------
----------
File: test/tests/printchar-seq.vp.out
r
a
t

----------
----------
File: test/tests/index_access.vp.out
3
a

----------
----------
File: test/tests/strings.vp.out
bla
hi
hello
@!#$*(YW(EF*H))
\rwow\twow\nwow\rwow\twow\nwow
11234569097654
----------
----------
File: test/tests/function_call.vp.out
100
54
1
0

----------
----------
File: test/tests/nestedloop.vp
for(int i = 0; i < 10; i++) {
    for(int j = 0; j < 10; j++) {
        print(j);
    }
}

----------
----------
File: test/tests/contains.vp.out
```

```
1
1
1
1
----------
----------
File: test/tests/ternary.vp.out

----------
----------
File: test/tests/condition.vp
print(5 > 0);
print(0 < 5);
print(5 >= 5);
print(5 <= 5);
print(true && true);
print(true || false);
print(false && true);
print(false && false || true);


if (true) {
    print(true);
} else {
    print(false);
}

if (false) {
    print(false);
} else {
    print(true);
}
----------
----------
File: test/tests/contains.vp
int[] ints = [1];
print(ints.contains(1));

string[] names = ["ratghav"];
print(names.contains("ratghav"));

float[] floats = [1.2];
print(floats.contains(1.2));

char[] chars = ['A'];
print(chars.contains('A'));
----------
----------
File: test/tests/skip_abort.vp
int[] list = [1,2,3,4,5];
for (int i in list){
    if (i == 3){ skip; }
```

```
    else if (i == 4){ abort; }
}

----------
----------
File: test/tests/matching.vp
int func positive(int x) {
    int test = ??
        x < 0 : 0
      | x > 0: 10
    ?? 0;
    return test;
}

print(positive(-3));
print(positive(10));

----------
----------
File: test/tests/modulo.vp
int ten = 10;
int twenty = 20;
int one = 1;
int two = 2;


/* 10 */
int a = ten % twenty;
print(a);

/* 0 */
int b = twenty % ten;
print(b);

/* 1 */
int c = one % two;
print(c);

/* 0 */
int d = two % one;
print(d);
----------
----------
File: test/tests/function_call.vp
int func test(int x) {
    return x * x;
}

int func sumOfList(int[] l) {
    int sum = 0;
    int i;
    for (i = 0; i < len(l); i+=1) {
```

```
        sum += l[i];
    }
    return sum;
}

bool func isIndexOfGreatest(int index, int[] l) {
    if (len(l) == 0) return false;
    int i;
    for (i = 0; i < len(l); i+=1) {
        if (l[i] > l[index]) {
            return false;
        }
    }
    return true;
}

print(test(10));

int[] fib = [1, 1, 2, 3, 5, 8, 13, 21];
print(sumOfList(fib));

int[] t = [1, 2, 3, 4, 5, 6, 7, 8];
print(isIndexOfGreatest(7, t));
print(isIndexOfGreatest(2, t));
----------
----------
File: test/tests/nestedloop.vp.out
0
1
2
3
4
5
6
7
8
9
0
1
2
3
4
5
6
7
8
9
0
1
2
3
4
```

```
7
8
9
0
1
2
3
4
5
6
7
8
9
0
1
2
3
4
5
6
7
8
9

----------
----------
File: test/tests/printchar-seq.vp
print('r');
print('a');
print('t');

----------
----------
File: test/tests/matching.vp.out
0
10
----------
----------
File: test/tests/printfloat.vp.out
3
38264.1
0
-99.9
----------
----------
File: test/tests/modulo.vp.out
10
0
1
0

----------
```

```
----------
File: test/tests/condition.vp.out
1
1
1
1
1
1
0
1
1
1
----------
----------
File: test/tests/indexed_function.vp.out
4
3
----------
----------
File: test/tests/for.vp.out
Outer:
0
Inner:
0
Inner:
1
Inner:
2
Inner:
3
Inner:
4
Inner:
5
Inner:
6
Inner:
7
Inner:
8
Inner:
9
Outer:
1
Inner:
1
Inner:
2
Inner:
3
Inner:
4
```

```
Inner:
5
Inner:
6
Inner:
7
Inner:
8
Inner:
9
Outer:
2
Inner:
2
Inner:
3
Inner:
4
Inner:
5
Inner:
6
Inner:
7
Inner:
8
Inner:
9
Outer:
3
Inner:
3
Inner:
4
Inner:
5
Inner:
6
Inner:
7
Inner:
8
Inner:
9
Outer:
4
Inner:
4
Inner:
5
Inner:
6
```

```
Inner:
7
Inner:
8
Inner:
9
----------
----------
File: test/tests/list-append-access.vp
string[] names = ["trey"];
print(names[0]);

names.append("ratghav");
print(names[1]);

int[] nums = [1];
print(nums[0]);

nums.append(2);
print(nums[1]);

char[] chars = ['A'];
print(chars[0]);

chars.append('B');
print(chars[1]);

float[] nums2 = [1.0];
print(nums2[0]);

nums2.append(2.0);
print(nums2[1]);
----------
----------
File: test/tests/incr-decr.vp.out
1
2
1
----------
----------
File: test/tests/skip.vp.out
0
1
2
3
4
bad
6
7
8
9
```

```
0
skipping
1
did not skip
2
did not skip
3
did not skip
4
skipping
5
did not skip
6
did not skip
7
did not skip
8
skipping
9
did not skip
10
did not skip
11
did not skip
12
skipping
13
did not skip
14
did not skip
15
did not skip
16
skipping
17
did not skip
18
did not skip
19
did not skip
20
skipping
21
did not skip
22
did not skip
23
did not skip
24
skipping
25
did not skip
```

```
26
did not skip
27
did not skip
28
skipping
29
did not skip
30
did not skip
31
did not skip
32
skipping
33
did not skip
34
did not skip
35
did not skip
36
skipping
37
did not skip
38
did not skip
39
did not skip
40
skipping
41
did not skip
42
did not skip
43
did not skip
44
skipping
45
did not skip
46
did not skip
47
did not skip
48
skipping
49
did not skip
50
did not skip
51
did not skip
```

```
52
did not skip
53
did not skip
54
did not skip
55
did not skip
56
did not skip
57
did not skip
58
did not skip
59
did not skip
60
did not skip
61
did not skip
62
did not skip
63
did not skip
64
did not skip
65
did not skip
66
did not skip
67
did not skip
68
did not skip
69
did not skip
70
did not skip
71
did not skip
72
did not skip
73
did not skip
74
did not skip
75
did not skip
76
did not skip
77
did not skip
```

```
78
did not skip
79
did not skip
80
did not skip
81
did not skip
82
did not skip
83
did not skip
84
did not skip
85
did not skip
86
did not skip
87
did not skip
88
did not skip
89
did not skip
90
did not skip
91
did not skip
92
did not skip
93
did not skip
94
did not skip
95
did not skip
96
did not skip
97
did not skip
98
did not skip
99
did not skip
----------
----------
File: test/tests/helloworld.vp
print("Hello World!");
----------
----------
File: test/tests/for.vp
int j;
```

```
for (int i = 0; i < 5; i+=1) {
    print("Outer:");
    print(i);
    for (j = i; j < 10; j+=1) {
        print("Inner:");
        print(j);
    }
}

/*
for(int f in arr) {
    j = j + f;
}

int g;

for(g in arr) {
    j = j + g;
}

for((int, bool) tup in list) {
    if (tup[1] == true) {
        j += tup[0];
    }
}

for((int iter, int index) in list) {
    j += index;
    j += iter;
}
*/

----------
----------
File: test/tests/pow.vp.out
4
16
767376
----------
----------
File: test/tests/skip_abort.vp.out

----------
----------
File: test/tests/incr-decr.vp
int a = 1;
print(a);

a++;
print(a);

a--;
```

```
print(a);

----------
----------
File: test/tests/helloworld.vp.out
Hello World!

----------
----------
File: test/tests/cast.vp.out
9
99
888
false
1
0
8
d
h
----------
----------
File: test/tests/printbools.vp.out
1
0
----------
----------
File: test/tests/arrowfunction_call.vp
int func sum (int c, int d) => c + d;
bool func between (int i, int lowerb, int upperb) => (i > lowerb && i < upperb);


print(sum(10, 40));
print(between(5, 1, 9));

----------
----------
File: test/tests/arrays.vp.out
1
2
3
4
5
a
b
c
d
----------
----------
File: test/tests/strings.vp
string test = "bla";
print(test);
string test2 = "hi";
```

```
string test3 = test2;
print(test3);

test = "hello";
print(test);



print("@!#$*(YW(EF*H))");
print("\rwow\twow\nwow\rwow\twow\nwow");
print("11234569097654");
----------
----------
File: test/tests/index_access.vp
int[] d = [1,2,3,4];
char[] c = ['a'];

print(d[2]);
print(c[0]);

/* throws error
print(c[-1]);
*/

----------
----------
File: test/tests/attribute_call.vp.out
testing with no parameters:
42
testing with many parameters:
0
----------
----------
File: test/tests/printbools.vp
print(true);
print(false);

----------
----------
File: test/tests/variable_decl.vp.out
10
20
10
20
a
1
0
1.1
2.2
3.3

----------
----------
```

```
File: test/tests/cast.vp
print(toInt(9.34));
print(toInt('c'));
print(toString(888));
print(toString(false));
print(toBool(400));
print(toBool(""));
print(toFloat(8));
print(toChar(100));
print(toChar("hi"));

----------
----------
File: test/tests/list-append-access.vp.out
trey
ratghav
1
2
A
B
1
2
----------
----------
File: test/tests/pow.vp
float a = pow2(2.0);
print(a);

float b = pow2(a);

print(b);

float c = pow2(876.0);
print(c);



----------
----------
File: test/tests/shorthands.vp
int a = 10;
int b = 20;
int c = 0;

/* 0 */
print(c);

/* 10 */
c += a;
print(c);

/* -10 */
```

```
c -= b;
print(c);

/* 190 */
c += a * b;
print(c);

/* -10 */
c -= a * b;
print(c);

/* 100 */
c *= c;
print(c);
----------
----------
File: test/tests/shorthands.vp.out
0
10
-10
190
-10
100

----------
----------
File: test/tests/arrowfunction_call.vp.out
50
1
----------
----------
File: test/tests/attribute_call.vp
int func tester(int x) {
  return 42;
}

bool func least(int a, int b, int d, int e, int f) {
  return a > b;
}

int z = 42.tester();
print("testing with no parameters:");
print(z);

print("testing with many parameters:");
int isLeast = 5;
print(isLeast.least(6, 3, -9, 88));
----------
----------
File: test/tests/arrays.vp

int[] s = [1, 2, 3, 4, 5];
```

```
char[] c = ['a', 'b', 'c', 'd'];
int[] e;

print(s[0]);
print(s[1]);
print(s[2]);
print(s[3]);
print(s[4]);

print(c[0]);
print(c[1]);
print(c[2]);
print(c[3]);



----------
----------
File: test/tests/skip.vp
for (int i = 0; i < 10; i += 1){
    if (i == 5) {
        int x = 444;
        print("bad");
        skip;
    }
    print(i);
}
print();
for (int i = 0; i < 100; i+= 1) {
    print(i);
    if (i < 50) {
        if (i % 4 == 0){
            print("skipping");
            skip;
        }
    }
    print("did not skip");
}

----------
----------
File: test/tests/printnum-seq.vp
print(3);
print(4);
print(5);

----------
----------
File: test/tests/printfloat.vp
print(3.0);
print(38264.1);
print(0.0);
```

```
print(-99.9);

----------
----------
File: test/tests/indexed_function.vp
int[] func returnArray() {
    return [1,2,3,4];
}

print(returnArray()[3]);

[char: int] func returnDict() {
    return ['A': 1, 'B': 2, 'C': 3];
}

print(returnDict()['C']);
----------
----------
File: test/tests/abort.vp.out
testing skip and abort together:
0
1
2
3
5



---------

testing abort in a while loop:
0
1
2
3

testing skip and abort with nested loops:
aborted inner for loop
1
0
aborted inner for loop
2
0
2
1
aborted inner for loop
4
0
4
1
4
2
4
```

```
3
aborted inner for loop
5
0
5
1
5
2
5
3
5
4
aborted inner for loop
6
0
6
1
6
2
6
3
6
4
6
5
aborted inner for loop
7
0
7
1
7
2
7
3
7
4
7
5
7
6
aborted inner for loop
8
0
8
1
8
2
8
3
8
4
8
```

```
5
8
6
8
7
aborted inner for loop
9
0
9
1
9
2
9
3
9
4
9
5
9
6
9
7
9
8
aborted inner for loop
10
0
10
1
10
2
10
3
10
4
10
5
10
6
10
7
10
8
10
9
aborted inner for loop
terminated outer while loop
----------
----------
File: test/tests/printnum-seq.vp.out
3
4
```

```
5

----------
----------
File: test/tests/abort.vp
print("testing skip and abort together:");
for (int i = 0; i < 10; i += 1) {
    if (i == 4) {
        skip;
    }
    if (i > 5) {
        abort;
    }
    print(i);
}

print();
print();
print("---------");
print();
print("testing abort in a while loop:");
int a = -1;
while (a += 1 < 80) {
    if (a > 3) {
        abort;
    }
    print(a);
}

/* Throws error
if (a > 9) {
    abort;
} */

print();
print("testing skip and abort with nested loops:");
int x = -1;
while (x < 10) {
    x += 1;
    if (x == 3) {
        skip;
    }
    int i = 0;
    for (i; i < 10; i += 1) {
        if (x > i) {
            print(x);
            print(i);
        } else {
            abort;
        }
    }
    print("aborted inner for loop");
```

```
}
print("terminated outer while loop");




----------
----------
File: test/tests/variable_decl.vp
int a = 10;
print(a);
int b = 20;
print(b);
int c;
c = 10;
print(c);
int d = c + 10;
print(d);

char e = 'a';
print(e);

bool x = true;
print(x);
bool y = false;
print(y);

float xx = 1.1;
print(xx);
float zz = 2.2;
print(zz);
float aa = xx + zz;
print(aa);

----------
----------
File: test/tests/printchar-seq.vp.out
r
a
t

----------
----------
File: test/tests/index_access.vp.out
3
a

----------
----------
File: test/tests/strings.vp.out
bla
hi
```

```
hello
@!#$*(YW(EF*H))
\rwow\twow\nwow\rwow\twow\nwow
11234569097654
----------
----------
File: test/tests/function_call.vp.out
100
54
1
0

----------
----------
File: test/tests/nestedloop.vp
for(int i = 0; i < 10; i++) {
    for(int j = 0; j < 10; j++) {
        print(j);
    }
}

----------
----------
File: test/tests/contains.vp.out
1
1
1
1
----------
----------
File: test/tests/ternary.vp.out

----------
----------
File: src/ast.ml
(* Abstract Syntax Tree and functions for printing it *)

type op = Add | Sub | Mult | Div | Mod | Equal | Neq | Less | Leq | Greater | Geq |
          And | Or | Has

type uop = Neg | Not | Incr | Decr

type typ =
    Int
  | Bool
  | Nah
  | Char
  | Float
  | String
  | Array of typ
  | Function of typ
  | Group of typ list
```

```
  | Dictionary of typ * typ

type bind = typ * string

type expr =
    IntegerLiteral of int
  | CharacterLiteral of char
  | BoolLit of bool
  | FloatLiteral of float
  | StringLiteral of string
  | ListLit of expr list
  | DictElem of expr * expr
  | DictLit of expr list

  | Id of string
  | Binop of expr * op * expr
  | Unop of uop * expr
  | Ternop of expr * expr * expr

  | Assign of string * expr
  | Deconstruct of bind list * expr
  | OpAssign of string * op * expr
  | DecAssign of typ * string * expr
  | Access of expr * expr
  | AccessAssign of expr * expr * expr

  | MatchPattern of expr list * expr
  | ConditionalPattern of expr * expr
  | PatternMatch of string * expr
  | DecPatternMatch of typ * string * expr

  | Call of string * expr list
  | AttributeCall of expr * string * expr list

  | Noexpr

type stmt =
    Block of stmt list
  | PretendBlock of stmt list
  | Expr of expr
  | Dec of typ * string
  | Return of expr
  | Skip of expr
  | Abort of expr
  | Panic of expr
  | If of expr * stmt * stmt
  | For of expr * expr * expr * stmt
  | ForIter of string * expr * stmt
  | DecForIter of typ * string * expr * stmt
  | DeconstForIter of bind list * expr * stmt
  | While of expr * stmt * stmt
```

```
type func_decl = {
    typ : typ;
    fname : string;
    formals : bind list;
    body : stmt list;
    autoreturn: bool;
  }

type program = stmt list * func_decl list

(* Pretty-printing functions *)

let string_of_op = function
    Add -> "+"
  | Sub -> "-"
  | Mult -> "*"
  | Div -> "/"
  | Mod -> "%"
  | Equal -> "=="
  | Neq -> "!="
  | Less -> "<"
  | Leq -> "<="
  | Greater -> ">"
  | Geq -> ">="
  | And -> "&&"
  | Or -> "||"
  | Has -> "has"

let string_of_uop = function
    Neg -> "-"
  | Not -> "!"
  | Incr -> "++"
  | Decr -> "--"

let rec string_of_typ = function
    Int -> "int"
  | Bool -> "bool"
  | Nah -> "nah"
  | Char -> "char"
  | Float -> "float"
  | String -> "string"
  | Array(t) -> string_of_typ t ^ "[]"
  | Function(t) -> string_of_typ t ^ " func"
  | Group(t) -> "(" ^ String.concat ", " (List.map string_of_typ t) ^ ")"
  | Dictionary(t1, t2) -> "[" ^ string_of_typ t1 ^ ":" ^ string_of_typ t2 ^ "]"

let rec string_of_expr = function
    IntegerLiteral(l) -> string_of_int l
  | CharacterLiteral(l) -> "'" ^ Char.escaped l ^ "'"
  | BoolLit(true) -> "true"
  | BoolLit(false) -> "false"
  | FloatLiteral(l) -> string_of_float l
```

```
  | StringLiteral(s) -> "\"" ^ s ^ "\""
  | ListLit(lst) -> "[" ^ String.concat ", " (List.map string_of_expr lst) ^ "]"
  | DictElem(e1, e2) -> string_of_expr e1 ^ ": " ^ string_of_expr e2
  | DictLit(lst) ->  "[" ^ String.concat ", " (List.map string_of_expr lst) ^ "]"

  | Id(s) -> s
  | Binop(e1, o, e2) ->
      string_of_expr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_expr e2
  | OpAssign(v, o, e) -> v ^ " " ^ string_of_op o ^ "= " ^ string_of_expr e
  | Ternop(e1, e2, e3) -> string_of_expr e1 ^ " ? " ^ string_of_expr e2 ^ " : " ^
string_of_expr e3
  | Unop(o, e) -> string_of_uop o ^ string_of_expr e
  | Assign(v, e) -> v ^ " = " ^ string_of_expr e
  | Access(e, l) -> string_of_expr e ^ "[" ^ string_of_expr l ^ "]" (*List.fold_left
(fun s e -> s ^ "[" ^ string_of_expr e ^ "]") "" l*)
  | DecAssign(t, v, e) -> string_of_typ t ^ " " ^ v ^ " = " ^ string_of_expr e
  | Deconstruct(v, e) ->  "(" ^ String.concat ", " (List.map snd v) ^ ") = " ^
string_of_expr e

  | AccessAssign(i, idx, e) -> string_of_expr i ^ "[" ^ string_of_expr idx ^ "]" ^ " =
" ^ string_of_expr e

  | ConditionalPattern(c, r) -> string_of_expr c ^ " : " ^ string_of_expr r
  | MatchPattern(c, b) -> "?? " ^ String.concat " | " (List.map string_of_expr c) ^ "
?? " ^ string_of_expr b
  | PatternMatch(s, e) -> s ^ " = " ^ string_of_expr e
  | DecPatternMatch(t, s, e) -> string_of_typ t ^ " " ^ s ^ " = " ^ string_of_expr e

  | Call(f, el) -> f ^ "(" ^ String.concat ", " (List.map string_of_expr el) ^ ")"
  | AttributeCall(e, f, el) -> string_of_expr e ^ "." ^ f ^ "(" ^ String.concat ", "
(List.map string_of_expr el) ^ ")"

  | Noexpr -> ""

let rec string_of_stmt = function
    Block(stmts) ->
      "{\n" ^ String.concat "" (List.map string_of_stmt stmts) ^ "}\n"
  | PretendBlock(stmts) -> "\n" ^ String.concat "" (List.map string_of_stmt stmts) ^
"\n"
  | Expr(expr) -> string_of_expr expr ^ ";\n";
  | Dec(t, v) -> string_of_typ t ^ " " ^ v ^ ";\n";
  | Return(expr) -> "return " ^ string_of_expr expr ^ ";\n";
  | Skip(expr) -> "skip " ^ string_of_expr expr ^ ";\n";
  | Abort(expr) -> "abort " ^ string_of_expr expr ^ ";\n";
  | Panic(expr) -> "panic " ^ string_of_expr expr ^ ";\n";
  | If(e, s, Block([])) -> "if (" ^ string_of_expr e ^ ")\n" ^ string_of_stmt s
  | If(e, s1, s2) ->  "if (" ^ string_of_expr e ^ ")\n" ^
      string_of_stmt s1 ^ "else\n" ^ string_of_stmt s2
  | For(e1, e2, e3, s) ->
      "for (" ^ string_of_expr e1  ^ " ; " ^ string_of_expr e2 ^ " ; " ^
      string_of_expr e3  ^ ") " ^ string_of_stmt s
  | ForIter(name, e2, s) ->
```

```
      "for (" ^ name ^ " in " ^ string_of_expr e2 ^ ") " ^ string_of_stmt s
  | DecForIter(t, name, e2, s) ->
      "for (" ^ string_of_typ t ^ " " ^ name ^ " in " ^ string_of_expr e2 ^ ") " ^
string_of_stmt s
  | DeconstForIter(p, expr, s) -> "for ((" ^ String.concat ", " (List.map snd p) ^ ")
in " ^ string_of_expr expr ^ ") " ^ string_of_stmt s
  | While(e, s, _) -> "while (" ^ string_of_expr e ^ ") " ^ string_of_stmt s

let string_of_vdecl (t, id) = string_of_typ t ^ " " ^ id ^ ";\n"

let string_of_fdecl fdecl =
  string_of_typ fdecl.typ ^ " " ^
  fdecl.fname ^ "(" ^ String.concat ", " (List.map snd fdecl.formals) ^
  ")\n{\n" ^ (if fdecl.autoreturn then "return " else "") ^
  String.concat "" (List.map string_of_stmt fdecl.body) ^
  "}\n"

let string_of_program (sts, funcs) =
  (* Do we reverse to pretty-print? Is upside down stuff an indicator of the AST *)
  String.concat "" (List.map string_of_fdecl funcs) ^ "\n" ^
  String.concat "\n" (List.map string_of_stmt (List.rev sts))


----------
----------
File: src/rhandlhand.ml
(*
Comparison checks to make sure the LHS and RHS of an expression work out
*)

open Sast
open Ast

let check_decassign ty1 s expr1 =
    match expr1 with
        (* Allows declarations of empty list and dictionary literals *)
        (Array(Nah), SListLiteral([])) -> (ty1, SDecAssign(ty1, s, expr1))
      | (Dictionary(Nah, Nah), SDictLiteral([])) -> (ty1, SDecAssign(ty1, s, expr1))
      | (Nah, SDictLiteral([])) -> (ty1, SDecAssign(ty1, s, (ty1, SDictLiteral([]))))
      | (ty2, _) when ty1 = ty2 -> (ty1, SDecAssign(ty1, s, expr1))
      | (ty2, _) -> raise (Failure ("lvalue " ^ string_of_typ ty1 ^ " not equal to
rvalue " ^ string_of_typ ty2))
  (*| DecAssign(ty, string, expr1) -> check_assign ty string (expr expr1)*)

let check_assign lvaluet rvaluet  =
      if lvaluet = rvaluet then lvaluet else raise (Failure "wrong assignment")


----------
----------
File: src/sast.ml
(* Semantically-checked Abstract Syntax Tree and functions for printing it *)
```

```
open Ast

type sexpr = typ * sx
and sx =
    SIntegerLiteral of int
  | SCharacterLiteral of char
  | SBoolLiteral of bool
  | SFloatLiteral of float
  | SStringLiteral of string
  | SListLiteral of sexpr list
  | SDictElem of sexpr * sexpr
  | SDictLiteral of sexpr list

  | SId of string
  | SBinop of sexpr * op * sexpr
  | SUnop of uop * sexpr

  | SAssign of string * sexpr
  | SDeconstruct of bind list * sexpr
  | SOpAssign of string * op * sexpr
  | SDecAssign of typ * string * sexpr
  | SAccess of sexpr * sexpr
  | SAccessAssign of sexpr * sexpr * sexpr

  | SCall of string * sexpr list
  | SAttributeCall of sexpr * string * sexpr list

  | SNoexpr

type sstmt =
    SBlock of sstmt list
  | SExpr of sexpr
  | SDec of typ * string
  | SReturn of sexpr
  | SSkip of sexpr
  | SAbort of sexpr
  | SPanic of sexpr
  | SIf of sexpr * sstmt * sstmt
  | SWhile of sexpr * sstmt * sstmt

type sfunc_decl = {
  styp : typ;
  sfname : string;
  sformals : bind list;
  sbody : sstmt list;
}

type sprogram = sstmt list * sfunc_decl list

(* Pretty-printing functions *)
```

```
let rec string_of_sexpr (t, e) =
  "(" ^ string_of_typ t ^ " : " ^ (match e with
    SIntegerLiteral(l) -> string_of_int l
  | SCharacterLiteral(l) -> "'" ^ Char.escaped l ^ "'"
  | SBoolLiteral(true) -> "true"
  | SBoolLiteral(false) -> "false"
  | SFloatLiteral(l) -> string_of_float l
  | SStringLiteral(s) -> "\"" ^ s ^ "\""
  | SListLiteral(list) -> "[" ^ String.concat ", " (List.map string_of_sexpr list) ^
"]"
  | SDictElem(e1, e2) -> string_of_sexpr e1 ^ ": " ^ string_of_sexpr e2
  | SDictLiteral(list) -> "[" ^ String.concat ", " (List.map string_of_sexpr list) ^
"]"

  | SId(s) -> s
  | SUnop(o, e) -> string_of_uop o ^ string_of_sexpr e
  | SBinop(e1, o, e2) ->
      string_of_sexpr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_sexpr e2
  | SOpAssign(v, o, e) -> v ^ " " ^ string_of_op o ^ "= " ^ string_of_sexpr e
  | SAssign(v, e) -> v ^ " = " ^ string_of_sexpr e
  | SAccess(e, l) -> string_of_sexpr e ^ "[" ^ string_of_sexpr l ^ "]"
  | SAccessAssign(i, index, e) -> string_of_sexpr i ^ "[" ^ string_of_sexpr index ^ "]
= " ^ string_of_sexpr e
  | SDecAssign(t, v, e) -> string_of_typ t ^ " " ^ v ^ " = " ^ string_of_sexpr e
  | SDeconstruct(v, e) -> "(" ^ String.concat ", " (List.map snd v) ^ ") = " ^
string_of_sexpr e

  | SCall(f, el) -> f ^ "(" ^ String.concat ", " (List.map string_of_sexpr el) ^ ")"
  | SAttributeCall(e, f, el) -> string_of_sexpr e ^ "." ^ f ^ "(" ^ String.concat ", "
(List.map string_of_sexpr el) ^ ")"

  | SNoexpr -> "")
    ^ ")"

let rec string_of_sstmt = function
    SBlock(stmts) -> "{\n" ^ String.concat "" (List.map string_of_sstmt stmts) ^ "}\n"
  | SExpr(expr) -> string_of_sexpr expr ^ ";\n"
  | SDec(t, v) -> string_of_typ t ^ " " ^ v ^ ";\n"
  | SReturn(expr) -> "return " ^ string_of_sexpr expr ^ ";\n"
  | SSkip(expr) -> "skip " ^ string_of_sexpr expr ^ ";\n"
  | SAbort(expr) -> "abort " ^ string_of_sexpr expr ^ ";\n"
  | SPanic(expr) -> "panic " ^ string_of_sexpr expr ^ ";\n"
  | SIf(e, s, SBlock([])) -> "if (" ^ string_of_sexpr e ^ ")\n" ^ string_of_sstmt s
  | SIf(e, s1, s2) ->  "if (" ^ string_of_sexpr e ^ ")\n" ^ string_of_sstmt s1 ^
"else\n" ^ string_of_sstmt s2
  | SWhile(e, s, _) -> "while (" ^ string_of_sexpr e ^ ") " ^ string_of_sstmt s

let string_of_sfdecl fdecl =
  string_of_typ fdecl.styp ^ " " ^
  fdecl.sfname ^ "(" ^ String.concat ", " (List.map snd fdecl.sformals) ^
  ")\n{\n" ^
  String.concat "" (List.map string_of_sstmt fdecl.sbody) ^
```

```
  "}\n"

let string_of_sprogram (sts, funcs) =
  String.concat "" (List.map string_of_sfdecl funcs) ^ "\n" ^
  String.concat "\n" (List.map string_of_sstmt (List.rev sts))

----------
----------
File: src/scanner.mll
(* Ocamllex scanner for viper, adapted from that of the MicroC compiler. Ref:
https://github.com/cwabbott0/microc-llvm *)

{ open Parser }

rule token = parse
  [' ' '\t' '\r' '\n'] { token lexbuf } (* Whitespace *)
| "/*"     { comment lexbuf }           (* Comments *)
| '('      { LPAREN }
| ')'      { RPAREN }
| '{'      { LBRACE }
| '}'      { RBRACE }
| '['      { ARROPEN }
| ']'      { ARRCLOSE }
| ';'      { SEMI }
| ','      { COMMA }
| '+'      { PLUS }
| '-'      { MINUS }
| '*'      { TIMES }
| '/'      { DIVIDE }
| '%'      { MODULO }
| '='      { ASSIGN }
| "+="     { PLUS_ASSIGN }
| "-="     { MINUS_ASSIGN }
| "*="     { TIMES_ASSIGN }
| "/="     { DIVIDE_ASSIGN }
| "=="     { EQ }
| "!="     { NEQ }
| '<'      { LT }
| "<="     { LEQ }
| ">"      { GT }
| ">="     { GEQ }
| "&&"     { AND }
| "||"     { OR }
| "!"      { NOT }
| "if"     { IF }
| "else"   { ELSE }
| "for"    { FOR }
| "while"  { WHILE }
| "return" { RETURN }
| "func"   { FUNC }
| "in"     { IN }
| "has"    { HAS }
```

```
| "int"    { INT }
| "char"   { CHAR }
| "bool"   { BOOL }
| "float"  { FLOAT }
| "string" { STRING }
| "nah"    { NAH }
| "=>"     { ARROW }
| "true"   { TRUE }
| "false"  { FALSE }
| "skip"   { SKIP }
| "abort"  { ABORT }
| "panic"  { PANIC }
| '?'      { QUESTION }
| "??"     { MATCH }
| '|'      { BAR }
| '.'      { DOT }
| ':'      { COLON }
| ['0'-'9']+['.']['0'-'9']+ as lxm { FLOATLIT(float_of_string lxm) }
| ['0'-'9']+ as lxm { INTLIT(int_of_string lxm) }
| ['\'']['\x20'-'\x7E'] as lxm)['\''] { CHARLIT(lxm) }
| ['\"']['\x20'-'\x21' '\x23' - '\x7E']* as lxm)['\"'] { STRLIT(lxm) }
| ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']* as lxm { ID(lxm) }
| eof { EOF }
| _ as char { raise (Failure("illegal character " ^ Char.escaped char)) }

and comment = parse
  "*/" { token lexbuf }
| _    { comment lexbuf }

----------
----------
File: src/boolcheck.ml
(*
One function that just checks if an if-condition or while loops have a bool predicate
*)

open Ast
open Sast
open Decs

let check_bool e =
    match e with
        (ty, l) -> if ty = Bool then (ty, l) else raise (Failure "value must be a
bool")

----------
----------
File: src/Makefile
# Make sure ocamlbuild can find opam-managed packages: first run
#
# eval `opam config env`
```

```
# Easiest way to build: using ocamlbuild, which in turn uses ocamlfind

.PHONY: all
all : viper.native library.o

.PHONY: viper.native
viper.native :
    ocamlbuild -use-ocamlfind -pkgs llvm,llvm.analysis,str -cflags -w,+a-4-42-45-27-
33-11-26-39-8-10-29-40 \
        viper.native

# "make clean" removes all generated files

.PHONY : clean
clean :
    ocamlbuild -clean
    rm -rf testall.log *.diff viper scanner.ml parser.ml parser.mli
    rm -rf printbig
    rm -rf *.cmx *.cmi *.cmo *.cmx *.o *.s *.ll *.out *.exe library

# More detailed: build using ocamlc/ocamlopt + ocamlfind to locate LLVM

OBJS = ast.cmx parser.cmx scanner.cmx viper.cmx semant.cmx

viper : $(OBJS)
    ocamlfind ocamlopt -linkpkg -package llvm -package llvm.analysis $(OBJS) -o viper

scanner.ml : scanner.mll
    ocamllex scanner.mll

parser.ml parser.mli : parser.mly
    ocamlyacc parser.mly

%.cmo : %.ml
    ocamlc -c $<

%.cmi : %.mli
    ocamlc -c $<

%.cmx : %.ml
    ocamlfind ocamlopt -c -package llvm $<

### Generated by "ocamldep *.ml *.mli" after building scanner.ml and parser.ml
ast.cmo :
ast.cmx :
viper.cmo : scanner.cmo parser.cmi ast.cmo
viper.cmx : scanner.cmx parser.cmx ast.cmx
parser.cmo : ast.cmo parser.cmi
parser.cmx : ast.cmx parser.cmi
scanner.cmo : parser.cmi
scanner.cmx : parser.cmx
semant.cmo : ast.cmo
```

```
semant.cmx : ast.cmx
parser.cmi : ast.cmo

### C Standard Library

library : library.c
    cc -o library -DBUILD_TEST library.c -lm

----------
----------
File: src/viper.ml
(*
Top-level of the Viper compiler: scan & parse the input, check the resulting AST,
generate LLVM IR, and dump the module
REF: https://github.com/cwabbott0/microc-llvm/blob/master/microc.ml
*)

type action = Ast | Sast | LLVM_IR | Compile

let _ =
  let action = ref Compile in
  let input = ref "" in
  let set_action a () = action := a in
  let speclist = [
    ("-a", Arg.Unit (set_action Ast), "Pretty print the AST");
    ("-s", Arg.Unit (set_action Sast), "Pretty print the SAST");
    ("-l", Arg.Unit (set_action LLVM_IR), "Print the generated LLVM IR");
    ("-c", Arg.Unit (set_action Compile),
     "Check and print the generated LLVM IR (default)");
  ] in
  let usage_msg = "usage: ./viper.native [-a|-l] [file]" in
  Arg.parse speclist (fun s -> input := s) usage_msg;
  let channel = if !input = "" then
      stdin
    else
      open_in !input
  in
  let lexbuf = Lexing.from_channel channel in
  let ast = Parser.program Scanner.token lexbuf in
  (* this is sast, currently not used so replace _ with sast when used *)
  let desugared = Desugar.desugar ast in
  let sast = Semantdriver.check desugared in
  match !action with
    Ast -> print_string (Ast.string_of_program desugared)
  | Sast -> print_string (Sast.string_of_sprogram sast)
  | LLVM_IR -> print_string (Llvm.string_of_llmodule (Codegen.translate sast))
  | Compile -> let m = Codegen.translate sast in
    Llvm_analysis.assert_valid_module m;
    print_string (Llvm.string_of_llmodule m)

----------
----------
```

```
File: src/desugar.ml
(* Desugar module to desugar an AST *)

open Ast
let placeholderCheck ast = ast

exception SemanticException of string

let rec clean_pattern_rec e base = match e with
    ConditionalPattern(cond, exp) :: [] -> Ternop(cond, exp, base)
  | ConditionalPattern(cond, exp) :: tail -> Ternop(cond, exp, (clean_pattern_rec tail
base))
  | _ -> Noexpr


let rec clean_expression expr = match expr with
    MatchPattern(p, b) -> clean_expression (clean_pattern_rec p b)
  | PatternMatch(s, e) -> Assign(s, clean_expression e)
  | DecPatternMatch(t, s, e) -> DecAssign(t, s, clean_expression e)
  | Ternop(e, e1, e2) -> Ternop((clean_expression e), (clean_expression e1),
(clean_expression e2))

  | Binop(e1, op, e2) -> Binop((clean_expression e1), op, (clean_expression e2))
  | Unop(op, e) -> Unop(op, (clean_expression e))

  | Assign(n, e) -> Assign(n, (clean_expression e))
  | OpAssign(n, o, e) -> OpAssign(n, o, (clean_expression e))
  | DecAssign(t, n, e) -> DecAssign(t, n, (clean_expression e))
  | AccessAssign(e1, e2, e3) -> AccessAssign((clean_expression e1), (clean_expression
e2), (clean_expression e3))

  | Call(n, l) -> Call(n, (List.map clean_expression l))
  | AttributeCall(e, f, args) -> Call(f, List.map clean_expression (e::args))

  | _ -> expr

let decompose_deconstforiter n e s =
    let comparison = Binop(Id("dfi_tmp_idx"), Less, Call("len", [e]))
    in
    let (idx_t, idx_name) = List.hd n
    in
    let (el_t, el_name) = List.hd (List.rev n)
    in
    let exec = Block([ Expr(DecAssign(idx_t, idx_name, Id("dfi_tmp_idx")));
Expr(DecAssign(el_t, el_name, Access(e, Id("dfi_tmp_idx")))); s; Expr(Unop(Incr,
Id("dfi_tmp_idx")))  ])
    in
    While(comparison, exec, Expr(Noexpr))

let decompose_foriter n e s =
    let comparison = Binop(Id("dfi_tmp_idx"), Less, Call("len", [e]))
    in
```

```
    let (iterator : Ast.stmt) = Expr(Unop(Incr, Id("dfi_tmp_idx"))) in
    let exec = Block([ Expr(Assign(n, Access(e, Id("dfi_tmp_idx")))); s; iterator])
    in
    While(comparison, exec, iterator)

let decompose_decforiter t n e s =
    let comparison = Binop(Id("dfi_tmp_idx"), Less, Call("len", [e]))
    in
    let exec = Block([ Expr(DecAssign(t, n, Access(e, Id("dfi_tmp_idx")))); s;
Expr(Unop(Incr, Id("dfi_tmp_idx")))  ])
    in
    While(comparison, exec, Expr(Noexpr))

let clean_statements stmts =
    let rec clean_statement stmt = match stmt with
        Block(s) -> Block(List.map clean_statement s)
      | Expr(expr) -> Expr(clean_expression expr)
      | For(e1, e2, e3, s) -> Block([Expr(e1); While(e2, Block([ (clean_statement s);
Expr(e3)]), Expr(e3))])
      | ForIter(name, e2, s) -> Block([ Expr(DecAssign(Int, "dfi_tmp_idx",
IntegerLiteral(0))); decompose_foriter name e2 (clean_statement s)])
      | DecForIter(t, name, e2, s) -> Block([ Expr(DecAssign(Int, "dfi_tmp_idx",
IntegerLiteral(0))); decompose_decforiter t name e2 (clean_statement s)])
      | DeconstForIter(p, e, s) ->  Block([ Expr(DecAssign(Int, "dfi_tmp_idx",
IntegerLiteral(0))); decompose_deconstforiter p e (clean_statement s)])
      | While(e, s, iterator) -> While(clean_expression e, clean_statement s,
clean_statement iterator)
      | If(cond, t, f) -> If(clean_expression cond, clean_statement t, clean_statement
f)
      | _ -> stmt
    in
    (List.map clean_statement stmts)

let reshape_arrow_function fdecl = {
    typ=fdecl.typ;
    formals=fdecl.formals;
    fname=fdecl.fname;
    body = (match List.hd fdecl.body with
        Expr(e) -> [Return(e)]
      | _ -> [Return(Noexpr)]
      );
    autoreturn=false;
}
let clean_normal_function fdecl = { typ=fdecl.typ; formals=fdecl.formals;
fname=fdecl.fname; body = (clean_statements fdecl.body); autoreturn=fdecl.autoreturn;
}

let clean_function fdecl = if fdecl.autoreturn then reshape_arrow_function fdecl else
clean_normal_function fdecl

(*
let rec eliminate_ternaries stmts =
```

```
  let new_functions = [] in

  let rec eliminate_ternaries_from_expr expr = match expr with
      Ternop(e, e1, e2) -> (* Do something here, return append_function_call + do
recursively *)
    | _ -> expr
  in

  let rec clean_statement = match stmt with
    Expr(e) -> eliminate_ternaries_from_expr e
  | _ -> stmt
  in

  let cleaned_stmts = List.map clean_statement stmts in
  (cleaned_stmts, new_functions)
*)

let rec sanitize_expr e = match e with
    Binop(e1, op, e2) -> Binop((sanitize_expr e1), op, (sanitize_expr e2))
  | Unop(op, e) -> Unop(op, (sanitize_expr e))

  | Assign(n, e) -> Assign(n, (sanitize_expr e))
  | OpAssign(n, o, e) -> OpAssign(n, o, (sanitize_expr e))
  | DecAssign(t, n, e) -> DecAssign(t, n, (sanitize_expr e))
  | AccessAssign(e1, e2, e3) -> AccessAssign((sanitize_expr e1), (sanitize_expr e2),
(sanitize_expr e3))

  | Call(n, l) -> Call(n, (List.map sanitize_expr l))
  | AttributeCall(e, f, args) -> Call(f, List.map sanitize_expr (e::args))
  | Ternop(e, e1, e2) -> Id("ternop_tempvar")
  | _ -> e

let rec decompose_nested_ternary e e1 e2 = match e2 with
    Ternop(e', e1', e2') -> If(e, Expr(Assign("ternop_tempvar", e1)),
(decompose_nested_ternary e' e1' e2'))
  | _ -> If(e, Expr(Assign("ternop_tempvar", e1)), Expr(Assign("ternop_tempvar", e2)))

let rec check_for_ternary e t = match e with
    DecAssign(t, n, e) -> check_for_ternary e t
  | Ternop(e, e1, e2) -> (true, decompose_nested_ternary e e1 e2, "ternary_tmpvar", t)
  | _ -> (false, PretendBlock([]), "ternary_tempvar_none", Int)

let sanitize_ternaries stmts =
    let generate_pb_if_needed e =
        let (to_sanitize, required_stmt, name, t) = (check_for_ternary e Int) in
        if to_sanitize then PretendBlock([ Dec(t, "ternop_tempvar"); required_stmt;
Expr(sanitize_expr e) ]) else Expr(e)
    in

    (* int a = TERNARY *)
    let check_stmt stmt = match stmt with
    Expr(e) -> generate_pb_if_needed e
```

```
  | _ -> stmt

    in List.map check_stmt stmts

let sanitize_ternaries_f fdecl = { typ=fdecl.typ; formals=fdecl.formals;
fname=fdecl.fname; body = (sanitize_ternaries fdecl.body);
autoreturn=fdecl.autoreturn; }

let desugar (stmts, functions) =
  let cleaned_statements = clean_statements stmts in
  let cleaned_functions = List.map clean_function functions in
  let adjusted_statements = sanitize_ternaries cleaned_statements in
  let adjusted_functions = List.map sanitize_ternaries_f cleaned_functions in
  (adjusted_statements, adjusted_functions)

----------
----------
File: src/codegen.ml
(*
Codegen module to handle all the LLVM
*)

module L = Llvm
module A = Ast
module Str = Str
open Cast
open Sast

exception Error of string

module StringMap = Map.Make(String)

(* translate : Sast.program -> Llvm.module
   a viper program consists of statements and function defs
*)
let translate (_, functions) =
  let context    = L.global_context () in

  (* Create the LLVM compilation module into which
     we will generate code *)
  let the_module = L.create_module context "Viper" in

  (* define llytype variables *)
  let i64_t      = L.i64_type    context
  and i32_t      = L.i32_type    context
  and i16_t      = L.i16_type    context
  and i8_t       = L.i8_type     context
  and i1_t       = L.i1_type     context
  and float_t    = L.float_type  context
  and double_t   = L.double_type context
  and void_t     = L.void_type   context
  and struct_t   = L.struct_type context in
```

```
  let str_t      = L.pointer_type i8_t
  in

  (* STRUCT CODE HERE *)

  (* create hashtbl for the structs we use in standard library *)
  let struct_types:(string, L.lltype) Hashtbl.t = Hashtbl.create 3 in

  (* function to lookup struct type *)
  let find_struct_type name = try Hashtbl.find struct_types name
    with Not_found -> raise (Error "Invalid struct name")
  in

  (* declare struct type in llvm and retain its lltype in a map *)
  let declare_struct_typ name =
    let struct_type = L.named_struct_type context name in
    Hashtbl.add struct_types name struct_type
  in

  (* build struct body by passing lltypes its body contains*)
  let define_struct_body name lltypes =
    let struct_type = find_struct_type name in
    L.struct_set_body struct_type lltypes false
  in

  (* declare list struct*)
  let _ = declare_struct_typ "list"
  and _ = define_struct_body "list" [| L.pointer_type (L.pointer_type i8_t); i32_t;
i32_t; L.pointer_type i8_t |]

  (* declare dict_elem struct*)
  and _ = declare_struct_typ "dict_elem"
  and _ = define_struct_body "dict_elem" [| L.pointer_type i8_t; L.pointer_type i8_t
|]

  (* declare dict struct*)
  and _ = declare_struct_typ "dict"
  and _ = define_struct_body "dict" [| (find_struct_type "list"); L.pointer_type i8_t;
L.pointer_type i8_t |]
  in

  (* Return the LLVM lltype for a Viper type *)
  let rec ltype_of_typ = function
      A.Int                     -> i32_t
    | A.Bool                    -> i1_t
    | A.Nah                     -> void_t
    | A.Char                    -> i8_t
    | A.Float                   -> float_t
    | A.String                  -> str_t
    | A.Array(_)                -> (L.pointer_type (find_struct_type "list"))
    | A.Dictionary(_, _)        -> (L.pointer_type (find_struct_type "dict"))
    | A.Function(_)             -> raise (Error "Function lltype not implemented")
```

```
    | A.Group(_)                -> raise (Error "Group lltype not implemented")
  in

  (* Return an initial value for a declaration *)
  let rec lvalue_of_typ typ = function
      A.Int | A.Bool | A.Nah | A.Char -> L.const_int (ltype_of_typ typ) 0
    | A.Float                         -> L.const_float (ltype_of_typ typ) 0.0
    | A.String                        -> L.const_pointer_null (ltype_of_typ typ)
    | A.Array(_)                      -> L.const_pointer_null (find_struct_type
"list")
    | A.Dictionary(_, _)              -> L.const_pointer_null (find_struct_type
"dict")
    | A.Function(_)                   -> raise (Error "Function lltype not
implemented")
    | A.Group(_)                      -> raise (Error "Group llvalue not implemented")
  in

  (* BUILT-IN FUNCTIONS HERE*)
  (* These functions are either built-in C functions or defined in our library.c file
*)

  (* PRINT FUNCTIONS HERE *)

  (* Prints a char[] list *)
  let print_char_list_t : L.lltype =
    L.function_type (ltype_of_typ A.Nah) [| (L.pointer_type (find_struct_type "list"))
|] in
  let print_char_list_func : L.llvalue =
    L.declare_function "print_char_list" print_char_list_t the_module in

  (* Prints a string[] list *)
  let print_str_list_t : L.lltype =
    L.function_type (ltype_of_typ A.Nah) [| (L.pointer_type (find_struct_type "list"))
|] in
  let print_str_list_func : L.llvalue =
    L.declare_function "print_str_list" print_str_list_t the_module in

  (* Prints an int[] list *)
  let print_int_list_t : L.lltype =
    L.function_type (ltype_of_typ A.Nah) [| (L.pointer_type (find_struct_type "list"))
|] in
  let print_int_list_func : L.llvalue =
    L.declare_function "print_int_list" print_int_list_t the_module in

  (* C STANDARD LIBRARY FUNCTIONS HERE *)

  (* C's printf function *)
  let printf_t : L.lltype =
    L.var_arg_function_type i32_t [| L.pointer_type i8_t |] in
  let printf_func : L.llvalue =
    L.declare_function "printf" printf_t the_module in
```

```
  (* Raises a float to the power of 2 *)
  let pow2_t : L.lltype =
    L.function_type float_t [| float_t |] in
  let pow2_func : L.llvalue =
    L.declare_function "pow2" pow2_t the_module in

  (* BUILTIN LIST FUNCTIONS HERE*)

  (* Creates an empty list given a type string *)
  let create_list_t : L.lltype =
    L.function_type (L.pointer_type (find_struct_type "list")) [| L.pointer_type i8_t
|] in
  let create_list_func : L.llvalue =
    L.declare_function "create_list" create_list_t the_module in

  (* given a pointer to list, returns length attribute out of struct *)
  let listlen_t : L.lltype =
    L.function_type (ltype_of_typ A.Int) [| (L.pointer_type (find_struct_type "list"))
|] in
  let listlen_func : L.llvalue =
    L.declare_function "listlen" listlen_t the_module in

  (* ACCESS LIST FUNCTIONS HERE*)

  (* takes in a char[] list and an int and returns the value at the index *)
  let access_char_t : L.lltype =
    L.function_type (ltype_of_typ A.Char) [| (L.pointer_type (find_struct_type
"list")); (ltype_of_typ A.Int) |] in
  let access_char_func : L.llvalue =
    L.declare_function "access_char" access_char_t the_module in

  (* takes in a int[] list and an int and returns the value at the index *)
  let access_int_t : L.lltype =
    L.function_type (ltype_of_typ A.Int) [| (L.pointer_type (find_struct_type
"list")); (ltype_of_typ A.Int)|] in
  let access_int_func : L.llvalue =
    L.declare_function "access_int" access_int_t the_module in

  (* takes in a string[] list and an int and returns the value at the index *)
  let access_str_t : L.lltype =
    L.function_type (ltype_of_typ A.String) [| (L.pointer_type (find_struct_type
"list")); (ltype_of_typ A.Int)|] in
  let access_str_func : L.llvalue =
    L.declare_function "access_str" access_str_t the_module in

  (* takes in a float[] list and an int and returns the value at the index*)
  let access_float_t : L.lltype =
    L.function_type (ltype_of_typ A.Float) [| (L.pointer_type (find_struct_type
"list")); (ltype_of_typ A.Int)|] in
  let access_float_func : L.llvalue =
    L.declare_function "access_float" access_float_t the_module in
```

```
  (* APPEND LIST FUNCTIONS HERE *)

  (* appends a char to a char[] list *)
  let append_char_t : L.lltype =
    L.function_type (ltype_of_typ A.Nah) [| (L.pointer_type (find_struct_type
"list")); (ltype_of_typ A.Char) |] in
  let append_char_func : L.llvalue =
    L.declare_function "append_char" append_char_t the_module in

  (* appends an int to a int[] list *)
  let append_int_t : L.lltype =
    L.function_type (ltype_of_typ A.Nah) [| (L.pointer_type (find_struct_type
"list")); (ltype_of_typ A.Int) |] in
  let append_int_func : L.llvalue =
    L.declare_function "append_int" append_int_t the_module in

  (* appends a string to a string[] list *)
  let append_str_t : L.lltype =
    L.function_type (ltype_of_typ A.Nah) [| (L.pointer_type (find_struct_type
"list")); (ltype_of_typ A.String) |] in
  let append_str_func : L.llvalue =
    L.declare_function "append_str" append_str_t the_module in

  (* appends a float to a float[] list *)
  let append_float_t : L.lltype =
    L.function_type (ltype_of_typ A.Nah) [| (L.pointer_type (find_struct_type
"list")); (ltype_of_typ A.Float) |] in
  let append_float_func : L.llvalue =
    L.declare_function "append_float" append_float_t the_module in

  (* CONTAINS LIST FUNCTIONS HERE *)

  let contains_char_t : L.lltype =
    L.function_type (ltype_of_typ A.Int) [| (L.pointer_type (find_struct_type
"list")); (ltype_of_typ A.Char) |] in
  let contains_char_func : L.llvalue =
    L.declare_function "contains_char" contains_char_t the_module in

  let contains_int_t : L.lltype =
    L.function_type (ltype_of_typ A.Int) [| (L.pointer_type (find_struct_type
"list")); (ltype_of_typ A.Int) |] in
  let contains_int_func : L.llvalue =
    L.declare_function "contains_int" contains_int_t the_module in

  let contains_str_t : L.lltype =
    L.function_type (ltype_of_typ A.Int) [| (L.pointer_type (find_struct_type
"list")); (ltype_of_typ A.String) |] in
  let contains_str_func : L.llvalue =
    L.declare_function "contains_str" contains_str_t the_module in

  let contains_float_t : L.lltype =
    L.function_type (ltype_of_typ A.Int) [| (L.pointer_type (find_struct_type
```

```
"list")); (ltype_of_typ A.Float) |] in
  let contains_float_func : L.llvalue =
    L.declare_function "contains_float" contains_float_t the_module in

  (* ASSIGN LIST FUNCTIONS HERE *)

  let assign_int_t : L.lltype =
    L.function_type (ltype_of_typ A.Int) [| (L.pointer_type (find_struct_type
"list")); (ltype_of_typ A.Int); (ltype_of_typ A.Int) |] in
  let assign_int_func : L.llvalue =
    L.declare_function "assign_int" assign_int_t the_module in

  (* BUILTIN DICT FUNCTIONS HERE *)

  (* takes in a char pointer to a string of the type *)
  let create_dict_t : L.lltype =
    L.function_type (L.pointer_type (find_struct_type "dict")) [| L.pointer_type i8_t;
L.pointer_type i8_t |] in
  let create_dict_func : L.llvalue =
    L.declare_function "create_dict" create_dict_t the_module in

  (* takes in a dict pointer and a void pointer to key and val and adds pair to dict
*)
  let add_keyval_t : L.lltype =
    L.function_type (ltype_of_typ A.Nah) [| (L.pointer_type (find_struct_type
"dict")); (L.pointer_type i8_t); (L.pointer_type i8_t) |] in
  let add_keyval_func : L.llvalue =
    L.declare_function "add_keyval" add_keyval_t the_module in

  (* DICT ACCESS FUNCTIONS HERE *)

  (* takes in a dict and a char key and returns a void pointer to the value *)
  let access_char_key_t : L.lltype =
    L.function_type (L.pointer_type i8_t) [| (L.pointer_type (find_struct_type
"dict")); (ltype_of_typ A.Char) |] in
  let access_char_key_func : L.llvalue =
    L.declare_function "access_char_key" access_char_key_t the_module in

  let access_str_key_t : L.lltype =
    L.function_type (L.pointer_type i8_t) [| (L.pointer_type (find_struct_type
"dict")); (L.pointer_type (ltype_of_typ A.Char)) |] in
  let access_str_key_func : L.llvalue =
    L.declare_function "access_str_key" access_str_key_t the_module in

  (* TYPE ALLOC FUNCTIONS HERE *)

  let int_alloc_t : L.lltype =
    L.function_type (L.pointer_type i8_t) [| (ltype_of_typ A.Int) |] in
  let int_alloc_func : L.llvalue =
    L.declare_function "int_alloc_zone" int_alloc_t the_module in

  let char_alloc_t : L.lltype =
```

```
    L.function_type (L.pointer_type i8_t) [| (ltype_of_typ A.Char) |] in
  let char_alloc_func : L.llvalue =
    L.declare_function "char_alloc_zone" char_alloc_t the_module in

  let str_alloc_t : L.lltype =
    L.function_type (L.pointer_type i8_t) [| (ltype_of_typ A.String) |] in
  let str_alloc_func : L.llvalue =
    L.declare_function "str_alloc_zone" str_alloc_t the_module in

  (* CONTAINS DICT FUNCTIONS HERE *)

  let contains_str_key_t : L.lltype =
    L.function_type (ltype_of_typ A.Int) [| (L.pointer_type (find_struct_type
"dict")); (L.pointer_type (ltype_of_typ A.Char)) |] in
  let contains_str_key_func : L.llvalue =
    L.declare_function "contains_str_key" contains_str_key_t the_module in

  (* GET DICT FUNCTIONS *)

  let get_str_keys_t : L.lltype =
    L.function_type (L.pointer_type (find_struct_type "list")) [| (L.pointer_type
(find_struct_type "dict")) |] in
  let get_str_keys_func: L.llvalue =
    L.declare_function "get_str_keys" get_str_keys_t the_module in

  (* Define each function (arguments and return type) so we can
     call it even before we've created its body *)
  let function_decls : (L.llvalue * sfunc_decl) StringMap.t =
    let function_decl m fdecl =
      let name = fdecl.sfname
      and formal_types =
        Array.of_list (List.map (fun (t,_) -> ltype_of_typ t) fdecl.sformals)
      in let ftype = L.function_type (ltype_of_typ fdecl.styp) formal_types in
      StringMap.add name (L.define_function name ftype the_module, fdecl) m in
    List.fold_left function_decl StringMap.empty functions in

  (* Fill in the body of the given function *)
  let build_function_body fdecl =
    let (the_function, _) = try StringMap.find fdecl.sfname function_decls
      with Not_found -> raise (Error "function definition not found")
    in
    let builder = L.builder_at_end context (L.entry_block the_function) in

    (* define global stringptr's for format str *)
    let char_format_str = L.build_global_stringptr "%c\n" "fmt" builder
    and int_format_str = L.build_global_stringptr "%d\n" "fmt" builder
    and str_format_str = L.build_global_stringptr "%s\n" "fmt" builder
    and float_format_str = L.build_global_stringptr "%g\n" "fmt" builder
    in

    (* create empty local_vars Hashtbl to track vars llvalue *)
    let local_vars:(string, L.llvalue) Hashtbl.t = Hashtbl.create 50 in
```

```
    (* function takes in a formal binding and parameter values to add to locals map *)
    let add_formal (t, n) p =
      L.set_value_name n p;
      let local = L.build_alloca (ltype_of_typ t) n builder in
      ignore (L.build_store p local builder);
      Hashtbl.add local_vars n local;
    in

    (* iterate over the list of formal bindings and their values to add to local_vars
*)
    let _ = List.iter2 add_formal fdecl.sformals (Array.to_list (L.params
the_function)) in

    (* Return the llvalue for a variable or formal argument *)
    let lookup n = try Hashtbl.find local_vars n
      with Not_found -> raise (Error ("variable " ^ n ^ " not found in locals map"))
    in

    (* LLVM insists each basic block end with exactly one "terminator"
       instruction that transfers control.  This function runs "instr builder"
       if the current block does not already have a terminator.  Used,
       e.g., to handle the "fall off the end of the function" case. *)
    let add_terminal builder instr =
      match L.block_terminator (L.insertion_block builder) with
        Some _ -> ()
      | None -> ignore (instr builder) in

    (* maps a Viper type to a string for use in create array/dict functions *)
    let rec get_type_string e_type = match e_type with
      | A.Array (li_t)              -> get_type_string li_t
      | A.Dictionary (key_t, val_t) ->
        let key_t_str = get_type_string key_t in
        let val_t_str = get_type_string val_t in
        key_t_str ^ " " ^ val_t_str
      | A.Char    -> "char"
      | A.Int     -> "int"
      | A.Nah     -> "nah"
      | A.String  -> "string"
      | A.Float   -> "float"
      | _         -> raise (Error "type string map not implemented")
    in

    (* returns an llvalue for an expression, builds necessary instructions for
evaluation *)
    let rec expr builder ((e_type, e) : sexpr) = match e with
        SIntegerLiteral(num)      -> L.const_int (ltype_of_typ A.Int) num
      | SCharacterLiteral(chr)    -> L.const_int (ltype_of_typ A.Char) (Char.code chr)
      | SBoolLiteral(bln)         -> L.const_int (ltype_of_typ A.Bool) (if bln then 1
else 0)
      | SFloatLiteral(flt)        -> L.const_float (ltype_of_typ A.Float) flt
      | SStringLiteral(str)       -> L.build_global_stringptr str "" builder
```

```
      | SListLiteral(list)        ->
        let type_string = (get_type_string e_type) in
        let type_string_ptr = expr builder (A.String, SStringLiteral(type_string)) in
        (* create empty list llvalue *)
        let li = L.build_call create_list_func [| type_string_ptr |] "create_list"
builder in
        (* map over elements to add to list *)
        let rec append_func typ = match typ with
            A.Int         -> append_int_func
          | A.Char        -> append_char_func
          | A.String      -> append_str_func
          | A.Float       -> append_float_func
          | A.Array(arr)  -> (append_func arr)
          | A.Nah         -> raise (Error "No such thing as nah append function")
          | _             -> raise (Error "list append function not defined for type")
        in
        let appender c = L.build_call (append_func e_type) [| li; (expr builder c) |]
"" builder in
        (List.map appender list); li

      | SDictLiteral(dict_elem_list)        ->
        let dict_type_string_tup  = Str.split (Str.regexp " ") (get_type_string
e_type) in
        let key_type_string       = List.hd dict_type_string_tup in
        let key_type_string_ptr   = expr builder (A.String,
SStringLiteral(key_type_string)) in
        let val_type_string       = List.nth dict_type_string_tup 1 in
        let val_type_string_ptr   = expr builder (A.String,
SStringLiteral(val_type_string)) in
        (* create empty dict llvalue *)
        let dict = L.build_call create_dict_func [| key_type_string_ptr;
val_type_string_ptr |] "create_dict" builder in
        (* takes in a SDictElem, adds its key and value to the dict above*)
        let adder (dict_elem_t, dict_elem) = (match dict_elem with
            | SDictElem(key, value) ->
              (* takes in (typ,sx) and returns a build_call to allocate space for
values in dict*)
              let void_alloc ((z_t, z_x) as z) = (match z_t with
                  | A.Int       -> L.build_call int_alloc_func [| (expr builder z) |]
"int_alloc" builder
                  | A.Char      -> L.build_call char_alloc_func [| (expr builder z) |]
"char_alloc" builder
                  | A.String    -> L.build_call str_alloc_func [| (expr builder z) |]
"str_alloc" builder
                  | A.Array(_)  ->
                    let list_ptr = expr builder z in
                    L.build_bitcast list_ptr (L.pointer_type i8_t) (L.value_name
list_ptr) builder
                  | A.Dictionary(_, _)  ->
                    let dict_ptr = expr builder z in
                    L.build_bitcast dict_ptr (L.pointer_type i8_t) (L.value_name
```

```
dict_ptr) builder
                  | _       -> raise (Error "No alloc function for type"))
              in
              (* call alloc functions for the key and value in the dict element*)
              let key_ll = (void_alloc key) in
              let val_ll = (void_alloc value) in
              (* call add_keyval_func *)
              L.build_call add_keyval_func [| dict; key_ll; val_ll |] "" builder

            | _              -> raise(Error "Dictionary should never add non dict-
elements. How did you get here???")
          ) in

        (* map over list of dict elements to add them to dict *)
        (List.map adder dict_elem_list); dict

      | SDictElem(e1, e2)         -> raise (Error "Dict Elements should never be
encountered. How did you get here???")

      | SId s                     -> L.build_load (lookup s) s builder

      | SBinop ((A.Float,_ ) as e1, op, e2) ->
        let e1' = expr builder e1
        and e2' = expr builder e2 in
        (match op with
           A.Add     -> L.build_fadd
         | A.Sub     -> L.build_fsub
         | A.Mult    -> L.build_fmul
         | A.Div     -> L.build_fdiv
         | A.Equal   -> L.build_fcmp L.Fcmp.Oeq
         | A.Neq     -> L.build_fcmp L.Fcmp.One
         | A.Less    -> L.build_fcmp L.Fcmp.Olt
         | A.Leq     -> L.build_fcmp L.Fcmp.Ole
         | A.Greater -> L.build_fcmp L.Fcmp.Ogt
         | A.Geq     -> L.build_fcmp L.Fcmp.Oge
         | A.Mod     -> L.build_frem
         | A.And | A.Or | A.Has ->
           raise (Error "Invalid expressions for Binary operator between floats,
semant should have stopped this")
        ) e1' e2' "tmp" builder

      | SBinop (e1, op, e2) ->
        let e1' = expr builder e1
        and e2' = expr builder e2 in
        (match op with
           A.Add     -> L.build_add
         | A.Sub     -> L.build_sub
         | A.Mult    -> L.build_mul
         | A.Div     -> L.build_sdiv
         | A.And     -> L.build_and
         | A.Or      -> L.build_or
         | A.Equal   -> L.build_icmp L.Icmp.Eq
```

```
         | A.Neq     -> L.build_icmp L.Icmp.Ne
         | A.Less    -> L.build_icmp L.Icmp.Slt
         | A.Leq     -> L.build_icmp L.Icmp.Sle
         | A.Greater -> L.build_icmp L.Icmp.Sgt
         | A.Geq     -> L.build_icmp L.Icmp.Sge
         | A.Mod     -> L.build_srem
         | A.Has     -> raise (Error "Has should have been desugared. How did you get
here???")
        ) e1' e2' "tmp" builder

      | SUnop(op, ((t, var) as e)) ->
        let e' = expr builder e in
        (match op with
           A.Neg when t = A.Float -> L.build_fneg e' "tmp" builder
         | A.Neg                  -> L.build_neg e' "tmp" builder
         | A.Not                  -> L.build_not e' "tmp" builder
         | A.Incr ->
           let added = L.build_nsw_add e' (L.const_int (ltype_of_typ t) 1)
(L.value_name e') builder in
           let var_name vr = (match vr with
               | SId(s)  -> s
               | _       -> raise (Error "Incr should only be used with variables."))
           in
           L.build_store added (lookup (var_name var)) builder

         | A.Decr ->
           let added = L.build_nsw_add e' (L.const_int (ltype_of_typ t) (-1))
(L.value_name e') builder in
           let var_name vr = (match vr with
               | SId(s)  -> s
               | _       -> raise (Error "Decr should only be used with variables."))
           in
           L.build_store added (lookup (var_name var)) builder
        )

      | SAssign (s, e)            ->
        let e' = expr builder e in
        ignore(L.build_store e' (lookup s) builder); e'
      | SDeconstruct(v, e)        -> raise (Error "Deconstruct should have been
removed")
      | SOpAssign(v, o, e)        ->
        (* compute value to assign *)
        let value = expr builder (A.Nah, SBinop( (A.Nah, SId(v)), o, e)) in
        (* assign result to variable*)
        ignore(L.build_store value (lookup v) builder); value
      | SDecAssign(t, s, e)       ->
        let local_var = L.build_alloca (ltype_of_typ t) s builder in
        Hashtbl.add local_vars s local_var;
        let e' = expr builder e in
        ignore(L.build_store e' (lookup s) builder); e'

      (* typ will be a dictionary or a dict *)
```

```
      | SAccess((typ, _) as e, l) ->
        let index       = expr builder l in
        let li          = expr builder e in
        let rec access_func typ = match typ with
            A.Int         -> access_int_func
          | A.Char        -> access_char_func
          | A.String      -> access_str_func
          | A.Float       -> access_float_func
          | A.Array(arr)  -> (access_func arr)
          | A.Dictionary(key_t, key_v) -> (match key_t with
              | A.Char    -> access_char_key_func
              | A.String  -> access_str_key_func
              | _     -> raise (Error "dictionary access function not defined for key
type")
            )
          | A.Nah         -> raise (Error "No such thing as nah access function")
          | _             -> raise (Error "list access function not defined for type")
        in
        (match typ with
         (* Dictionary access requires pointer bitcasting *)
         | A.Dictionary(_, val_t) ->
           let void_ptr = L.build_call (access_func typ) [| li; index |] "access"
builder in
           (match val_t with
            | A.Int ->
              let int_ptr = L.build_bitcast void_ptr (L.pointer_type (ltype_of_typ
A.Int)) (L.value_name void_ptr) builder in
              L.build_load int_ptr (L.value_name int_ptr) builder
            | A.Char -> raise (Error "val is char")
            | A.Dictionary(_, _) ->
              L.build_bitcast void_ptr (L.pointer_type (find_struct_type "dict"))
(L.value_name void_ptr) builder
            | _       -> raise (Error "idk what this dict val type is chief"))
         (* Array access is a simple function call*)
         | A.Array(_)        -> L.build_call (access_func typ) [| li; index |]
"access" builder
         | _                 -> raise (Error "SAccess on something that isn't a list
or a dictionary not supported"))

      | SAccessAssign(i, idx, e) ->
        let li = expr builder i in
        let index = expr builder idx in
        let value = expr builder e in
        L.build_call assign_int_func [| li; index; value |] "assign_int" builder

      (* USER-EXPOSED BUILT-INS: Must be added to check in decs.ml *)

      (* print *)
      | SCall("print", []) -> let newline = expr builder (String, SStringLiteral(""))
        in L.build_call printf_func [| str_format_str ; newline |] "printf" builder

      | SCall("print", [((p_t, p_v) as params)]) -> (match p_t with
```

```
          | A.Dictionary(_,_) -> raise (Error "Printing dictionary not supported
currently")
          | A.Array(arr) -> (match arr with
              | A.Char   -> L.build_call print_char_list_func [| (expr builder params)
|] "" builder
              | A.String -> L.build_call print_str_list_func [| (expr builder params)
|] "" builder
              | A.Int    -> L.build_call print_int_list_func [| (expr builder params)
|] "" builder
              | _ -> raise (Error "Print not supported for array type"))
          | _ ->
            let print_value = (match p_t with
                | A.Float -> L.build_fpext (expr builder params) double_t "ext"
builder
                | _       -> expr builder params)
            in
            let format_str = (match p_t with
                  A.Int     -> int_format_str
                | A.Char    -> char_format_str
                | A.String  -> str_format_str
                | A.Float   -> float_format_str
                | A.Bool    -> int_format_str
                | _ -> raise (Error "print passed an invalid type"))
            in
            L.build_call printf_func [| format_str ; print_value |] "printf" builder)

      | SCall("print", params) -> raise (Error "print not supported with multiple
params currently")

      (* casts *)
      | SCall("toChar", params) -> expr builder (Cast.to_char params)
      | SCall("toInt", params) -> expr builder (Cast.to_int params)
      | SCall("toFloat", params) -> expr builder (Cast.to_float params)
      | SCall("toBool", params) -> expr builder (Cast.to_bool params)
      | SCall("toString", params) -> expr builder (Cast.to_string params)
      | SCall("toNah", params) -> expr builder (Cast.to_nah params)

      (* pow2 *)
      | SCall ("pow2", [params])    -> let value = expr builder params in
        L.build_call pow2_func [| value |] "pow2" builder

      (* append *)
      | SCall ("append", params)  ->
        let li_e = List.hd params in
        let p_e = List.nth params 1 in
        let li = expr builder li_e in
        let p = expr builder p_e in
        let append_func = (match p_e with
            | (A.Char, _)    -> append_char_func
            | (A.String, _)  -> append_str_func
            | (A.Int, _)     -> append_int_func
            | (A.Float, _)   -> append_float_func
```

```
            | _         -> raise (Error "Append parameter type invalid"))
        in
        L.build_call append_func [| li; p |] "" builder

      (* len *)
      | SCall ("len", params)  ->
        let li = expr builder (List.hd params) in
        L.build_call listlen_func [| li |] "len" builder

      (* contains *)
      | SCall ("contains", params) ->
        let typ = (fst (List.hd params)) in
        let li = expr builder (List.hd params) in
        let p = expr builder (List.nth params 1) in
        let rec contains_func typ = match typ with
            A.Int         -> contains_int_func
          | A.Char        -> contains_char_func
          | A.Float       -> contains_float_func
          | A.String      -> contains_str_func
          | A.Nah         -> raise (Error "No such thing as nah contains function")
          | A.Array(arr)  -> (contains_func arr)
          | A.Dictionary(_,_) -> contains_str_key_func
          | _             -> raise (Error "contains function not defined for type")
        in
        L.build_call (contains_func typ) [| li; p |] "contains" builder

      (* takes in a dict pointer and returns a list pointer of the keys in the dict *)
      | SCall ("keys", [params]) ->
        let typ = fst params in
        (match typ with
         | A.Dictionary(_,_) ->
           let dict = expr builder params in
           L.build_call get_str_keys_func [| dict |] "get_str_keys" builder
         | _                 -> raise (Error "Keys only supported for Dictionaries"))

      (* add is a wrapper over keyval, takes in dict, void * key, void * to val *)
      | SCall ("add", params) ->
        let ds = expr builder (List.hd params) in
        let key = (List.nth params 1) in
        let value = (List.nth params 2) in

        let void_alloc ((z_t, z_x) as z) = (match z_t with
            | A.Int       -> L.build_call int_alloc_func [| (expr builder z) |]
"int_alloc" builder
            | A.Char      -> L.build_call char_alloc_func [| (expr builder z) |]
"char_alloc" builder
            | A.String    -> L.build_call str_alloc_func [| (expr builder z) |]
"str_alloc" builder
            | A.Array(_)  ->
              let list_ptr = expr builder z in
              L.build_bitcast list_ptr (L.pointer_type i8_t) (L.value_name list_ptr)
builder
```

```
            (* cast to void pointer here *)
            | A.Dictionary(_, _)  ->
              let dict_ptr = expr builder z in
              L.build_bitcast dict_ptr (L.pointer_type i8_t) (L.value_name dict_ptr)
builder
            (* cast to void pointer here *)
            | _       -> raise (Error "No alloc function for type"))
        in
        (* cast key and value pointers to void pointers*)
        let key_void_ptr = (void_alloc key) in
        let value_void_ptr = (void_alloc value) in
        L.build_call add_keyval_func [| ds; key_void_ptr; value_void_ptr |] "" builder

      (* SCall for user defined functions *)
      | SCall (f, args)           ->
        let (fdef, fdecl) = try StringMap.find f function_decls
          with Not_found -> raise (Error "User defined function call not found")
        in
        let llargs = List.rev (List.map (expr builder) (List.rev args)) in
        let result = (match fdecl.styp with
              A.Nah -> ""
            | _ -> f ^ "_result") in
        L.build_call fdef (Array.of_list llargs) result builder

      (* clever desugar *)
      | SAttributeCall(e, f, el)  -> expr builder (A.Nah, SCall(f, e::el))
      | SNoexpr                   -> L.const_int i32_t 0
      | _ -> raise (Error "Expression match not implemented")
    in

    let rec stmt builder = function
      | SBlock sl                               -> List.fold_left stmt builder sl
      | SExpr e                                 -> ignore(expr builder e); builder
      | SDec (t, n)                             ->
        let local_var = L.build_alloca (ltype_of_typ t) n builder
        in Hashtbl.add local_vars n local_var; builder
      | SReturn e -> ignore(match fdecl.styp with
          (* Special "return nothing" instr *)
            A.Nah -> L.build_ret_void builder
          (* Build return statement *)
          | _ -> L.build_ret (expr builder e) builder );
        builder

      | SIf (predicate, then_stmt, else_stmt)   ->
        let bool_val = expr builder predicate in
        let merge_bb = L.append_block context "merge" the_function in
        let build_br_merge = L.build_br merge_bb in (* partial function *)

        let then_bb = L.append_block context "then" the_function in
        add_terminal (stmt (L.builder_at_end context then_bb) then_stmt)
          build_br_merge;
```

```
        let else_bb = L.append_block context "else" the_function in
        add_terminal (stmt (L.builder_at_end context else_bb) else_stmt)
          build_br_merge;

        ignore(L.build_cond_br bool_val then_bb else_bb builder);
        L.builder_at_end context merge_bb

      | SWhile (predicate, body, increment) ->
        let rec loop_stmt loop_bb exit_bb builder = (function
              SBlock(sl) -> List.fold_left (fun b s -> loop_stmt loop_bb exit_bb b s)
builder sl
            | SIf (predicate, then_stmt, else_stmt)   ->
              let bool_val = expr builder predicate in
              let merge_bb = L.append_block context "merge" the_function in
              let build_br_merge = L.build_br merge_bb in (* partial function *)

              let then_bb = L.append_block context "then" the_function in
              add_terminal (loop_stmt loop_bb exit_bb (L.builder_at_end context
then_bb) then_stmt)
                build_br_merge;

              let else_bb = L.append_block context "else" the_function in
              add_terminal (loop_stmt loop_bb exit_bb (L.builder_at_end context
else_bb) else_stmt)
                build_br_merge;

              ignore(L.build_cond_br bool_val then_bb else_bb builder);
              L.builder_at_end context merge_bb
            | SSkip _ ->
              let skip_bb = L.append_block context "skip" the_function in
              ignore (L.build_br skip_bb builder);
              let skip_builder = (L.builder_at_end context skip_bb) in
              add_terminal (loop_stmt loop_bb exit_bb skip_builder increment)
(L.build_br loop_bb);
              builder
            | SAbort _ -> ignore(L.build_br exit_bb builder); builder
            | _ as e -> stmt builder e) in

        let pred_bb = L.append_block context "while" the_function
        and merge_bb = L.append_block context "merge" the_function in

        ignore(L.build_br pred_bb builder);
        let body_bb = L.append_block context "while_body" the_function in
        add_terminal (loop_stmt pred_bb merge_bb (L.builder_at_end context body_bb)
body)
          (L.build_br pred_bb);

        let pred_builder = L.builder_at_end context pred_bb in
        let bool_val = expr pred_builder predicate in

        ignore(L.build_cond_br bool_val body_bb merge_bb pred_builder);
        L.builder_at_end context merge_bb
```

```
      | SSkip _     -> raise (Failure "Error: skip occurs outside of loop")
      | SAbort _    -> raise (Failure "Error: abort occurs outside of loop")
      | SPanic expr -> raise (Error "Panic statement not implemented")
      | _           -> raise (Error "Statement match for stmt builder not
implemented")
    in

    (* Build the code for each statement in the function *)
    let builder = stmt builder (SBlock fdecl.sbody) in

    (* Add a return if the last block falls off the end *)
    add_terminal builder (match fdecl.styp with
          A.Nah -> L.build_ret_void
        | A.Float -> L.build_ret (L.const_float float_t 0.0)
        | t -> L.build_ret (L.const_int (ltype_of_typ t) 0))
  in

  (* build all function bodies *)
  let _ = List.map build_function_body functions in

  (* return the LLVM module *)
  the_module
----------
----------
File: src/decs.ml
(*
   This file semantically checks all variable and function declarations in the AST.
   Duplicate variables, nah variable declarations, and function declarations
   with the same parameters throw errors.
   A scoped symbol table mapping variable names to their types is returned, along with
   a mapping of functions to their scoped formal and local variables.
*)

open Ast

module StringMap = Map.Make(String)

(* Data structure that holds scoped variables *)
type scope_table = {
  variables : typ StringMap.t;
  parent : scope_table option;
}

let rec string_of_scope scope =
  let print_bind name ty str = ("(" ^ name ^ ": " ^ (string_of_typ ty) ^ ")\n" ^ str)
in
  let this_scope = StringMap.fold (fun k t s -> print_bind k t s) scope.variables ""
in
  let parent_scope = match scope.parent with
      Some(parent) -> string_of_scope parent
    | None -> ""
  in "{" ^ this_scope ^ "}\n|\nV\n{" ^ parent_scope ^ "}"
```

```
(* Data structure that stores function declarations *)
type func_table = {
  formals : scope_table;
  locals : scope_table;
  ret_typ : typ;
}

(* Names of built-in Viper functions *)
(* User-defined functions cannot have any name in this list *)
let illegal_func_names = ["print"; "len"; "int"; "char"; "float"; "bool"; "string";
"nah"; "pow2"; "append"]

(* Retrieves a type from an ID name *)
let rec toi scope s =
  if StringMap.mem s scope.variables then
    StringMap.find s scope.variables
  else match scope.parent with
      Some(parent) -> toi parent s
    | _ -> raise (Failure ("Variable " ^ s ^ " not found"))

(* Builds a comma-separated string out of a list of parameters *)
(* For example, [int, char, bool] becomes "(int, char, bool)" *)
let rec string_of_params params = match params with
    (typ, _) :: [] -> string_of_typ typ
  | (typ, _) ::  p -> string_of_typ typ ^ ", " ^ string_of_params p
  | _              -> ""

(* Creates a key string used in mapping functions to their declarations *)
(* Strings include function name and a list of parameters to allow overloaded
functions *)
and key_string name params = name ^ " (" ^ string_of_params params ^ ")"

let rec string_of_params_built_in params = match params with
  | typ :: []   -> string_of_typ typ
  | typ :: p    -> string_of_typ typ ^ ", " ^ string_of_params_built_in p
  | _           -> ""

and key_string_built_in_functions name params = name ^ " (" ^
string_of_params_built_in params ^ ")"

(* Checks to see if a variable name is a duplicate *)
let rec is_valid_dec name scope =
  if StringMap.mem name scope.variables then
    raise (Failure ("Error: variable " ^ name ^ " is already defined"))
  else match scope.parent with
      Some(parent) -> is_valid_dec name parent
    | _ -> true

(* Adds a (name, type) pair to a symbol table *)
let add_symbol name ty scope = match ty with
```

```
    Nah -> raise (Failure ("Error: variable " ^ name ^ " declared with type nah"))
  | _  -> if is_valid_dec name scope then {
      variables = StringMap.add name ty scope.variables;
      parent = scope.parent;
    } else scope (* Never enters else clause, but still needed to avoid type error *)

let add_symbol_driver name ty scope = match ty with
    Nah -> raise (Failure ("Error: variable " ^ name ^ " declared with type nah"))
  | _  ->  {
      variables = StringMap.add name ty scope.variables;
      parent = scope.parent;
    }

(* Adds declarations from a bind into a symbol table *)
let get_bind_decs scope bind =
  let ty, name = bind in add_symbol name ty scope

(* Adds declarations from expressions into a symbol table *)
let rec get_expr_decs scope expr =
  match expr with
    Binop(e1, _, e2) ->
    let expr_list = [e1; e2] in List.fold_left get_expr_decs scope expr_list
  | Unop(_, e) -> get_expr_decs scope e
  | Ternop(e1, e2, e3) ->
    let expr_list = [e1; e2; e3] in List.fold_left get_expr_decs scope expr_list
  | OpAssign(_, _, e)
  | Assign(_, e) -> get_expr_decs scope e
  | DecAssign(ty, name, e) ->
    let updated_scope = get_expr_decs scope e in add_symbol name ty updated_scope
  | Deconstruct(b_list, e) ->
    let updated_scope = List.fold_left get_bind_decs scope b_list in get_expr_decs
updated_scope e
  | Access(arr, index) ->
    let expr_list = [arr; index] in List.fold_left get_expr_decs scope expr_list
  | AccessAssign(e1, e2, e3) ->
    let expr_list = [e1; e2; e3] in List.fold_left get_expr_decs scope expr_list
  | Call(_, expr_list) -> List.fold_left get_expr_decs scope expr_list
  | AttributeCall(e1, _, e_list) ->
    let expr_list = e1 :: e_list in List.fold_left (get_expr_decs) scope expr_list
  | _ -> scope

(* Adds declarations from statements into a symbol table *)
let rec get_stmt_decs scope stmt =
  let new_scope = {
    variables = StringMap.empty;
    parent = Some(scope);
  } in match stmt with
    Block(s_list) ->
    let _ = List.fold_left get_stmt_decs new_scope s_list in scope
  | Expr(e) -> get_expr_decs scope e
  | Dec(ty, name) -> add_symbol name ty scope
  | If(cond, then_s, else_s) ->
```

```
    let cond_scope = get_expr_decs new_scope cond in
    let _ = (get_stmt_decs cond_scope then_s, get_stmt_decs cond_scope else_s) in
scope
  | For(e1, e2, e3, s) ->
    let expr_list = [e1; e2; e3] in
    let for_scope = List.fold_left get_expr_decs new_scope expr_list in
    let _ = get_stmt_decs for_scope s in scope
  | DecForIter(ty, name, e, s) ->
    let iter_scope = add_symbol name ty new_scope in
    let for_scope = get_expr_decs iter_scope e in
    let _ = get_stmt_decs for_scope s in scope
  | While(e, s, _) ->
    let while_scope = get_expr_decs new_scope e in
    let _ = get_stmt_decs while_scope s in scope
  | PretendBlock(s_list) ->
    let _ = List.fold_left get_stmt_decs new_scope s_list in scope
  | _ -> scope

(* Driver for getting declarations from a list of statements *)
let get_vars scope s_list = List.fold_left get_stmt_decs scope s_list

(* Ensure that no functions are duplicated *)
(* Functions are invalid if they share the name with a built-in Viper function, or it
   two user-defined functions have the same name and list of formal arguments *)
let valid_func_name fd map =
  let rec unused_name name illegals = match illegals with
      [] -> ()
    | illegal_name :: _ when name = illegal_name ->
      raise (Failure ("Error: illegal function name " ^ name))
    | _ :: tail -> unused_name name tail
  in let _ = unused_name fd.fname illegal_func_names in
  let key = key_string fd.fname fd.formals in
  if StringMap.mem key map then
    raise (Failure("Error: function " ^ fd.fname ^ " is already defined with formal
arguments (" ^
                   (string_of_params fd.formals) ^ ")"))
  else key

(* Builds a function table for a function declaration *)
let build_func_table global_scope (fd : func_decl) map =
  let key = valid_func_name fd map in
  let formals_scope = List.fold_left get_bind_decs {
      variables = StringMap.empty;
      parent = None;
    } fd.formals in
  let updated_scope = {
    variables = formals_scope.variables;
    parent = Some(global_scope); } in
  let locals_scope = get_vars {
      variables = StringMap.empty;
      parent = Some(updated_scope);
    } fd.body in StringMap.add key {
```

```
    formals = updated_scope;
    locals = locals_scope;
    ret_typ = fd.typ;
  } map

(* Driver for getting declarations *)
let get_decs (s_list, f_list) =
  let globals = get_vars {
      variables = StringMap.empty;
      parent = None;
    } (List.rev s_list) in

  (* Collect declarations for Viper's built-in functions *)
  let built_in_funcs =
    let build_built_in_func_table map (name, param_typ, typ) =
      let args = List.fold_left
          (fun m f -> let param_name = ("p" ^ string_of_int (StringMap.cardinal
m.variables)) in add_symbol param_name f m) {
          variables = StringMap.empty;
          parent = None;
        } param_typ in
      let key = (key_string_built_in_functions name param_typ)
      in StringMap.add key {
        formals = args;
        locals = {
          variables = StringMap.empty;
          parent = None;
        };
        ret_typ = typ;
      } map
    in List.fold_left build_built_in_func_table StringMap.empty [
      ("print", [], Nah);
      ("print", [Int], Nah);
      ("print", [String], Nah);
      ("print", [Char], Nah);
      ("print", [Bool], Nah);
      ("print", [Float], Nah);
      ("print", [Array(Char)], Nah);
      ("print", [Array(String)], Nah);
      ("print", [Array(Int)], Nah);

      ("len", [Array(Int)], Int);
      ("len", [Array(Float)], Int);
      ("len", [Array(Bool)], Int);
      ("len", [Array(String)], Int);
      ("len", [Array(Char)], Int);

      ("append", [Array(Char); Char], Nah);
      ("append", [Array(String); String], Nah);
      ("append", [Array(Int); Int], Nah);
      ("append", [Array(Float); Float], Nah);
```

```
      ("contains", [Array(Char); Char], Int);
      ("contains", [Array(String); String], Int);
      ("contains", [Array(Float); Float], Int);
      ("contains", [Array(Int); Int], Int);
      ("contains", [Dictionary(String, Int); String], Int);

      ("add", [String; Int], Nah);
      ("add", [Dictionary(String, Int); String; Int], Nah);
      ("add", [Dictionary(Char, Int); Char; Int], Nah);

      ("keys", [Dictionary(String, Int)], Array(String));

      ("toInt", [Float], Int);
      ("toInt", [String], Int);
      ("toInt", [Char], Int);
      ("toInt", [Bool], Int);
      ("toInt", [Int], Int);
      ("toChar", [Int], Char);
      ("toChar", [String], Char);
      ("toChar", [Char], Char);
      ("toFloat", [Int], Float);
      ("toFloat", [String], Float);
      ("toFloat", [Char], Float);
      ("toFloat", [Float], Float);
      ("toBool", [Int], Bool);
      ("toBool", [String], Bool);
      ("toBool", [Char], Bool);
      ("toBool", [Bool], Bool);
      ("toString", [Int], String);
      ("toString", [Float], String);
      ("toString", [Bool], String);
      ("toString", [String], String);
      ("toNah", [Int], Nah);
      ("toNah", [String], Nah);
      ("toNah", [Char], Nah);
      ("toNah", [Float], Nah);
      ("toNah", [Bool], Nah);
      ("pow2", [Float], Float);
      ("pow2", [Int], Float);
    ]
  in

  (* Collects function tables for a list of function declarations *)
  let get_funcs f_list =
    List.fold_left (fun m f -> build_func_table globals f m) built_in_funcs f_list

  in (globals, get_funcs f_list)

----------
----------
File: src/library.c
/*
```

```
Viper's C standard library
*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>

struct list
{
    void **data;
    int size;
    int mem;
    char *type;
};

struct dict_elem
{
    void *key;
    void *val;
};

struct dict
{
    struct list *pairs;
    char *key_type;
    char *val_type;
};

struct list *create_list(char *type)
{
    struct list *inlist = malloc(sizeof(struct list));
    char *toadd = malloc(strlen(type));
    strcpy(toadd, type);
    inlist->type = toadd;
    inlist->size = 0;
    inlist->mem = 64;
    inlist->data = malloc(64);
    return inlist;
}

void realloc_check(struct list *inlist)
{
    if (inlist->mem <= (inlist->size * 8))
    {
        void **newdata = realloc(inlist->data, inlist->mem + 64);
        if (newdata == NULL)
        {
            printf("Failure to reallocate data");
            return;
        }
        inlist->data = newdata;
```

```
    }
}

void append_pair(struct list *inlist, struct dict_elem *pair)
{
    if (strcmp(inlist->type, "dict"))
    {
        printf("Can only append %s, not dict\n", inlist->type);
        return;
    }

    realloc_check(inlist);
    inlist->data[(inlist->size)] = pair;
    inlist->size = inlist->size + 1;
}

void append_str(struct list *inlist, char *str)
{
    if (strcmp(inlist->type, "string"))
    {
        printf("Can only append %s, not string\n", inlist->type);
        return;
    }

    realloc_check(inlist);
    char *toadd = malloc(strlen(str));
    strcpy(toadd, str);
    inlist->data[(inlist->size)] = toadd;
    inlist->size = inlist->size + 1;
}

void append_char(struct list *inlist, char chr)
{
    if (strcmp(inlist->type, "char"))
    {
        printf("Can only append %s, not char\n", inlist->type);
        return;
    }

    realloc_check(inlist);
    char *toadd = malloc(1);
    *toadd = chr;
    inlist->data[(inlist->size)] = toadd;
    inlist->size = inlist->size + 1;
}

void append_int(struct list *inlist, int num)
{
    if (strcmp(inlist->type, "int"))
    {
        printf("Can only append %s, not int\n", inlist->type);
        return;
```

```
    }

    realloc_check(inlist);
    int *toadd = malloc(4);
    *toadd = num;
    inlist->data[(inlist->size)] = toadd;
    inlist->size = inlist->size + 1;
}

void append_float(struct list *inlist, float flt)
{
    if (strcmp(inlist->type, "float"))
    {
        printf("Can only append %s, not float\n", inlist->type);
        return;
    }

    realloc_check(inlist);
    float *toadd = malloc(4);
    *toadd = flt;
    inlist->data[(inlist->size)] = toadd;
    inlist->size = inlist->size + 1;
}

void append_list(struct list *inlist, struct list *outlist)
{
    if (strcmp(inlist->type, "list"))
    {
        printf("Can only append %s, not list\n", inlist->type);
        return;
    }

    realloc_check(inlist);
    inlist->data[(inlist->size)] = outlist;
    inlist->size = inlist->size + 1;
}

void *access(struct list *inlist, int index)
{
    return inlist->data[index];
}

int access_int(struct list *inlist, int index)
{
    if (strcmp(inlist->type, "int"))
    {
        printf("Can only access %s, not int\n", inlist->type);
        return 0;
    }

    int *num = (int *)access(inlist, index);
    if (num == NULL)
```

```
    {
        printf("Illegal index accessed\n");
        return 0;
    }
    return *num;
}

char access_char(struct list *inlist, int index)
{
    if (strcmp(inlist->type, "char"))
    {
        printf("Can only access %s, not char\n", inlist->type);
        return 0;
    }

    char *chr = (char *)access(inlist, index);
    if (chr == NULL)
    {
        printf("Illegal index accessed\n");
        return 0;
    }
    return *chr;
}

float access_float(struct list *inlist, int index)
{
    if (strcmp(inlist->type, "float"))
    {
        printf("Can only access %s, not float\n", inlist->type);
        return 0;
    }

    float *flt = (float *)access(inlist, index);
    if (flt == NULL)
    {
        printf("Illegal index accessed\n");
        return 0;
    }
    return *flt;
}

char *access_str(struct list *inlist, int index)
{
    if (strcmp(inlist->type, "string"))
    {
        printf("Can only access %s, not string\n", inlist->type);
        return 0;
    }

    char *str = (char *)access(inlist, index);
    if (str == NULL)
    {
```

```
        printf("Illegal index accessed\n");
        return NULL;
    }
    return str;
}

int contains_int(struct list *inlist, int tocheck)
{
    if (strcmp(inlist->type, "int"))
    {
        printf("Can only check membership for %s, not int\n", inlist->type);
        return 0;
    }

    int i;
    for (i = 0; i < inlist->size; i++)
    {
        int *num = (int *)access(inlist, i);
        if (*num == tocheck)
        {
            return 1;
        }
    }
    return 0;
}

int contains_char(struct list *inlist, char tocheck)
{
    if (strcmp(inlist->type, "char"))
    {
        printf("Can only check membership for %s, not char\n", inlist->type);
        return 0;
    }

    int i;
    for (i = 0; i < inlist->size; i++)
    {
        char *chr = (char *)access(inlist, i);
        if (*chr == tocheck)
        {
            return 1;
        }
    }
    return 0;
}

int contains_float(struct list *inlist, float tocheck)
{
    if (strcmp(inlist->type, "float"))
    {
        printf("Can only check membership for %s, not float\n", inlist->type);
        return 0;
```

```
    }

    int i;
    for (i = 0; i < inlist->size; i++)
    {
        float *flt = (float *)access(inlist, i);
        if (*flt == tocheck)
        {
            return 1;
        }
    }
    return 0;
}

int contains_str(struct list *inlist, char *tocheck)
{
    if (strcmp(inlist->type, "string"))
    {
        printf("Can only check membership for %s, not string\n", inlist->type);
        return 0;
    }

    int i;
    for (i = 0; i < inlist->size; i++)
    {
        char *str = (char *)access(inlist, i);
        if (!strcmp(str, tocheck))
        {
            return 1;
        }
    }
    return 0;
}

struct list *access_list(struct list *inlist, int index)
{
    if (strcmp(inlist->type, "list"))
    {
        printf("Can only access %s, not list\n", inlist->type);
        return 0;
    }

    struct list *lst = (struct list *)access(inlist, index);
    if (lst == NULL)
    {
        printf("Illegal index accessed\n");
        return NULL;
    }
    return lst;
}

struct dict_elem *access_pair(struct list *inlist, int index)
```

```
{
    if (strcmp(inlist->type, "dict"))
    {
        printf("Can only access %s, not dict\n", inlist->type);
        return 0;
    }

    struct dict_elem *lst = (struct dict_elem *)access(inlist, index);
    if (lst == NULL)
    {
        printf("Illegal index accessed\n");
        return NULL;
    }
    return lst;
}

int assign_int(struct list *inlist, int index, int toAssign)
{
    if (strcmp(inlist->type, "int"))
    {
        printf("Can only access %s, not int\n", inlist->type);
        return 0;
    }

    if (index >= inlist->size || index < 0)
    {
        printf("Illegal index accessed\n");
        return 0;
    }

    *((int *)inlist->data[index]) = toAssign;

    return toAssign;
}

char assign_char(struct list *inlist, int index, char toAssign)
{
    if (strcmp(inlist->type, "char"))
    {
        printf("Can only access %s, not char\n", inlist->type);
        return 0;
    }

    if (index >= inlist->size || index < 0)
    {
        printf("Illegal index accessed\n");
        return 0;
    }

    *((char *)inlist->data[index]) = toAssign;

    return toAssign;
```

```
}

float assign_float(struct list *inlist, int index, float toAssign)
{
    if (strcmp(inlist->type, "float"))
    {
        printf("Can only access %s, not float\n", inlist->type);
        return 0;
    }

    if (index >= inlist->size || index < 0)
    {
        printf("Illegal index accessed\n");
        return 0;
    }

    *((float *)inlist->data[index]) = toAssign;

    return toAssign;
}

char *assign_str(struct list *inlist, int index, char *toAssign)
{
    if (strcmp(inlist->type, "string"))
    {
        printf("Can only access %s, not string\n", inlist->type);
        return 0;
    }

    if (index >= inlist->size || index < 0)
    {
        printf("Illegal index accessed\n");
        return 0;
    }

    inlist->data[index] = toAssign;

    return toAssign;
}

struct list *assign_list(struct list *inlist, int index, struct list *toAssign)
{
    if (strcmp(inlist->type, "list"))
    {
        printf("Can only access %s, not list\n", inlist->type);
        return 0;
    }

    if (index >= inlist->size || index < 0)
    {
        printf("Illegal index accessed\n");
        return 0;
```

```
    }

    inlist->data[index] = toAssign;

    return toAssign;
}

char *get_type(struct list *inlist)
{
    return inlist->type;
}

int listlen(struct list *inlist)
{
    return inlist->size;
}

struct dict *create_dict(char *ktype, char *vtype)
{
    struct dict *indict = malloc(sizeof(struct dict));
    char *ktoadd = malloc(strlen(ktype));
    strcpy(ktoadd, ktype);
    char *vtoadd = malloc(strlen(vtype));
    strcpy(vtoadd, vtype);
    indict->key_type = ktoadd;
    indict->val_type = vtoadd;
    indict->pairs = create_list("dict");
    return indict;
}

void *int_alloc_zone(int input)
{
    int *toadd = malloc(4);
    *toadd = input;
    return (void *)toadd;
}

void *char_alloc_zone(char input)
{
    char *toadd = malloc(1);
    *toadd = input;
    return (void *)toadd;
}

void *float_alloc_zone(float input)
{
    float *toadd = malloc(4);
    *toadd = input;
    return (void *)toadd;
}

void *str_alloc_zone(char *input)
```

```
{
    char *toadd = malloc(strlen(input));
    strcpy(toadd, input);
    return (void *)toadd;
}

void add_keyval(struct dict *indict, void *key, void *val)
{
    struct dict_elem *pair = malloc(sizeof(struct dict_elem));
    pair->key = key;
    pair->val = val;
    append_pair(indict->pairs, pair);
}

void *access_str_key(struct dict *indict, char *key)
{
    void *toret = NULL;
    for (int i = 0; i < indict->pairs->size; i++)
    {
        if (!strcmp((char *)access_pair(indict->pairs, i)->key, key))
        {
            toret = access_pair(indict->pairs, i)->val;
        }
    }
    return toret;
}

void *access_char_key(struct dict *indict, char key)
{
    void *toret = NULL;
    for (int i = 0; i < indict->pairs->size; i++)
    {
        if (*((char *)access_pair(indict->pairs, i)->key) == key)
        {
            toret = access_pair(indict->pairs, i)->val;
        }
    }
    return toret;
}

void print_char_list(struct list *inlist)
{
    if (strcmp(inlist->type, "char"))
    {
        printf("Can only access %s, not char\n", inlist->type);
    }

    printf("[");

    for (int i = 0; i < inlist->size; i++)
    {
        printf("\'%c\'", access_char(inlist, i));
```

```
        if (i < inlist->size - 1)
        {
            printf(", ");
        }
    }

    printf("]\n");
}

void print_int_list(struct list *inlist)
{
    if (strcmp(inlist->type, "int"))
    {
        printf("Can only access %s, not int\n", inlist->type);
    }

    printf("[");

    for (int i = 0; i < inlist->size; i++)
    {
        printf("%d", access_int(inlist, i));

        if (i < inlist->size - 1)
        {
            printf(", ");
        }
    }

    printf("]\n");
}

void print_str_list(struct list *inlist)
{
    if (strcmp(inlist->type, "string"))
    {
        printf("Can only access %s, not string\n", inlist->type);
    }

    printf("[");

    for (int i = 0; i < inlist->size; i++)
    {
        printf("\"%s\"", access_str(inlist, i));

        if (i < inlist->size - 1)
        {
            printf(", ");
        }
    }

    printf("]\n");
```

```
}

int contains_str_key(struct dict *indict, char *key)
{
    for (int i = 0; i < indict->pairs->size; i++)
    {
        if (!strcmp((char *)access_pair(indict->pairs, i)->key, key))
        {
            return 1;
        }
    }
    return 0;
}

int contains_char_key(struct dict *indict, char key)
{
    for (int i = 0; i < indict->pairs->size; i++)
    {
        if (*((char *)access_pair(indict->pairs, i)->key) != key)
        {
            return 1;
        }
    }
    return 0;
}

struct list *get_char_keys(struct dict *indict)
{
    if (strcmp(indict->key_type, "char"))
    {
        printf("Can only access %s, not char\n", indict->key_type);
        return 0;
    }

    struct list *keys = create_list("char");
    for (int i = 0; i < indict->pairs->size; i++)
    {
        char toadd = *((char *)access_pair(indict->pairs, i)->key);
        if (!contains_char(keys, toadd)){
            append_char(keys, toadd);
        }
    }
    return keys;
}

struct list *get_str_keys(struct dict *indict)
{
    if (strcmp(indict->key_type, "string"))
    {
        printf("Can only access %s, not string\n", indict->key_type);
        return 0;
    }
```

```
    struct list *keys = create_list("string");
    for (int i = 0; i < indict->pairs->size; i++)
    {
        char *toadd = (char *)access_pair(indict->pairs, i)->key;
        if (!contains_str(keys, toadd)){
            append_str(keys, toadd);
        }
    }
    return keys;
}

void remove_str_key(struct dict *indict, char *key)
{
    int index_to_remove = -1;
    for (int i = 0; i < indict->pairs->size; i++)
    {
        if (!strcmp((char *)access_pair(indict->pairs, i)->key, key))
        {
            access_pair(indict->pairs, i)->key = NULL;
            index_to_remove = i;
        }
    }

    for (int i = index_to_remove; i < indict->pairs->size - 1; i++)
    {
        access_pair(indict->pairs, i)->key = access_pair(indict->pairs, i + 1)->key;
        access_pair(indict->pairs, i)->val = access_pair(indict->pairs, i + 1)->val;
    }

    indict->pairs->size -= 1;
}

void remove_char_key(struct dict *indict, char key)
{
    int index_to_remove = -1;
    for (int i = 0; i < indict->pairs->size; i++)
    {
        if (*((char *)access_pair(indict->pairs, i)->key) != key)
        {
            access_pair(indict->pairs, i)->key = NULL;
            index_to_remove = i;
        }
    }

    for (int i = index_to_remove; i < indict->pairs->size - 1; i++)
    {
        access_pair(indict->pairs, i)->key = access_pair(indict->pairs, i + 1)->key;
        access_pair(indict->pairs, i)->val = access_pair(indict->pairs, i + 1)->val;
    }

    indict->pairs->size -= 1;
```

```
}

int void_to_int(void *v_ptr)
{
    return *((int *)v_ptr);
}

float pow2(float base)
{
    return pow(base, 2);
}

int imax(int first, int second)
{
    return (int)fmax(first, second);
}

char cmax(char first, char second)
{
    return (char)fmax(first, second);
}

int imin(int first, int second)
{
    return (int)fmin(first, second);
}

char cmin(char first, char second)
{
    return (char)fmin(first, second);
}

float ptrunc(float input, int decs)
{
    return floor(pow(10, decs) * input) / pow(10, decs);
}

// TODO: Remove these
void testy()
{
    int a = 0;
    a++;
    a--;
}

void floaty()
{
}

void listy()
{
    struct list *chrlist = create_list("char");
```

```
    append_char(chrlist, 'a');
    append_char(chrlist, 'b');
    append_char(chrlist, 'c');
    access_char(chrlist, 0);
    // print_char_list(chrlist);
}

void dicty()
{
    struct dict *chardict = create_dict("char", "int");
    // void* test1 = char_alloc_zone('A');
    // void* test2 = int_alloc_zone(0);
    add_keyval(chardict, char_alloc_zone('A'), int_alloc_zone(3));
    // printf("access_char_key(chrdict, 'A'): %d\n",
void_to_int(access_char_key(testdict, 'A')));
    // printf("access_char_key(chrdict, 'A'): %d\n", *((int *)
(access_char_key(testdict, 'A'))));

    struct dict *testdict = create_dict("char", "dict");
    add_keyval(testdict, char_alloc_zone('Z'), (void *)chardict);
    struct dict *char_dict_ptr = (struct dict *)access_char_key(testdict, 'Z');
    printf("access_char_key(chrdict, 'A'): %d\n", *((int
*)access_char_key(char_dict_ptr, 'A')));
}

#ifdef BUILD_TEST
int main(void)
{
    //dicty();
    printf("sqrt(100) = %f\n", sqrt(100));
    printf("sqrt(100.7) = %f\n", sqrt(100.7));
    printf("pow(100, 3) = %f\n", pow(100, 3));
    printf("pow(1.7, 12) = %f\n", pow(1.7, 12));
    printf("pow2(3) = %f\n", pow2(3));
    printf("floor(2.4) = %f\n", floor(2.4));
    printf("ceil(2.4) = %f\n", ceil(2.4));
    printf("round(2.4) = %f\n", round(2.4));
    printf("round(2.4) = %f\n", round(2.6));
    printf("imax(2, 7) = %d\n", imax(2, 7));
    printf("fmax(2.6, 3.5) = %f\n", fmax(2.6, 3.5));
    printf("cmax('k', 'm') = %c\n", cmax('k', 'm'));
    printf("trunc(5.121212) = %f\n", trunc(5.121212));
    printf("ptrunc(5.121212, 3) = %f\n", ptrunc(5.121212, 3));

    struct list *charlist = create_list("char");
    append_char(charlist, 'a');
    append_char(charlist, 'b');
    append_char(charlist, 'c');

    struct list *mylist = create_list("int");
    append_int(mylist, 0);
    append_str(mylist, "yooo");
```

```
    append_int(mylist, 1);
    append_int(mylist, 2);
    append_int(mylist, 3);
    append_int(mylist, 4);
    append_int(mylist, 5);
    append_int(mylist, 6);
    append_int(mylist, 7);
    append_int(mylist, 8);
    append_float(mylist, 4.5);
    append_char(mylist, 'c');
    struct list *otherlist = create_list("list");
    append_int(otherlist, 1);
    append_list(otherlist, mylist);

    printf("Type: %s\n", get_type(mylist));
    printf("Type: %s\n", get_type(otherlist));
    printf("Length: %d\n", listlen(mylist));
    printf("Length: %d\n", listlen(otherlist));

    printf("@ index 0: %d\n", access_int(mylist, 0));
    printf("@ index 5: %d\n", access_int(mylist, 5));
    printf("@ Illegal Index: %d\n", access_int(mylist, 1000));

    struct list *accessed = access_list(otherlist, 0);
    printf("@ index 5: %d\n", access_int(accessed, 5));

    printf("contains_int(mylist, 5): %d\n", contains_int(mylist, 5));
    printf("contains_int(mylist, 10): %d\n", contains_int(mylist, 10));

    int test = assign_int(mylist, 1, 7);
    printf("Test assign return val: %d\n", test);
    printf("@ index 1: %d\n", access_int(mylist, 1));
    printf("contains_int(mylist, 1): %d\n", contains_int(mylist, 1));

    struct list *evillist = create_list("int");
    append_int(evillist, 100);
    append_int(evillist, 200);
    append_int(evillist, 300);
    struct list *testo = assign_list(otherlist, 0, evillist);
    printf("Test assign return list: %d\n", access_int(evillist, 0));
    struct list *sameto = access_list(otherlist, 0);
    printf("Test accessed list: %d\n", access_int(evillist, 1));

    // struct dict *testdict = create_dict("string", "int");
    // //void* test1 = str_alloc_zone("yo");
    //void* test2 = int_alloc_zone(1);
    //add_keyval(testdict, str_alloc_zone("yo"), int_alloc_zone(1));
    //add_keyval(testdict, str_alloc_zone("gabba"), int_alloc_zone(12));
    //printf("access_str_key(testdict, \"yo\"): %d\n", *((int
*)access_str_key(testdict, "yo")));
    //printf("access_str_key(testdict, \"yo\"): %d\n", *((int
*)access_str_key(testdict, "gabba")));
```

```
    struct dict *chrdict = create_dict("char", "int");
    add_keyval(chrdict, char_alloc_zone('A'), int_alloc_zone(7));
    add_keyval(chrdict, char_alloc_zone('B'), int_alloc_zone(8));
    struct list *chrkeys = get_char_keys(chrdict);
    print_char_list(chrkeys);
    printf("access_char_key(chrdict, 'A'): %d\n", *((int *)access_char_key(chrdict,
'A')));
    printf("access_char_key(chrdict, 'B'): %d\n", *((int *)access_char_key(chrdict,
'B')));
    printf("contains_char_key(chrdict, 'A'): %d\n", contains_char_key(chrdict, 'A'));

    struct dict *strdict = create_dict("string", "int");
    add_keyval(strdict, str_alloc_zone("AAA"), int_alloc_zone(7));
    add_keyval(strdict, str_alloc_zone("BBB"), int_alloc_zone(8));
    add_keyval(strdict, str_alloc_zone("CCC"), int_alloc_zone(8));
    struct list *strkeys = get_str_keys(strdict);
    print_str_list(strkeys);
    printf("contains_str_key(strdict, 'CCC'): %d\n", contains_str_key(strdict,
"CCC"));
    remove_str_key(strdict, "CCC");
    struct list *strkeys2 = get_str_keys(strdict);
    print_str_list(strkeys2);
    printf("contains_str_key(strdict, 'CCC'): %d\n", contains_str_key(strdict,
"CCC"));

    // struct list *intlist = create_list("int");
    // append_int(intlist, 1);
    // append_int(intlist, 2);
    // append_int(intlist, 3);
    // print_int_list(intlist);

    // struct list *strlist = create_list("string");
    // append_str(strlist, "aaa");
    // append_str(strlist, "bbb");
    // append_str(strlist, "ccc");
    // print_str_list(strlist);
}
#endif

----------
----------
File: src/cast.ml
(*
This file handles Viper's type casting, defining what types can and cannot be cast to
another.
*)

open Ast
open Sast

let verify_params params func =
```

```
  if List.length params != 1 then
    raise (Failure ("Error: " ^ func ^ "() requires one argument"))
  else List.hd params

let to_char params =
  match verify_params params "char" with
      (_, (SCharacterLiteral(_) as c)) -> (Char, c)
    | (_, SIntegerLiteral(i)) when i > -1 && i < 256 -> (Char,
SCharacterLiteral(char_of_int i))
    | (_, SIntegerLiteral(i)) -> raise (Failure("Error: integer " ^ string_of_int i ^
" cannot be cast to char,
        it must have a value between 0 and 255"))
    | (_, SStringLiteral("")) -> raise (Failure("Error: empty string cannot be cast to
char"))
    | (_, SStringLiteral(s)) -> (Char, SCharacterLiteral(String.get s 0))
    | (typ, _) -> raise (Failure ("Error: type " ^ string_of_typ typ ^ " cannot be
cast to char"))

let to_int params =
  match verify_params params "int" with
      (_, (SIntegerLiteral(_) as i)) -> (Int, i)
    | (_, SCharacterLiteral(c)) -> (Int, SIntegerLiteral(int_of_char c))
    | (_, SFloatLiteral(f)) -> (Int, SIntegerLiteral(int_of_float f))
    | (_, SBoolLiteral(true)) -> (Int, SIntegerLiteral(1))
    | (_, SBoolLiteral(false)) -> (Int, SIntegerLiteral(0))
    | (_, SStringLiteral(s)) -> (
        try (Int, SIntegerLiteral(int_of_string s))
        with Failure _ -> raise (Failure ("Error: string \"" ^ s ^ "\" cannot be cast
to int")))
    | (typ, _) -> raise (Failure ("Error: type " ^ string_of_typ typ ^ " cannot be
cast to int"))

let to_float params =
  match verify_params params "float" with
      (_, (SFloatLiteral(_) as f)) -> (Float, f)
    | (_, SIntegerLiteral(i)) -> (Float, SFloatLiteral(float_of_int i))
    | (_, SCharacterLiteral(c)) -> (Float, SFloatLiteral(float_of_int (int_of_char
c)))
    | (_, SStringLiteral(s)) -> (
      try (Int, SFloatLiteral(float_of_string s))
      with Failure _ -> raise (Failure ("Error: string \"" ^ s ^ "\" cannot be cast to
float")))
    | (typ, _) -> raise (Failure ("Error: type " ^ string_of_typ typ ^ " cannot be
cast to float"))

let to_bool params =
  match verify_params params "bool" with
      (_, (SBoolLiteral(_) as b)) -> (Bool, b)
    | (_, SCharacterLiteral(c)) when c = '\x00' -> (Bool, SBoolLiteral(false))
    | (_, SIntegerLiteral(i)) when i = 0 -> (Bool, SBoolLiteral(false))
    | (_, SFloatLiteral(f)) when f = 0.0 -> (Bool, SBoolLiteral(false))
    | (_, SStringLiteral(s)) when s = "" -> (Bool, SBoolLiteral(false))
```

```
    | (Nah, _) -> (Bool, SBoolLiteral(false))
    | (_, SCharacterLiteral(_))
    | (_, SIntegerLiteral(_))
    | (_, SFloatLiteral(_))
    | (_, SStringLiteral(_)) -> (Bool, SBoolLiteral(true))
    | (typ, _) -> raise (Failure ("Error: type " ^ string_of_typ typ ^ " cannot be
cast to bool"))

let to_string params =
  match verify_params params "string" with
      (_, (SStringLiteral(_) as s)) -> (String, s)
    | (_, SCharacterLiteral(c)) -> (String, SStringLiteral(String.make 1 c))
    | (_, SIntegerLiteral(i)) -> (String, SStringLiteral(string_of_int i))
    | (_, SFloatLiteral(f)) -> (String, SStringLiteral(string_of_float f))
    | (_, SBoolLiteral(true)) -> (String, SStringLiteral("true"))
    | (_, SBoolLiteral(false)) -> (String, SStringLiteral("false"))
    | (Nah, _) -> (String, SStringLiteral("nah"))
    | (typ, _) -> raise (Failure ("Error: type " ^ string_of_typ typ ^ " cannot be
cast to string"))

let to_nah params =
  match verify_params params "nah" with
      _ -> (Nah, SNoexpr)

----------
----------
File: src/parser.mly
%{
open Ast
%}

%token SEMI LPAREN RPAREN LBRACE RBRACE COMMA ARROPEN ARRCLOSE DOT
%token PLUS MINUS TIMES DIVIDE ASSIGN NOT PLUS_ASSIGN MINUS_ASSIGN TIMES_ASSIGN
DIVIDE_ASSIGN MODULO HAS QUESTION COLON
%token EQ NEQ LT LEQ GT GEQ TRUE FALSE AND OR MATCH BAR
%token RETURN IF ELSE FOR WHILE INT CHAR BOOL FLOAT STRING NAH FUNC IN ARROW PANIC
%token SKIP ABORT
%token <int> INTLIT
%token <char> CHARLIT
%token <float> FLOATLIT
%token <string> STRLIT
%token <string> ID
%token EOF

%nonassoc NOELSE
%nonassoc ELSE
%right ASSIGN
%left QUESTION COLON MATCH
%left BAR
%left OR
%left AND
%left EQ NEQ
```

```
%left LT GT LEQ GEQ
%right HAS
%left PLUS MINUS PLUS_ASSIGN MINUS_ASSIGN
%left TIMES DIVIDE MODULO TIMES_ASSIGN DIVIDE_ASSIGN
%nonassoc INCR DECR
%right NOT NEG
%left ARROPEN ARRCLOSE DOT

%start program
%type <Ast.program> program

%%

program:
  decls EOF { $1 }

decls:
   /* nothing */ { [], [] }
 | decls fdecl { fst $1, ($2 :: snd $1) }
 | decls stmt { ($2 :: fst $1), snd $1 }

fdecl:
   typ ID LPAREN formals_opt RPAREN LBRACE stmt_list RBRACE
     { { typ = $1;
       fname = $2;
       formals = $4;
     body = List.rev $7;
     autoreturn = false } }
    | typ ID LPAREN formals_opt RPAREN ARROW stmt
     { { typ = $1;
     fname = $2;
     formals = $4;
     body = [$7];
     autoreturn = true } }

formals_opt:
    /* nothing */ { [] }
  | formal_list   { List.rev $1 }

formal_list:
    typ ID                   { [($1,$2)] }
  | formal_list COMMA typ ID { ($3,$4) :: $1 }

typ:
    INT { Int }
  | BOOL { Bool }
  | NAH { Nah }
  | CHAR { Char }
  | FLOAT { Float }
  | STRING { String }
  | typ FUNC { Function($1) }
  | typ ARROPEN ARRCLOSE { Array($1) }
```

```
  | LPAREN type_list RPAREN { Group($2) }
  | ARROPEN typ COLON typ ARRCLOSE { Dictionary($2, $4) }

type_list:
    typ            { [$1] }
  | typ COMMA type_list { $1 :: $3 }

stmt_list:
    /* nothing */  { [] }
  | stmt_list stmt { $2 :: $1 }

stmt:
    expr SEMI { Expr $1 }
  | typ ID SEMI { Dec($1, $2) }
  | RETURN SEMI { Return Noexpr }
  | RETURN expr SEMI { Return $2 }
  | SKIP SEMI { Skip Noexpr }
  | ABORT SEMI { Abort Noexpr }
  | PANIC expr SEMI { Panic $2 }
  | LBRACE stmt_list RBRACE { Block(List.rev $2) }
  | IF LPAREN expr RPAREN stmt %prec NOELSE { If($3, $5, Block([])) }
  | IF LPAREN expr RPAREN stmt ELSE stmt    { If($3, $5, $7) }
  | FOR LPAREN expr_opt SEMI expr SEMI expr_opt RPAREN stmt
     { For($3, $5, $7, $9) }
  | FOR LPAREN ID IN expr RPAREN stmt { ForIter($3, $5, $7) }
  | FOR LPAREN typ ID IN expr RPAREN stmt { DecForIter($3, $4, $6, $8) }
  | FOR LPAREN LPAREN formal_list RPAREN IN expr RPAREN stmt { DeconstForIter($4, $7,
$9) }
  | WHILE LPAREN expr RPAREN stmt { While($3, $5, Expr(Noexpr)) }

expr_opt:
    /* nothing */ { Noexpr }
  | expr          { $1 }

expr:
    INTLIT          { IntegerLiteral($1) }
  | CHARLIT         { CharacterLiteral($1) }
  | FLOATLIT        { FloatLiteral($1) }
  | STRLIT          { StringLiteral($1) }
  | TRUE             { BoolLit(true) }
  | FALSE            { BoolLit(false) }
  | ID               { Id($1) }
  | list_exp         { $1 }
  | dict_exp         { $1 }

  | expr PLUS   expr { Binop($1, Add,   $3) }
  | expr MINUS  expr { Binop($1, Sub,   $3) }
  | expr TIMES  expr { Binop($1, Mult,  $3) }
  | expr DIVIDE expr { Binop($1, Div,   $3) }
  | expr MODULO expr { Binop($1, Mod, $3) }

  | ID PLUS_ASSIGN expr { OpAssign($1, Add, $3) }
```

```
  | ID MINUS_ASSIGN expr { OpAssign($1, Sub, $3) }
  | ID TIMES_ASSIGN expr { OpAssign($1, Mult, $3) }
  | ID DIVIDE_ASSIGN expr { OpAssign($1, Div, $3) }

  | expr EQ     expr { Binop($1, Equal, $3) }
  | expr NEQ    expr { Binop($1, Neq,   $3) }
  | expr LT     expr { Binop($1, Less,  $3) }
  | expr LEQ    expr { Binop($1, Leq,   $3) }
  | expr GT     expr { Binop($1, Greater, $3) }
  | expr GEQ    expr { Binop($1, Geq,   $3) }
  | expr AND    expr { Binop($1, And,   $3) }
  | expr OR     expr { Binop($1, Or,    $3) }
  | expr HAS     expr { Binop($1, Has,    $3) }

  | expr QUESTION expr COLON expr { Ternop($1, $3, $5) }

  | MINUS expr %prec NEG { Unop(Neg, $2) }
  | NOT expr         { Unop(Not, $2) }
  | expr PLUS PLUS %prec INCR   { Unop(Incr, $1) }
  | expr MINUS MINUS %prec DECR { Unop(Decr, $1) }

  | typ ID ASSIGN expr { DecAssign($1, $2, $4) }
  | ID ASSIGN expr   { Assign($1, $3) }
  | LPAREN formal_list RPAREN ASSIGN expr { Deconstruct($2, $5) }

  | expr ARROPEN expr ARRCLOSE { Access($1, $3) }
  | expr ARROPEN expr ARRCLOSE ASSIGN expr { AccessAssign($1, $3, $6) }

  | typ ID ASSIGN MATCH pattern { DecPatternMatch($1, $2, $5) }
  | ID ASSIGN MATCH pattern { PatternMatch($1, $4) }

  | ID LPAREN actuals_opt RPAREN { Call($1, $3) }
  | expr DOT ID LPAREN actuals_opt RPAREN { AttributeCall($1, $3, $5) }

  | LPAREN expr RPAREN { $2 }

pattern:
    c_pattern MATCH expr { MatchPattern($1, $3) }

c_pattern:
    expr COLON expr { [ConditionalPattern($1, $3)] }
  | expr COLON expr BAR c_pattern { ConditionalPattern($1, $3) :: $5 }

dict_exp:
    ARROPEN dict_elems ARRCLOSE { DictLit($2) }

dict_elems:
    dict_elem   { [$1] }
    | dict_elem COMMA dict_elems  { $1 :: $3 }

dict_elem:
    expr COLON expr { DictElem($1, $3) }
```

```
list_exp:
  ARROPEN list_elems ARRCLOSE { ListLit($2) }

list_elems:
  /* nothing */           { [] }
  | expr                  { [$1] }
  | expr COMMA list_elems { $1 :: $3 }

actuals_opt:
    /* nothing */ { [] }
  | actuals_list  { List.rev $1 }

actuals_list:
    expr                    { [$1] }
  | actuals_list COMMA expr { $3 :: $1 }


/* MARK: Stuff that we aren't using, but we might use (lol) */

/*
tuple_exp:
    LPAREN tuple_elems RPAREN   { TupleLit($2) }

tuple_elems:
    expr           { [$1] }
  | expr COMMA tuple_elems  { $1 :: $3 }
*/

/*
    | typ LPAREN formals_opt RPAREN ARROW stmt
      { { typ = $1;
      fname = "anon";
      formals = $4;
      body = [$7];
      autoreturn = true } }
    | typ LPAREN formals_opt RPAREN ARROW LBRACE stmt_list RBRACE
      { { typ = $1;
      fname = "anon";
      formals = $4;
      body = List.rev $8;
      autoreturn = false } }
 */

----------
----------
File: src/listcheck.ml
(*
Checking for array types
*)

open SAST
```

```
let eval_list = List.map (expr scope deepscope) l in

let rec check_types = function
        (t1, _) :: [] -> (t1, SListLiteral(eval_list))
      | ((t1,_) :: (t2,_) :: _) when n1 != n2 ->
      raise (Failure "List types are inconsistent")
      | _ :: t -> check_types t
    in check_types eval_list
----------
----------
File: src/semantdriver.ml
(*
Module to generate a SAST given a desugared AST
*)
open Ast
open Sast
open Boolcheck
open Rhandlhand
open Decs

let check (statements, functions) =

(* Gets symbol table of statement scope, and a list of symbol tables with each
function's scope *)
let symbol_table = get_decs (statements, functions) in
let global_scope = fst symbol_table in
let function_scopes = snd symbol_table in

(* Verifies that a function has a valid return statement *)
let rec check_return slist ret = match slist with
    Return _ :: _ -> if ret != Nah then true else raise(Failure "Function of type Nah
should not have a return statement")
  | s :: ss -> check_return ss ret
  | [] -> if ret = Nah then true else raise (Failure "Function has an empty body at
the highest level but returns (?)") in


let check_expr_scope scope = function
    DecAssign(ty, s, _) -> add_symbol_driver s ty scope
  | _ -> scope in

let rec check_stmt_scope scope = function
    Expr(e) -> check_expr_scope scope e
  | Dec(ty, s) -> add_symbol_driver s ty scope
  | While(p, _, _) -> check_expr_scope scope p
  | If(p, _, _) -> check_expr_scope scope p
  | PretendBlock(sl) -> List.fold_left (fun m f -> check_stmt_scope m f) scope sl
  | _ -> scope in

(* Bug fix for function return type mismatching *)
let return_func = function
```

```
    Function(e) -> e
  | e           -> e
  | _           -> raise (Failure "function return type is flawed") in

let type_check t1 t2 =
  let type1 = match t1 with
  Group([ta; tb]) -> (ta, tb)
| _ -> (t1, t1) in
  let type2 = match t2 with
  Group([tc; td]) -> (tc, td)
| _ -> (t2, t2) in
(fst type1 = fst type2) && (snd type1 = snd type2)

in

(* Driver for semantically checking expressions *)
let rec expr scope deepscope e = match e with
    IntegerLiteral l -> (Int, SIntegerLiteral l)
  | CharacterLiteral l -> (Char, SCharacterLiteral l)
  | BoolLit l -> (Bool, SBoolLiteral l)
  | FloatLiteral l -> (Float, SFloatLiteral l)
  | StringLiteral l -> (String, SStringLiteral l)
  | Noexpr -> (Nah, SNoexpr)
  | ListLit l -> let eval_list = List.map (expr scope deepscope) l in
      let rec check_types = function
          [] -> (Nah, SDictLiteral([]))
        | (t1, _) :: [] -> (Array(t1), SListLiteral(eval_list))
        |   ((t1,_) :: (t2,_) :: _) when t1 != t2 ->
          raise (Failure ("Error: list types " ^ string_of_typ t1 ^ " and " ^
string_of_typ t2 ^ " are inconsistent"))
        | _ :: t -> check_types t
      in check_types eval_list
  | DictElem(l, s) -> let (t1, e1) = expr scope deepscope l in
      let (t2, e2) = expr scope deepscope s in
      (Group([t1; t2]), SDictElem((t1, e1), (t2, e2)))
  | DictLit l -> let eval_list = List.map (expr scope deepscope) l in
      let rec check_types = function
        | (Group([t1; t2]), _) :: [] -> (Dictionary(t1, t2), SDictLiteral(eval_list))
        |   ((Group([t1; t2]), _) :: (Group([t3; t4]), _) :: _) when not ((type_check
t1 t3)) || not ((type_check t2 t4)) (*t1 != t3 || t2 != t4 *)->
           raise (Failure (string_of_typ t1 ^ string_of_typ t2 ^ string_of_typ t3 ^
string_of_typ t4))
        | _ :: t -> check_types t
      in check_types eval_list
  | Id l -> (toi scope l, SId l)
  | Binop(e1, op, e2) as e ->
      let (t1, e1') = expr scope deepscope e1
      and (t2, e2') = expr scope deepscope e2 in
      (* All binary operators require operands of the same type *)
      let same = t1 = t2 in
      (* Determine expression type based on operator and operand types *)
      let ty = match op with
```

```
          Add | Sub | Mult | Div | Mod when same && t1 = Int   -> Int
        | Add | Sub | Mult | Div | Mod when same && t1 = Float -> Float
        | Equal | Neq                  when same               -> Bool
        | Less | Leq | Greater | Geq   when same && (t1 = Int || t1 = Float) -> Bool
        | And | Or | Has               when same && t1 = Bool -> Bool
        | _ -> raise (Failure ("illegal binary operator " ^
                     string_of_typ t1 ^ " " ^ string_of_op op ^ " " ^
                     string_of_typ t2 ^ " in " ^ string_of_expr e))
      in (ty, SBinop((t1, e1'), op, (t2, e2')))
  | Unop(uop, e) as ex ->
      let (t, e') = expr scope deepscope e in
      let ty = match uop with
          Neg | Incr | Decr when t = Int || t = Float -> t
        | Not when t = Bool -> Bool
        | _ -> raise (Failure ("illegal unary operator " ^
                              string_of_uop uop ^ string_of_typ t ^
                              " in " ^ string_of_expr ex))
      in (ty, SUnop(uop, (t, e')))
  | Assign(s, e) ->
      let lt = toi scope s
      and (rt, e') = expr scope deepscope e in
      (check_assign lt rt, SAssign(s, (rt, e')))
  | Deconstruct(l, e) -> (****** Work in progress, discrepancy between group and list
literals ******)
      let (e_typ, _) = expr scope deepscope e in
      let _ = match e_typ with
          Group(typs) -> typs
        | _ -> raise (Failure ("Error: deconstruct requires a Group, but was given " ^
string_of_typ e_typ ^ " " ^ string_of_expr e))
      in (Int, SDeconstruct(l, expr scope deepscope e))
  | OpAssign(s, op, e) -> let (t, e1) = expr scope deepscope e in
      if t = (toi scope s) then (t, SOpAssign(s, op, (t, e1))) else raise (Failure
"types not the same")
  | DecAssign(ty, l, expr1) -> (match ty, l, expr1 with
        (Array(t), n, ListLit([])) -> (Array(t), SDecAssign(Array(t), n, (Array(t),
SListLiteral([]))))
      | (Dictionary(t1, t2), n, DictLit([])) -> (Dictionary(t1, t2),
SDecAssign(Dictionary(t1, t2), n, (Dictionary(t1, t2), SDictLiteral([]))))
      | _ -> check_decassign ty l (expr scope deepscope expr1) )
  | Access(e1, e2) ->
      let (t1, e1') = expr scope deepscope e1
      and (t2, e2') = expr scope deepscope e2 in
      (match t1 with
          Array(t) when t2 = Int -> (t, SAccess((t1, e1'), (t2, e2')))
        | Array(_) -> raise (Failure ("Error: integer required for array access, given
type " ^ string_of_typ t2))
        | Dictionary((key_t, value_t)) when t2 = key_t -> (value_t, SAccess((t1, e1'),
(t2, e2')))
        | Dictionary((key_t, _)) -> raise (Failure ("Error: " ^ string_of_typ key_t ^
" required for dictionary access, given type " ^ string_of_typ t2))
        | _ -> raise (Failure ("Error: access not invalid for type " ^ string_of_typ
t1)))
```

```
  | AccessAssign(e1, e2, e3) ->
      let (t1, e1') = expr scope deepscope e1
      and (t2, e2') = expr scope deepscope e2
      and (t3, e3') = expr scope deepscope e3 in
      (match t1 with
          Dictionary((key_t, val_t)) when t3 = val_t ->
            if t2 = key_t then (t3, SAccessAssign((t1, e1'), (t2, e2'), (t3, e3')))
            else raise (Failure ("Error: key type " ^ string_of_typ key_t ^ " expected
for Dictionary access, but " ^
                                 string_of_typ t2 ^ " given in expression " ^
string_of_expr e2))
        | Dictionary((_, val_t)) -> raise (Failure ("Error: value type " ^
string_of_typ t3 ^ " cannot be included in Dictionary " ^
                                           string_of_expr e1 ^ " with value type " ^
string_of_typ val_t))
        | Array(t) when t = t3 ->
            if t2 = Int then (t3, SAccessAssign((t1, e1'), (t2, e2'), (t3, e3')))
            else raise (Failure ("Error: integer expected for Array access, but " ^
string_of_typ t2 ^
                                 " given in expression " ^ string_of_expr e2))
        | Array(t) -> raise (Failure ("Error: type " ^ string_of_typ t3 ^ " cannot be
included in Array " ^ string_of_expr e1 ^
                                      " with type " ^ string_of_typ t))
        | _ -> raise (Failure ("Error: expression " ^ string_of_expr e1 ^ " has type "
^ string_of_typ t1 ^
                                 ", expected type Array")))
  | Call(fname, args) ->
      let eval_list = List.map (expr scope deepscope) args in
      let key_func = key_string fname eval_list in
      let fd = StringMap.find key_func function_scopes in
      let param_length = StringMap.cardinal fd.formals.variables in
      if List.length args != param_length then
      raise (Failure ("expecting " ^ string_of_int param_length ^
                " arguments in function call" ))
      else let check_call (_, ft) e =
      let (et, e') = expr scope deepscope e in
      (check_assign ft et, e')
      in
      let args' = List.map2 check_call (StringMap.bindings fd.formals.variables) args
      in (return_func fd.ret_typ, SCall(fname, args'))
  | AttributeCall(e, fname, args) ->
      let eval_list = List.map (expr scope deepscope) args in
      let key_func = key_string fname eval_list in
      let fd = StringMap.find key_func function_scopes in
      let param_length = StringMap.cardinal fd.formals.variables in
      if List.length args + 1 != param_length then raise (Failure ("expecting " ^
string_of_int param_length ^ " arguments in function call"))
      else let check_call (_, ft) e =
      let (et, e') = expr scope deepscope e in
      (check_assign ft et, e') in
      let args' = List.map2 check_call (StringMap.bindings fd.formals.variables) args
      in (return_func fd.ret_typ, SAttributeCall(expr scope deepscope e, fname,
```

```
args'))
  (*
  | Ternop(e1, e2, e3) ->
      let (e1t, e1') = expr scope deepscope e1 in
      if e1t != Bool then raise (Failure "Error: expected bool in first expression of
ternary operator") else
      let (e2t, e2') = expr scope deepscope e2
      and (e3t, e3') = expr scope deepscope e3 in
      if e2t != e3t then raise (Failure ("Error: ternary operator types " ^
string_of_typ e2t ^ " and " ^ string_of_typ e3t ^ " do not match"))
      else (e2t, STernop((e1t, e1'), (e2t, e2'), (e3t, e3'))) *)
  | _  -> raise (Failure "expression is not an expression")
in

(* Driver for semantically checking statements *)
let rec check_stmt scope inloop s =
  let new_scope = {
    variables = StringMap.empty;
    parent = Some(scope);
  } in match s with
    Expr e -> SExpr (expr scope inloop e)
  | Skip e -> if inloop then SSkip (expr scope inloop e) else raise (Failure "skip not
in a loop")
  | Abort e -> if inloop then SAbort (expr scope inloop e) else raise (Failure "abort
not in a loop")
  | Panic e -> SPanic (expr scope inloop e)
  | If(p, b1, b2) as i ->
      let scope = get_expr_decs scope p in
      let pred = check_bool (expr scope inloop p)
      and t = check_stmt scope inloop b1
      and f = check_stmt scope inloop b2 in SIf(pred, t, f)
  | While(p, s, inc) ->
      let scope = get_expr_decs scope p in
      let pred = check_bool (expr scope inloop p)
      and loop = check_stmt scope true s in SWhile(pred, loop, check_stmt scope inloop
inc)
  | For(e1, e2, e3, s) -> raise (Failure ("Error: nested for loops currently broken"))
  | Return _ -> raise (Failure "return outside a function")
  | Block sl ->
      let rec check_stmt_list blockscope = function
          [Return _ as s] -> [check_stmt blockscope inloop s]
        | Return _ :: _   -> raise (Failure "nothing may follow a return")
        (*| Block s :: ss   -> check_stmt_list blockscope (s @ ss)  Flatten blocks
        | PretendBlock sl :: ss -> check_stmt_list blockscope (sl @ ss)  Flatten
blocks *)
        | s :: ss         -> check_stmt blockscope inloop s :: check_stmt_list
blockscope ss
        | [] -> []
      in SBlock(check_stmt_list (List.fold_left (fun m f -> check_stmt_scope m f)
new_scope sl) sl)
  | PretendBlock sl ->
        SBlock(List.map (check_stmt scope inloop) sl)
```

```
  | Dec(ty, l) -> SDec(ty, l)
  | _ as s -> raise (Failure ("statement " ^ string_of_stmt s ^ " is not a
statement"))
in

(* Check statements within functions *)
let rec check_stmt_func scope inloop ret =
  let new_scope = {
    variables = StringMap.empty;
    parent = Some(scope);
  } in function
    Expr e -> SExpr (expr scope inloop e)
  | Skip e -> if inloop then SSkip (expr scope inloop e) else raise (Failure "skip not
in a loop")
  | Abort e -> if inloop then SAbort (expr scope inloop e) else raise (Failure "abort
not in a loop")
  | Panic e -> SPanic (expr scope inloop e)
  | If(p, b1, b2) -> SIf(check_bool (expr scope inloop p), check_stmt_func scope
inloop ret b1, check_stmt_func scope inloop ret b2)
  | While(p, s, inc) -> SWhile(check_bool (expr scope inloop p), check_stmt_func
new_scope true ret s, check_stmt scope inloop inc)
  | Return e -> let (t, e') = expr scope inloop e in
      if t = ret then SReturn (t, e')
      else raise (
      Failure ("return gives " ^ string_of_typ t ^ " expected " ^
        string_of_typ ret ^ " in " ^ string_of_expr e))
  | Block sl ->
      let rec check_stmt_list blockscope = function
          [Return _ as s] -> [check_stmt_func blockscope inloop ret s]
        | Return _ :: _   -> raise (Failure "nothing may follow a return")
        (*| Block sl :: ss  -> check_stmt_list blockscope (sl @ ss)  Flatten blocks
        | PretendBlock sl :: ss -> check_stmt_list blockscope (sl @ ss)  Flatten
blocks *)
        | s :: ss         -> check_stmt_func blockscope inloop ret s ::
check_stmt_list blockscope ss
        | []              -> []
      in SBlock(check_stmt_list (List.fold_left (fun m f -> check_stmt_scope m f)
new_scope sl) sl)
  | PretendBlock sl ->
      SBlock(List.map (check_stmt_func scope inloop ret) sl)
  | Dec(ty, l) -> SDec(ty, l)
  | _ as s -> raise (Failure ("statement " ^ string_of_stmt s ^ " is not a
statement"))
in

(* Check function declarations *)
let check_function (fd : func_decl) =
  if check_return fd.body (return_func fd.typ) then
    let key_func = key_string fd.fname fd.formals in
      let current_function = StringMap.find key_func function_scopes in
      { styp = return_func fd.typ;
        sfname = fd.fname;
```

```
        sformals = fd.formals;
        sbody = match check_stmt_func current_function.locals false (return_func
fd.typ) (Block fd.body) with
              SBlock(sl) -> sl
          | _ -> raise (Failure ("internal error: block didn't become a block?"))
    }
    else raise (Failure "there is not return statement at the highest level of the
function")
in

(* Collect the SAST of semantically-check statements and functions *)
let sstatements = List.map (check_stmt global_scope false) statements in
let sfuncs = List.map check_function functions in

(* Aggregate statements into an implicit main function if one isn't already defined *)
let rec has_main sfuncs = match sfuncs with
    [] -> false
  | sfd :: _ when sfd.sfname = "main" && sfd.styp = Int -> true
  | sfd :: _ when sfd.sfname = "main" -> raise (Failure ("Error: function main must
return type Int, not type " ^ string_of_typ sfd.styp))
  | _ :: tail -> has_main tail in

let updated_sfuncs = if has_main sfuncs then sfuncs else
  { styp = Int;
    sfname = "main";
    sformals = [];
    sbody = List.rev sstatements;
  } :: sfuncs in

(sstatements, updated_sfuncs)

----------
```

↩️ Back to Contents 📌