The Stop Programming Language
Stop, a Simple funcTional Object--oriented Programming language

Jillian Knoll (jak2246) | Jonathan Barrios (jeb2239) | James Stenger (jms2431) | Lusa Zhan (lz2371)

Table of Contents

1. INTRODUCTION 3
1.1 RELATED WORK 3
2. TUTORIAL 4
2.1 ENVIRONMENT 4
2.2 HELLO WORLD 4
2.3 BASICS 4
2.4 CONTROL FLOW 5
2.5 CLASSES 6
2.6 FUNCTIONS 6
3. LANGUAGE REFERENCE MANUAL 8
3.1 TYPES 8
3.2 LEXICAL CONVENTIONS 9
3.3 EXPRESSIONS 12
3.4 OPERATORS 14
3.5 STATEMENTS 19
3.6 STANDARD LIBRARY 21
4. PROJECT PLAN (LUSA) 22
4.1 PLANNING PROCESS 22
4.2 SPECIFICATION PROCESS 22
4.3 DEVELOPMENT PROCESS 22
4.4 STYLE GUIDE 22
4.5 Timeline 23
4.6 TESTING PROCESS 23
4.7 TEAM ROLES 23
4.8 SOFTWARE DEVELOPMENT TOOLS 23
5. LANGUAGE EVOLUTION (JILLIAN) 25
5.1 INTRODUCTION 25
5.2 STRUCTURE 25
5.3 VARIABLES 25
5.4 FUNCTION DEFINITION 26
5.5 CLASS DEFINITION 26
6. TRANSLATOR ARCHITECTURE (JAMES + JONATHAN) 27
1. Introduction

Stop is a general purpose programming language, syntactically similar to Scala, that compiles to the LLVM Intermediate Representation (LLVM IR). Stop is both functional and object oriented. Program structure is oriented surrounding classes that contain a main method and several function definitions.

As a general purpose language, Stop has limitless possibilities for use cases. Users can harness the speed of a relatively expressive language compiled to bytecode, allowing users to combine Stop in a library with the language of their choosing. Stop can be utilized for a wide range of purposes, ranging from high level data computation to application building.

Since Stop is object-oriented, it allows for robust user created libraries. Every function and variable will be considered intrinsically as an object. This system will allow users of our language to easily create easily reusable inheritance structures to allow libraries to interact with additional languages.

We wanted to create a language that compiles to LLVM to produce the most efficient bytecode. As a side effect of compiling LLVM, stop bytecode files can be easily integrated into components of programs written in other languages that also compile to LLVM and follow C-style calling conventions.

1.1 Related work

The compiler structure is largely based on the Dice and MicroC compilers as they both provided excellent examples of how to use the llvm ocaml bindings. The language itself was inspired by the syntax of the Scala programming language. Scala has a unique types system that allows smooth communication between functional and object oriented concepts. This is predominantly achieved by having functions as first class objects. Similarly to Java or any other object-oriented non-functional programming language, a Stop user can define a “Person” object with several variable fields, declare the members of these fields, and continue to access these fields while the object remains in scope.

Function application is largely syntactic sugar to overlay calling a function object’s “apply” method. This abstraction also provides a framework for transforming higher level functional features into imperative IR. This is how we came up with the idea of generating structs to keep track of closure information. Similar methods have been used in imperative languages which later had lambda functions added as a result of user demand (eg Java 8, C++11).
2. Tutorial

2.1. Environment

The environment for the compiler was an Ubuntu 15.10 VM. All the testing and the coding were done on this VM. Opam was first installed on the virtual machine, which was used to install Ocaml, Core, and Llvm as well.

2.2. Hello World

In order to run the compiler, we need to first use make in the stop directory. This will compile all the necessary components to translate Stop files to LLVM IR. Let us consider the following program in stop, 'test-helloworld.stp':

```ocaml
def main = ():Int {
    printf("Hello, World!");
    return 0;
}
```

There are two steps to running this program, assuming test-helloworld.stp is located in the stop directory:

1) Run ./stop.native -cff test-helloworld.stp
2) Run lli test-helloworld.ll

This should print out the desired result "Hello, World!".

2.3. Basics

2.3.1. 'def main=():Int' :

In order to compile, the Stop program must have a main method as an entry point, as shown above with the test-helloworld.stp program.

2.3.2. Variable declaration

Variables are declared as follows:

```ocaml
var <var_name>:<Type>;
```

A few examples of initializations are:

```ocaml
var a:Int=2;
var b:Char = 'a';
var c:Bool = true;
```
var d:Float = 1.1;
var arr:Int[] = Int[3];

2.3.3. Printf

Printing works similarly to C by calling the printf function. It takes a string, and possibly variables as arguments.

var a:Int = 0;
var c:Char = ‘a’;
Var f:Float = 1.1;
printf(“Hello”);
printf(“%d %c %f”, a,c,f); //prints 0 a 1.1

2.4. Control Flow

2.4.1. If

It is not possible to declare variables inside the if body. An example of an if, followed by optional else if statements is:

var a:Int;
var b:Int=1;
if(b<1){
    a = 0;
} else if(b>1){
    a = 1;
} else{
    a = 2;
}//a=2

2.4.2. For and While

Note that variables cannot be declared inside a for or while statement. Below is an example of a for and while loop that would work:

var a:Int;
for(a=0;a<5;a++){  
    printf(“%d”, a);
} //prints 01234

while(a<10){
    printf(“%d”,a);
    a++;
}
2.5. Classes

Classes are declared outside of the main function using the keyword “class”. Below is an example of a Rectangle class that is accessed inside the main function.

```java
def main = ():Int {
    var j:Rectangle;
    j.w = 5;
    j.h = 10;
    printf("%d %d\n", j.w, j.h); //prints 5 10
    return 0;
}

class Rectangle = {
    var w:Int;
    var h:Int;
}
```

2.6. Functions

Functions outside of the main function are declared using the keyword def or as anonymous functions using the ‘@’ operator in the following pattern:

```java
def <fun_name> = (<var_name1>:<var_type1>, <var_name2>:<var_type2> ....):<return_type>{
//statements}
```

Functions that have return type Util do not return anything (similar to void). However, if functions have another return type, they must return the given type. An example is shown below:

```java
def f1 = (a:Int):Int {
    return a + 5;
}

def f2 = (a:Int):Util {
    print("%d", a);
}

def main = ():Int {
    printf("%d", f1(2)); //prints 5
    f2(4); //prints 4
    return 0;
```
//anonymous function
var f2 = @(b:Int):Int {
    return a + b;
};

2.6.1. Nested Functions

Functions can be declared within other functions and these functions can be passed as objects between functions. The notation `<type1> -> <type2>` is denoted as a function return type where type1 refers to the argument(s) taken by the function and type2 refers to the return type of the function.

Thus functions that return other functions can be declared as follows:

```java
def f1 = (a:Int):Int->Int {
    var f2 = @(b:Int):Int {
        return a + b;
    };
    return f2;
}
```

The 'Int -> Int' notation specifies that the return type is a function rather than a primitive or user defined data type.

3.1. Types

- **Primitive Data Types**
  - **Integers**
    - The type integer stores numerical values in 32 bits.
    - Ways to declare an integer
      - var a:Int ;
      - var a:Int = 1 ;
  - **Float**
    - The type float stores numerical values in 64 bits.
    - Methods of declaring a float
      - var f:Float ;
      - var f:Float = 1.0;
  - **Unit**
    - The type unit indicates that a function does not return an object at the point of the function call.
      - This is utilized in functions that print values rather than returning an object.
      - def square = (var u:Float):Unit { printf("%f",u*u); }  
      - The above function is of type Unit as the printf("%f",u*u) statement does not create an object to return.
  - **Char**
    - The type char indicates a single character stored as 1 byte integer. char is declared by enclosing the single character with a single set of quotation marks.
    - Methods of declaring a char
      - var a:Char;
      - var a:Char = ‘a’;
      - var a:Char = 2 ;
  - **Bool**
    - The boolean type is a binary value that stores a true or false in one bit. A type null cannot be used to declare a boolean variable.
    - Ways of declaring a boolean
      - var b:Bool = true;
      - var b:Bool = false;

- **Scope of primitive data types**
  - The scope of primitive data types is only within the classes or functions in which the data types are declared. Primitive data types declared within outer functions can be accessed within inner functions.
```java
def main = ():Int {
    var a:Int = 5;
    var f1 = @(a:Int):Int {
        return a;
    };
    return 0;
}
```

The integer variable ‘a’ declared in the outer class is accessible from within the inner class. The integer ‘a’ is not accessible from outside of the main() method.

- **Casting**
  - Casting is prohibited and will result in a compile time error.

- **Non-Primitive Data Types**
  - **Arrays**
    - Arrays are a data structure used to store objects, consisting of both primitive types and other user created or standard library datatypes. Array indexing begins at 0.
    - **Array declaration**
      - Arrays can be declared in the following methods
        - Type Declaration method, empty array:
          - var arr:<Type>[];
        - var arr:<Type>[]=<Type>[size]; //declared array initialized to 0’s
      - **Array access**
        - Accessed through the [<index>] operator in shorthand, representative of the .operator[<index>] in longhand
          - var x:Int = arr[3]; //x now contains the integer stored at position 3

3.2. **Lexical Conventions**

The types of tokens include identifiers, keywords, literals, comments, separators, white space, and operators. White space serves to separate tokens.

- **Identifiers**
  - Only alphabetical letters, digits, and the underscore character ‘_’ may be used in variable, object, function, and class declaration. All such names must begin with a lowercase ‘a’-‘z’ letter but may include any other listed characters in the remainder of the name.
  - We also have type identifiers, these must start with an uppercase letter ‘A’-‘Z’ but can contain any letter afterwards

- **Keywords**
Keywords are reserved words within the language. These words cannot be overloaded by another function declaration. Keywords include the names of primitive and nonprimitive data types as well as words defining conditional statements, modifiers, and function declaration components.

- if, else, for, while, break, continue, return
- Int, Float, Bool, Char, Unit
- true, false
- def, class, #include

**Literals**

- Literals are the notation in source code for representing primitive data types in source code.
  - **Integer Literals**
    - Integer literals are indicated using optionally signed decimal notation, denoted by a string of repeating digits. Integers cannot be declared using scientific or exponential notation.
      
      an integer is matched with the regular expression
      \[\text{int} = \['0'-'9']^{+}\]
      
  - **Character Literals**
    - Character literals are denoted by a single, lowercase, a-z character enclosed by two single quotes. These are the only forms of information which may be stored within a character variable.
      
      a char is matched with the regular expression
      \[\text{char} = \['a'-'z']\]
      
  - **Float Literals**
    - A float literal is denoted by an integer part, a decimal point, 0 or more digits after the decimal point, an e, and an optionally signed integer exponent. Either the integer or digits after the decimal point may be missing, either the decimal point or the exponent portion may be missing.
      
  - **Boolean Literals**
    - A boolean literal is denoted by one of two reserved words, true or false.
      
      a bool is matched with the regular expression
      \[\text{bool} = \['true'\]|\'false\']\]
      
  - **String Literals**
    - A string literal is denoted by a series of ASCII characters and whitespace enclosed with double quotes. Escape sequences must be used for the identification of whitespace literals within a string.

**Separators**
Separators are used to denote the distinction between multiple tokens in source code.

```plaintext
rule token = parse
    [' ' 't' 'r' 'n'] { token lexbuf }
  | '"""" { single_comment lexbuf }
  | '""*' { multi_comment lexbuf }
  | \n' { NEWLINE }
  | '(' { LPAREN }
  | ')' { RPAREN }
  | '{' { LBRACE }
  | '}' { RBRACE }
  | '[' { LSQUARE }
  | ']' { RSQUARE }
  | ':' { COLON }
  | ';' { SEMI }
  | ',' { COMMA }
```

- Operators

  The following operators are in use as lexical tokens. These operators

  (* Operators *)

  ```plaintext
  | '+'  { PLUS }
  | '-'  { MINUS }
  | '*'  { TIMES }
  | '/'  { DIVIDE }
  | '='  { ASSIGN }
  | '%'  { MODULO }
  | '==' { EQ }
  | '!'= { NEQ }
  | '<'  { LT }
  | '<=' { LEQ }
  | '>'  { GT }
  | '>=' { GEQ }
  | '&&' { AND }
  | '||' { OR }
  | '!'  { NOT }
  | '.'  { DOT }
  ```

- White Space

  The following white space characters are in use and must be referenced as literals using the escape character `\`

  ```plaintext
  \n' \t' \r' \n'
  ```

- Capitalization

  Capital letters are used to define classes as a user defined object type.
  Variables must be instantiated by names beginning with a lowercase 'a'- 'z' letter. Capital letters can be used within the remainder of the variable.
name. The only reserved word that is capitalized is the void return type of "Unit".

- **White Space**
  - One single space, ' ', is required to separate tokens
  - All other white space is ignored. Thus the following forms are equivalent
    - def true_or_false = (var u:Bool):Unit
      {printf(u);}
    - def true_or_false = (var u:Bool):Unit
      {printf(u);}
  - def true_or_false = (var u:Bool):Unit {printf(u);}

- **Comments**
  - Comments are denoted with the '//' or '/*....*/' sign, differentiating a single line or comments or multiple lines of comments
  - #include is a reserved keyword though the word include is not otherwise reserved.
    - // single line of comments
    - /* multiple lines of comments */

3.3. **Expressions**

- **Function definition**
  - Functions are defined with the def keyword, the function name, the assignment operator, a list of arguments, and a type declaration for the return type of the function. A return statement must exist within the function if the function does not return type unit.
    - def <function Name> = (<var <varname>:<Var Type>, additional args):<ReturnType>
      { statements to execute
        return <Object of Return Type>;
    
    - def negate = (i:Int):Int
      { return -i;}
    
    - def count =(arr:Array):Unit
      { for(a=0;a<10;a++)
        { printf("%d",arr[a]); } } }

- **Object instantiation**
  - Objects are instantiated upon declaration, it is possible to create initializer functions to automate object initialization.
  - Objects are instantiated with type declarative format.
    - var x:Char = "a"; //a is type char
    - var x:Int = 1; //declared that x is of type int
var x:Rectangle; //this has memory allocated but needs to be instantiated manually
    x.width = 4;
x.length = 9;

○ When specified in function calls as arguments or return types the type of the object must be specified.
  ■ def funct_name = (var var_name:<Var_type> <with additional args separated by commas>):<Return_type>
  ■ def square = (var u:Int):Int {return(u*u); }  //square returns an int
  ■ The type int is specified after var u with a colon in the argument list of the function to produce (var u:Int)
  ■ The type int is specified as a return type outside of the argument list for the function with the colon to produce (var u:Int):Int

● Function calls
  ○ Functions are called by referencing the function name and any arguments in parentheses. Unless the return type is of type Unit, the function call must be enclosed within a conditional statement or on the right hand of the assignment operator.
    def increm(var a:Int):Int {return a + 1;}
    var x:Int = increm(3);
    if(increm(x) == 2) <statements>

● Class declaration and instantiation
  ○ Classes are declared with the class keyword and can be instantiated as objects outside of the class definition. A user defined rectangle object can be defined in this manner using the class keyword

    class Rectangle {
        var x:Int;
        var y:Int;
        var w:Int;
        var h:Int;
    }

  ○ An object of type rectangle can be declared using the general variable assignment sequence

    var s:Rectangle;

The variable components can be referred to publicly by referencing the variable name.

    s.w = 5;
s.h = 10;
3.4. Operators

The following methods are associated with the operator tokens described in the Keywords section.

- **Assignment**
  - The `=` operator assigns values to variables, function, and class declaration.
  - The left side of the `=` operator contains a reference to the type of expression that is being called, def for function declaration, var for variable instantiation, or class for class definition followed by the name of the variable. The first letter of the variable corresponds to whether this is the declaration of a new type of object, as denoted by a capital first letter, or a lowercase letter for function declaration or variable declaration.
  - The right side of the `=` operator contains an expression that will produce the corresponding type specified by the left hand side of the operator. Otherwise a compile time error will be produced.

```scala
def square = (var u:Int):Unit { print(u*u); }
var x:Int = 3;
```

- **Anonymous function operator**
  - Anonymous functions are specified using the `@` operator in place of a `def <name>...` function declaration.

```scala
var f2 = @ (b:Int):Int {
  return a + b;
}
```

- **Arithmetic**
  - Arithmetic operators must take two objects of the same type as arguments.

```scala
var x = 1;
var y = 4.2;
var z = x + y; //cannot perform operation on two objects of different types.
```
  - The addition operator.

```scala
var x:Int = 4 + 2; //6 is stored in x
var y:Float= 4.0 + 2.0; //6.0 is stored in y
```
  - The subtraction operator.

```scala
var x:Int = 4 - 2; //2 is stored in x
var y:Float = 4.0 - 2.0; //2.0 is stored in y
```
  - The subtraction operator can also be used for negation.
var x:Int = 2; //2 is stored in x
var y:Float = -x; // -2 is stored in y

- *
  - The multiplication operator.
  var x:Int = 4 * 2; // 8 is stored in x
  var y:Float = 4.0 * 2.0; // 8.0 is stored in y

- /
  - The division operator.
  var x:Int = 4 / 2; // 2 is stored in x
  var y:Float = 4.0 / 2.0; // 2.0 is stored in y

- %
  - The modulo operator. Only integer variables or literals may be taken as arguments.
  var x:Int = 10 % 2; // 2 is stored in x
  var y:Float = 12.5 % 2; // This will not compile as one argument is a float literal

● Array
  ○ Access and Declaration
    - Array access is performed with the [index] operator. Array indexes start at 0.
    - Arrays must be declared with a specified size. The type[] specification must be made after the var name element of type declaration.
    var x:Int[] = Int[2];
    - Values can be inserted into arrays by accessing individual array indexes in an iterative function such as a for loop.
    var a:Int[][] = Int[2][5]();
    var i:Int;
    var j:Int;
    for (i=0;i<2;i=i+1){
        for (j=0;j<5;j=j+1){
            a[i][j]=i+j;
        }
    }
    - Values can also be declared at array indexes through individual assignment.
    var a:Int[] = Int[2]();
    a[0] = 34;
    a[1] = 42;

● Conditional Operators
Conditional operators return a boolean value when the condition defining the operator is met. Comparisons to null are acceptable for the ‘==’ and ‘!=’ operators only. The ‘<=’, ‘>=’, ‘<’, and ‘>’ operators are only defined for the int, float, and char primitive datatypes. User defined object types must contain a definition of the ‘<=’, ‘>=’, ‘<’, and ‘>’ operators. These operators can be overloaded through function definition.

○ ==
  ■ The conditional operator ‘==’ will evaluate whether the two objects surrounding the operator are equivalent. The condition will evaluate to true for equivalence and false otherwise.
  ■ The ‘==’ operator should not be used to evaluate the equivalence of type float variables or literals due to rounding error.

```java
var x:Int = 1;
var y:Int = 1;
if(x == y) {return true;} //condition evaluates to true
```

○ !=
  ■ The conditional operator ‘!’=’ will evaluate whether the two objects surrounding the operator are equivalent. The condition will evaluate to true for unequal objects and false otherwise.
  ■ The ‘!’=’ operator should not be used to evaluate the equivalence of type float variables or literals due to rounding error.

```java
var x:Float = 1.5;
var y:Float = 2.5;
if(x != y) {return true;} //condition evaluates to true
```

○ <=
  ■ The ‘<=’ operator will evaluate whether the argument on the right hand side contains a value that is less than or equal to the value contained on the right hand side.

```java
var x:Float = 1.5;
var y:Float = 2.5;
if(x <= y) {return true;} //condition evaluates to true
```

```java
var x:Char = ‘c’;
var y:Float = 1.5;
if(x <= y) {return true;} //cannot attempt to compare two objects of different types.
```

○ >=
  ■ The ‘>=’ operator will evaluate whether the argument on the right hand side contains a value that is greater than or equal to the value contained on the right hand side.

```java
var x:Float = 1.5;
var y:Float = 2.5;
if(y >= x) {return true;} //condition evaluates to true
```
var x:Char = 'c';
var y:Float = 1.5;
if (x > y) {return true;} //cannot attempt to compare two objects of different types.

○ >
  ■ The ‘>’ operator will evaluate whether the argument on the right hand side contains a value that is greater than the value contained on the right hand side.

var x:Float = 1.5;
var y:Float = 2.5;
if(y > x) {return true;} //condition evaluates to true

var x:Char = 'c';
var y:Float = 1.5;
if (x > y) {return true;} //cannot attempt to compare two objects of different types.

○ <
  ■ The ‘<’ operator will evaluate whether the argument on the right hand side contains a value that is less than the value contained on the right hand side.

var x:Float = 1.5;
var y:Float = 2.5;
if(x < y) {return true;} //condition evaluates to true

var x:Char = 'c';
var y:Float = 1.5;
if (x > y) {return true;} //cannot attempt to compare two objects of different types.

● Dot operator
  ○ Shorthand
  Operators can be referenced by writing the string literal of the lexical token in the source code.
  var a:Int = 3 + 5; // the + operator is referenced by its string literal
  ○ Longhand
  Operators can be referred to with the .operator<type>(<argument>) notation
  var x:Int = 1 + 2; //shorthand notation
  var x:Int = 1.operator+(2) /*longhand notation produces logically equivalent outcome*/

● Logical operators
  ○ Logical operators can be used to separate multiple conditions within conditional statements.

  ○ ||
The OR operator will evaluate to true when at least one of the conditions on either side of the OR operator evaluates to true. The OR operator can be used to sequence multiple logical conditions. One of the conditions must evaluate to true in order for the sequenced OR statements to evaluate to true.

```javascript
var x:Bool = false;
var y:Bool = true;
var z:Bool = false;
if (x || y) //the condition within the if statement evaluates to true
if (x || y || z) //the condition within the if statement evaluates to true
```

The AND operator will evaluate to true if both conditions on the either side of the AND operator evaluate to true.

```javascript
var x:Bool = true;
var y:Bool = false;
var z:Bool = true;
if (x && y) //the condition within the if statement evaluates to false
if (x && z) //the condition within the if statement evaluates to true
```

The NOT operator will negate the logical value provided by the conditional statement following the NOT operator.

```javascript
var x = false;
var y = !x; //true is stored in x
if (!(3 == 4)) //the condition within if statement evaluates to true
```

### Precedence

The order of precedence is as follows, ordered by from highest to lowest precedence. The order of precedence refers to the lexical tokens utilized for each operator. If the operator is overloaded then the overloaded operator will maintain the original level of precedence. When multiple operators have equivalent precedence the expressions are evaluated from left to right.

- Calls to functions
- Array access operators
- Arithmetic negation and logical negation
- Multiplication and division operators
- Addition and subtraction operators
- Greater than or equal to, less than or equal to, greater than, or less than operators (‘>’, ‘>=’, ‘<’, ‘<’)
- Equals (‘==’) and not equals (‘!=’) conditional statements
- Logical AND operator
- Logical OR operator
- Assignment operations
3.5. Statements

- Expressions
  - The format for expressions involving arithmetic, array, assignment, conditional statements, and logical operators is described in the Operators section. These declarations must be terminated by a semicolon.

- Declarations
  - The declaration format for variables, functions, and classes is described in the Expression section. These declarations must be terminated by a semicolon.

- Control Flow
  - if
    - The if statement provides a series of steps to execute when the condition inside the parentheses is met. If the condition is not met then the next instruction after the \{<statements>\} will be executed. if statements can be accompanied by an else statement; the else statement need not contain any statements to execute when the if condition is not met.
      - Declaration
      - if (<one or more logical conditions>)
      - \{ <statements to execute when the condition evaluates to true> \}
      - Example
        ```
        if(a == 0){
            return true;
        }
        else{
            return false;
        }
        ```
  - else if
    - The else if statement provides a condition and accompanying series of steps to execute when the conditions in the above if and optional else if statements do not evaluate to true. An else statement must accompany a series of if and else if statements.
      - Declaration
      - else if(<one or more logical conditions>)
      - \{<statements to evaluate when the condition evaluates to true>\}
      - Example
        ```
        if( a == 1 )
        {return true;}
        else if( a % 2 == 0 )
        {return true;}
        else
        {return false;}
        ```
The else statement provides a condition and series of steps to execute when the above series of if and optional elsif statements does not produce conditions which evaluate to true.

Declaration
else{<statements to evaluate when no if or else if conditions evaluate to true>};

Example:
if( a == 0)
  {return true;};
else
  {return false;}

looping
  ○ while
    A while loop will compute the statements enclosed in brackets below the while loop as long as the logical condition within the while loop evaluates to true. The execution path will then move to the next statement after the brackets below the while loop.

  Example:
  var i:Int = 1;
  while( i < 5)
  { printf("%d",i); i = i + 1;}

  ○ for
    A for loop will compute the statements enclosed in the brackets below the for loop as long as the series of logical conditions within the for loop evaluates to true. The computation within the for loop will occur at the beginning of every iteration of the for loop.

    Integer variables used within the body of the for loop must be instantiated outside of the for loop

    arr = (1,2,3,4,5);
    var arr:Int[] = Int[5]();
    var a:Int;
    for(a=0;a<5;a++) { printf("%d",arr[a]); } //prints 12345

return
  ○ Functions must be accompanied by a return statement unless the return type is specified as type Unit.
  //functions of type unit do not require a return statement
def declaring=():Unit
  { var a:Int=0; }

  //if a non-void return type is specified then a return statement is required that will return a variable of the specified type.
def increment=(a: Int): Int { var b: Int = a + 1; return b; }

- For anonymous functions return types must also be specified using the `:<returnType>` notation
  
  ```
  var f2 = @(b: Int): Int {
    return a + b;
  };
  ```

- When the return type is another function the `<ArgType> -> <ReturnTyp`e`e` notation is used to denote returning a function

  ```
  def f1 = (a: Int): Int -> Int {
    var f2 = @(b: Int): Int {
      return a + b;
    };
    return f2;
  }
  ```

  The function `f1` returns a function defined in the variable `f2`.

- If there is no return statement accompanying all logical outcomes of conditional statements the program will not compile.

  ```
  // The following program will not compile
  def even_odd=(a: Int): Bool {
    if(a % 2 == 0) {
      return true
    }
    else {
      // there is no return statement when this logical condition is met
    }
  }
  ```

### 3.6. Standard Library

The following objects and their functions are included in the standard library. This is really the bare minimum but our language can be used to write much larger libraries.

- **Print**
  - The print method is defined for string literals.
    - `printf("abc");` // prints abc
    - `var y: Char = ‘a’;
        printf("%c", y);` // prints a
4. Project Plan (Lusa)

4.1. Planning Process

The initial steps in the planning process were to assign the responsibilities as specified in the Team Roles section (see 4.5) and to set up a time for weekly meetings. During the first weekly meetings we discussed language specifications, long-term goals, and set up the development environment. Group meetings continued throughout the rest of the semester every week, during which we specified small goals for the upcoming week.

We also met up a few times with our assigned TA, Daniel, and David in order to set specific goals and go through the steps towards implementing our language features.

4.2. Specification Process

With respect to our language, we have decided early on to create a language that compiled to LLVM IR in order to deepen our understanding of the compiling process. Jillian, as our Language Guru, decided on the syntax of our language, which we aimed to be similar to Scala.

In terms of features, our initial goals included to create an object-oriented language with type inference and automatic garbage collection. More specifically, we were aiming to implement classes, inheritance, interfaces, type inference, and garbage collection. After discussing these goals with Daniel and David, we have finalized our language features to include classes and nested functions.

4.3. Development Process

The development process followed the deliverable deadlines of the class. We began with the parser and the scanner for the compiler in order to create the Language Reference Manual. The next step was to get the entire pipeline working in order for our “Hello World” program to work. After these deliverables, we met to collaboratively code together in order to implement the other language features. The most basic features, as included in MicroC were implemented first. Nested functions were implemented last.

4.4. Style Guide

The programming style was largely up to each group member, but the most common practices were used. Among these were indentation levels to indicate a level in the corresponding code and the use of vertical alignment in order to facilitate reading the pattern matchings.
4.5. **Timeline**

Below is a timeline with milestones reached on the given dates.

- 01/25/16: Group Formation
- 02/07/16: Language Name and Specifications/Features
- 02/10/16: Proposal
- 03/07/16: Initial Language Reference Manual and Parser and Scanner
- 04/04/16: Full pipeline for print
- 04/06/16: Hello World
- 04/29/16: Classes
- 05/07/16: Nested Functions
- 05/11/16: Submission of final deliverables

4.6. **Testing Process**

The initial test suite was set up by James and expanded by each group member. The test script was taken after the script presented in class by Professor Edwards. Throughout the project, we wrote several tests for each feature. Those would remain in the test suite and be run every time a new language feature was implemented. Tests that failed after new features have been added would allow us to immediately identify issues with new functionalities.

4.7. **Team Roles**

The roles for this project, as specified in the first lecture, were distributed as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lusa Zhan</td>
<td>Manager</td>
</tr>
<tr>
<td>Jillian Knoll</td>
<td>Language Guru</td>
</tr>
<tr>
<td>Jonathan Barrios</td>
<td>Tester</td>
</tr>
<tr>
<td>James Stenger</td>
<td>System Architect</td>
</tr>
</tbody>
</table>

This represents a rough distribution of responsibilities, as every member of our group contributed heavily each part of the compiler.

4.8. **Software Development Tools**

The following software development were used throughout our project:
• **Ubuntu 15.10**: A member of our group created custom vagrant boxes which had all the necessary tools for working on the compiler. This enabled each member of the group to have identical development environments which could be accessed from the command line.

• **LLVM 3.7**: The most stable version of llvm at the time we began development.

• **Ocaml**: We used the opam package manager to install and configure our ocaml programming environment. The two major packages we used were the llvm ocaml bindings and Jane Street’s Core standard library.

• **Git & Bitbucket**: For version control. Bitbucket was chosen instead of Github in order to use private repositories for free.

*Sublime or Vim*: Text editors. Each member used their preferred text editor, as this did not interfere with the group's overall coding.
5. Language Evolution (Jillian)

5.1. Introduction

We set out to create a language that was both functional and object oriented, inspired by Scala. Since our language is both object oriented and functional, the direction of the program is motivated by classes and functions.

Similar to a general object-oriented programming language, users can define main methods with classes and functions. The characteristically functional feature of this language is that variables can be defined as the result of calls to anonymous functions, denoted by the ‘@’ symbol. We choose the ‘@’ symbol to represent anonymous functions after also considering using the term ‘lambda’.

Functions contained within classes, functional arguments, and return types must be specified outside of the classes containing the definition of a nested function. Functions are only accessible from within the scope of the classes wherein nested functions are defined. In order to define functions within classes. Functions can be returned as objects from classes using the ‘->’ notation as the return type specifying both the nested function argument type on the left of the arrow and the nested function return type on the right side of the arrow. This notation was selected to allow writers of class objects to easily clarify to future code readers whether the class returns a variable object or a function object.

Users can create new data types with the ‘class’ keyword. These data types have public member variables that can be declared after the object is instantiated.

5.2. Structure

The program is structurally composed of both classes and functions. Functions can be defined within classes and classes can return functions as objects. User defined datatypes can be defined using ‘class’ whereas functions are defined using ‘def’. Functions can be defined within classes and returned as objects.

5.3. Variables

Though we had initially set out to create a type inferred language, we soon found this unfeasible in terms of our available time to complete the project. Thus we moved to a system of scala like variable declaration in the form: ‘var name:Type = <Literal | Function call’.
This syntax imitates the variable instantiation of most functional language. The general ‘var’ keyword accompanied by the ‘:<Type>’ declarator allows for the easy declaration of both primitive data types and user defined data types that were previously created within classes.

5.4. Function Definition

Functions can be defined in both a declarative and anonymous format. Function return types must be specified as part of the function declaration, using the “:<Type>” notation so that a function that takes no arguments but returns an int would be declared as ‘def return_int():Int{ <statements>}’. When a function is returned from a function this is denoted with the <ArgType> -> <ReturnType> of the returned function in place of a return type. Thus the arrow notation provides evidence that a function rather than another object is being returned.

We envision users generating anonymous functions for the purpose of computing repetitive operations, such as list manipulation as anonymous functions accomplish this task with very little code.

5.5. Class Definition

Classes are user defined datatypes. Much like in any object-oriented programming language, classes can be declared with multiple member variable fields. These fields must be assigned after the class object is instantiated

```java
def main = ():Int {
    var j:Rectangle;
    j.w = 5;
    j.h = 10;
    printf("%d %d\n", j.w, j.h); //prints 5 10
    return 0;
}

class Rectangle = {
    var w:Int;
    var h:Int;
}
```

An instance of the rectangle object is declared with the var j:Rectangle; command. However j’s member fields can be accessed using the ‘.’ operator and modified continuously after the object has been created. Thus class objects have fully mutable member fields.
6. Translator Architecture (James + Jonathan)

6.1. Overview

The compiler consists of the following parts:
- scanner.mll
- parser.mly
- ast.ml
- analysis.ml
- sast.ml
- codegen.ml
- exceptions.ml
- generator.ml
- util.ml
- Stop.ml

A high level overview our translator is given by the diagram below.

6.2. Scanner

The scanner is rather straightforward and it was generated by ocamllex.

6.3. Parser

Less straightforward, our parser is generated by ocamlyacc. Builds the ast from tokens and also will throw exceptions relating to syntax errors. We have a counter which keeps track of the line numbers so if a syntax error occurs we can locate the line in the program which caused it.

Parsing function types, and function literals (lambdas) was particularly challenging as it required us to set up a mutually recursive relationship between statements and expressions. Since a function literal is an expression which is defined by its parameters its return type and a list of statements to be executed.
6.4. Analysis

This is where most of the work occurs. Here we resolve all variable names, perform basic name mangling for anonymous functions. The major components which enable us to do this are a translation environment similar to the one shown on slide 65 of the Types and Semantics slide deck from lecture. We also have a hashtable which stores our access links which we use for implementing nested functions. We generate a class for each function declaration which has the fields every function instance will need. This is similar to how C++11 lambdas are implemented in that the compiler generates a closure class and every closure class has a different type. We call these generated classes record types but conceptually it is the same.

6.5. Utils

In the utils file we have a large collection of printing and debugging utilities. These were crucial tools for understanding our compilers output and also helped make up for the lack of ocaml specific debuggers. The two functions, string_of_program and string_of_sast were used for pretty printing our data structures for debugging purposes. The program is transformed between the ast and the sast, for example in the sast our record links are visible but in the ast they are not.

6.6. Code Generation

We tried to keep the amount of analysis being performed in codgen to a minimum to prevent bugs. Originally we tried to implement access links in codegen but several hundred segfaults later we decided to make sure structs worked and then leverage them to implement nested functions. Not every function in the codgen file is actually used, as there were some extra features like method calls which we ended up not getting done. Code is generated by traversing our sast and pattern matching to select the proper codegen function for each language element. Not that much checking is done here because that was taken care of in analysis.ml.
7. Test plan and scripts (Jonathan +James)

7.1. Testing Phases

Testing was done incrementally as each new feature was added. In addition to having automated testing, we also used utils.ml in order to print abstract syntax trees and semantically checked abstract syntax trees. This allowed for us to manually check each step of the compilation process and identify at which point of the process certain errors occurred.

7.2. Test Suites

All of our tests were put in a separate directory called tests. As mentioned before, tests were added each time new features were added, which also largely determined which test cases we chose. Most of our tests aim to cover different kinds of possibilities, including edge cases.

To see all of our tests, please refer to the code submission, under the directory ‘tests’.

7.3. Automated Testing

We created an automated regression test suite largely borrowed from the MicroC Compiler. The test cases are written out using the notation ‘test-<name>.stp’ and the corresponding output as ‘test-<name>.out’. The automated test suite compares the output produced by the two programs.

According to the principles of Test Driven Development, we wrote our tests prior to implementing the related features in our program. Most elements of our parser have a corresponding test to check the relating code generation.

The test script is executed with the ./testall command which will then display a list of tests that pass, fail, or produce a printed output that differs from the desired printed output. One alternative testing technique that we learned about towards the end of the project was examining the resulting .ll files produced by our test programs. The .ll files gave us new insight into determining where there might be logical errors in our code generation.

7.4. Sample Programs

Here are some sample stop programs with the target language program for each stop program.
7.4.1. test-fib_rec.stp

    /* test-fib_rec.stp */
    */

    def fib = (i: Int): Int {
        if (i == 0) {
            return 1;
        } else if (i == 1) {
            return 1;
        } else {
            return fib(i-1) + fib(i-2);
        }
    }

    def main = (): Int {
        printf("%d
", fib(10));
        printf("%d
", fib(9));
        printf("%d
", fib(8));
        return 0;
    }

The corresponding test-fib_rec.ll:

; ModuleID = 'Stop'

%fib.record = type <{ i32, i32 }>
%main.record = type <{ i32 }>

@fmt = private unnamed_addr constant [4 x i8] c"%d\n"
@fmt.1 = private unnamed_addr constant [4 x i8] c"%d\n"
@fmt.2 = private unnamed_addr constant [4 x i8] c"%d\n"

declare i32 @printf(i8*, ...)
declare noalias i8* @malloc(i32)
declare i32 @open(i8*, i32)
declare i32 @close(i32)
declare i32 @read(i32, i8*, i32)
declare i32 @write(i32, i8*, i32)
declare i32 @lseek(i32, i32, i32)
declare void @exit(i32)
declare i8* @realloc(i8*, i32)
declare i32 @getchar()

define i32 @fib(i32 %i) {
  entry:
    %malloccall = tail call i8* @malloc(i32 ptrtoint (%fib.record* getelementptr (%fib.record, %fib.record* null, i32 1) to i32))
    %fib_record = bitcast i8* %malloccall to %fib.record*
    %i1 = getelementptr inbounds %fib.record, %fib.record* %fib_record, i32 0, i32 1
    store i32 %i, i32* %i1
    %i2 = getelementptr inbounds %fib.record, %fib.record* %fib_record, i32 0, i32 1
    %i3 = load i32, i32* %i2
    %eqtmp = icmp eq i32 %i3, 0
    br i1 %eqtmp, label %then, label %else
  then: ; preds = %entry
    ret i32 1
    br label %ifcont15
  else: ; preds = %entry
    %i4 = getelementptr inbounds %fib.record, %fib.record* %fib_record, i32 0, i32 1
    %i5 = load i32, i32* %i4
    %eqtmp6 = icmp eq i32 %i5, 1
    br i1 %eqtmp6, label %then7, label %else8
  then7: ; preds = %else
    ret i32 1
    br label %ifcont
  else8: ; preds = %else
    %i9 = getelementptr inbounds %fib.record, %fib.record* %fib_record, i32 0, i32 1
    %i10 = load i32, i32* %i9
%subtmp = sub i32 %i10, 1
%tmp = call i32 %fib(%subtmp)
%i11 = getelementptr inbounds %fib.record, %fib.record* %fib_record, i32 0, i32 1
%i12 = load i32, i32* %i11
%subtmp13 = sub i32 %i12, 2
%tmp14 = call i32 %fib(%subtmp13)
%addtmp = add i32 %tmp, %tmp14
ret i32 %addtmp
br label %ifcont

ifcont: ; preds = %else8, %then7
br label %ifcont15

ifcont15: ; preds = %ifcont, %then
ret i32 0
}
define i32 @main(i32 %argc, i8** %argv) {
entry:
%malloccall = tail call i8* @malloc
(i32 ptrtoint (%main.record* getelementptr (%main.record,
%main.record* null, i32 1) to i32))
%main_record = bitcast i8* %malloccall to %main.record*
%tmp = call i32 %fib(i32 10)
%printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt, i32 0, i32 0), i32 %tmp)
%tmp1 = call i32 %fib(i32 9)
%printf2 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.1, i32 0, i32 0), i32 %tmp1)
%tmp3 = call i32 %fib(i32 8)
%printf4 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.2, i32 0, i32 0), i32 %tmp3)
ret i32 0
}

7.4.2. test-class2.stp

/*
 * test-class2.stp
 * ===============
 */
def main = ():Int {
    var j:Rectangle;
    j.w = 5;
j.h = 10;
printf("%d %d\n", j.w, j.h);
return 0;
}

class Rectangle = {
    var w:Int;
    var h:Int;
}

The corresponding test-class2.ll:

; ModuleID = 'Stop'
%main.record = type <{ i32, %Rectangle* }>
%Rectangle = type <{ i32, i32, i32 }>

@fmt = private unnamed_addr constant [7 x i8] c"%d %d\0A\00"
define i32 @main(i32 %argc, i8** %argv) {
    entry:
7.4.3. test-nested.stp

def main = ():Int {
    var j:Int = 3;
    var lambda_add_i = @(i:Int):Int {
        return i + j;
    };

    printf("%d\n", lambda_add_i(5));
    return 0;
}
The corresponding test-nested.ll:

; ModuleID = 'Stop'

%main.record = type <{ i32, i32 (%main.record*, i32)*, i32 }>
%"@1.record" = type <{ i32, i32, %main.record* }>

@fmt = private unnamed_addr constant [4 x i8] c"%d\0A\00"
declare i32 @printf(i8*, ...)
declare noalias i8* @malloc(i32)
declare i32 @open(i8*, i32)
declare i32 @close(i32)
declare i32 @read(i32, i8*, i32)
declare i32 @write(i32, i8*, i32)
declare i32 @lseek(i32, i32, i32)
declare void @exit(i32)
declare i8* @realloc(i8*, i32)
declare i32 @getchar()
declare i32 @sizeof(i32)

define i32 @"@1"(%main.record* "%"@1_@link", i32 %i) {
  entry:
    %malloccall = tail call i8* @malloc(i32 ptrtoint (%"@1.record"* getelementptr (%"@1.record", %"@1.record"* null, i32 1) to i32))
    %"@1_record" = bitcast i8* %malloccall to %"@1.record"
    %i1 = getelementptr inbounds %"@1.record", %"@1.record"* %"@1_record", i32 0, i32 1
    store i32 %i, i32* %i1
    %"@1_@link2" = getelementptr inbounds %"@1.record", %"@1.record"* %"@1_record", i32 0, i32 2
    store %main.record* %"@1_@link", %main.record** %"@1_@link2"
    %i3 = getelementptr inbounds %"@1.record", %"@1.record"* %"@1_record", i32 0, i32 1
    %i4 = load i32, i32* %i3

define i32 @main(i32 %argc, i8** %argv) {
  entry:
    %malloccall = tail call i8* @malloc(i32 ptrtoint (%main.record* getelementptr (%main.record, %main.record* null, i32 1) to i32))
    %main_record = bitcast i8* %malloccall to %main.record*
    %j = getelementptr inbounds %main.record, %main.record* %main_record, i32 0, i32 2
    store i32 3, i32* %j
    %lambda_add_i = getelementptr inbounds %main.record, %main.record* %main_record, i32 0, i32 1
    store i32 (%main.record*, i32)* %@1, i32 (%main.record*, i32)** %lambda_add_i
    %lambda_add_i1 = getelementptr inbounds %main.record, %main.record* %main_record, i32 0, i32 1
    %lambda_add_i2 = load i32 (%main.record*, i32)*, i32 (%main.record*, i32)** %lambda_add_i1
    %tmp = call i32 %lambda_add_i2(%main.record* %main_record, i32 5)
    %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt, i32 0, i32 0), i32 %tmp)
    ret i32 0
}
8. Conclusions & Lessons Learned

8.1. Jillian Knoll

I think that it is very important to get a grounding in Ocaml early on in the course by completing more complex practice problems and trying to modify the MicroC parser. I should have started learning Ocaml last semester after I decided that I was going to take this class. However once you are able to think in Ocaml you develop a new understanding about how to write amazingly efficient programs.

I learned that sometimes you need to make very extreme changes to the language when it is clear that one feature would take up all of your time to implement. We made several such decisions that enabled us to complete the project on time.

I would advise future groups to set an internal deadline of two weeks before the actual deadline and to enforce this deadline by scheduling more group work sessions. We found towards the end of our project that we were most productive in group work sessions as the group provides an important resource to turn to when asking questions or deciding on the best plan for implementing language features.

Lastly I have stubbornly admitted that I will not be able to ‘make’ our project on my macintosh computer without the aide of our virtual machine as there was something problematic about ocaml-llvm that I could not solve, even with my intermediate-advanced Stack Overflow skillset. Using virtual machines is much preferred for testing purposes. Additionally, we transitioned to using the Ocaml Core library rather than the standard library as the Core library has named parameters which makes function declaration much more intuitive for a former Java programmer.

8.2. Jonathan Barrios

The class and this project not only introduced me to principles of compiler design but also provided me with a greater understanding of functional programming languages and type systems. The nature of the features we were trying to implement forced us to deal with practical details of language design but also theoretical ones.

I am extremely proud of every member of my group. Everyone contributed and learned a lot. We are leaving this class with a much stronger understanding of the relationship between language design and program design.

When we were in early in the language design phase we promised ourselves that we would not make a compiler that compiled to C or javascript or sang and dance. We
wanted to gain a better understanding of how great languages and their features were implemented. This was unfamiliar territory for all of us so we of course ran into many issues. It goes without saying that having a better plan from the beginning would have improved our product.

8.3. James Stenger

Overall, this project proved both intellectually and logistically challenging and I very much enjoyed the time I spent working on it. Because of the scope of the project I would stress the importance of getting started early and establishing deadlines so as to avoid a last-minute crunch. It was also very helpful to determine the feature set of the language at the outset and then incrementally work towards getting everything functional. This last point is especially important since many advanced features require a lot of planning and infrastructure in different phases of the compiler (for instance, nested functions and semantic analysis). You need to figure out precisely how you want to integrate these features into the compiler in order to properly and efficiently implement them.

8.4. Lusa Zhan

The project was overall quite challenging, yet very rewarding at the same time. I believe that it did not only teach us a lot about the compilation process, but also about the logistics and team organization. In terms of the coding, I recommend starting to learn Ocaml early on and set up the environment for it as soon as possible as well. Ocaml seems intimidating at first, but once I became more familiar with it, it turned out to be much better than originally anticipated.

As the project manager, I believe that project planning was one of the bigger challenges, apart from actually creating the product. I recommend that future groups set a tight schedule and try to adhere to it as much as possible. That being said, I extremely enjoyed working with my group members. We all contributed to the code, split up the work, and did our best to reach our promised goals on time, which I greatly appreciate.
9. Commit History & Attribution

The Stop programming language was developed mostly within full group work sessions for design, architecture, and tutorial purposes followed by individual sessions to work on testing and bug fixes. Throughout our group work sessions we often participated in paired programming sessions, as Ocaml is a language that lends itself to having multiple sets of eyes attuned to generating code. We found that paired programming made our work sessions much more efficient than two group members working alone separately especially when implementing architectural decisions whereas writing codegen or analysis for already defined parser items can be accomplished sufficiently by a single code writer.

The commit history is below:

* commit 4fd1462b472b1f340c70c5ba918ef1f14bb1494e
  Author: James Stenger <jms2431@columbia.edu>
  Date: Wed May 11 21:09:07 2016 +0000
  stable; changed to not malloc

* commit 11848dd89228388d666357d6a4c36b355a547127
  Author: James Stenger <jms2431@columbia.edu>
  Date: Wed May 11 21:04:06 2016 +0000
  stable; fixed arrays per Lusa's change

* commit e939843308349c2a5c59345820c1341f482efd8b
  Author: Jonathan Barrios <jeb2239@columbia.edu>
  Date: Wed May 11 15:57:49 2016 -0400
  try to make this work

* commit e56f386cc52bf29fc133af6da406873525d7e47a
  Author: Jonathan Barrios <jeb2239@columbia.edu>
  Date: Wed May 11 15:48:48 2016 -0400
  fixed naming issue

* commit 5123fac20045e6d17d3e4083de209d59a22fb682
  Author: James Stenger <jms2431@columbia.edu>
  Date: Wed May 11 19:03:02 2016 +0000
stable; nested functions functional - returning functions

* commit d5dc5f501c30911562be5f36b8673af1bfed40f7
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Wed May 11 18:53:33 2016 +0000

  stable; roadmap2 functional & working (access vars outside of function definition); but still
  writing that functionality so only works up one layer

* commit ec78ebff49d94591bdac408999d2e1d4e153e8cf
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Wed May 11 16:54:44 2016 +0000

  stable; added roadmaps; now passing access links; TODO: resolve vars named outside of
  scope to access accesslinks to reach them (in semant); also roadmap3 as stretch goal

* commit 6372654bf1c9f7e72aa96cf53575c1a44a46bdef
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Wed May 11 16:35:28 2016 +0000

  stable; resolved function definition/assignment issue (ftype needed to be same, args were in
  reverse order in codegen llttype of ftype

* commit 9991d64d5a79c5f88d01df2c03f101c423f77ede
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Wed May 11 14:19:46 2016 +0000

  saved stuff

* commit e78d1db799dec6ffec308dc2f03725c4b122d5c5c
  Author: Jonathan Barrios <jeb2239@columbia.edu>
  Date:   Wed May 11 15:33:12 2016 -0400

arrastuff

* commit 346f3e3a6b17695aab7fa6037c7aa97aab7826e0
  Author: Jonathan Barrios <jeb2239@columbia.edu>
  Date:   Wed May 11 13:45:55 2016 -0400

  still trying to get array access working, see temp- test.stp

* commit f13ab1e0272e93e97705b2e1e435e81879cb7db9
  Author: Jillian Knoll <jillianknoll@users.noreply.github.com>
array access

* commit 0c483c9149f589ae9282af47399318920ef3424b
  Author: Jillian Knoll <jillianknoll@users.noreply.github.com>
  Date:   Wed May 11 12:40:40 2016 -0400

array access checks

* commit 8b3ce98bd6b9c0fe4e51caf709c7ac419e7ede36
  Author: Jonathan Barrios <jeb2239@columbia.edu>
  Date:   Wed May 11 10:36:48 2016 -0400

hi

* commit e5bddd49c13f40293f4cc5efc585d4eafe3e4e587
  Merge: 55e8038 8c665e5
  Author: Jonathan Barrios <jeb2239@columbia.edu>
  Date:   Wed May 11 02:12:05 2016 -0400

merged everybody into object_access

* commit 8c665e5f3e1f3a3b095ccd27d16ae144e884a361
  Merge: f52e5f2 a864752
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Wed May 11 01:45:57 2016 +0000

unstable; resolved merge conflict

* commit a8647527166a2cfb5b3b0460ffe9ca100678a5f0
  Author: Lusa <lusa_zhan@yahoo.de>
  Date:   Wed May 11 00:52:34 2016 +0000

not working: float not stored

* commit 1ce1a3a2a7ce67e05a8322014f150ecf0e5ec1bb
  Author: Lusa <lusa_zhan@yahoo.de>
  Date:   Tue May 10 23:21:43 2016 +0000

float binary ops test - fail

* commit 55e8038bb264169e9f94a954f75b019906265105
  Author: Jonathan Barrios <jeb2239@columbia.edu>
added more printers for debug

* commit bcd0fce8fd98c961f76dc77c47e69d505c0fd72a
  * Author: Jillian Knoll <jillianknoll@users.noreply.github.com>
  * Date: Tue May 10 20:54:06 2016 -0400

float output

* commit 087f67d4517f985bf5e4031bb0e8a7527a6e9df9
  * Author: Jillian Knoll <jillianknoll@users.noreply.github.com>
  * Date: Tue May 10 19:44:12 2016 -0400

testing for float array instantiation and output

* commit a334a2d758a7956e40c6e25d01b0bfe252b65a26
  * Author: Jonathan Barrios <jeb2239@columbia.edu>
  * Date: Mon May 9 03:50:06 2016 -0400

methods are making it thru Sast next need to handle codegen of method calls

* commit f711c6f5da5250f6ade9288198c8b3b3b6f01000
  * Author: Jonathan Barrios <jeb2239@columbia.edu>
  * Date: Sun May 8 21:05:02 2016 -0400

segfaulting at line 276

* commit 63b603d4736db2eac727f568bf7f2c0f62c9adeb
  * Author: Jonathan Barrios <jeb2239@columbia.edu>
  * Date: Sun May 8 19:24:58 2016 -0400

need to make it so that the name in nested structs recursivley build themselves correctly

* commit 517d88be408931df3696a8bd3855e5c5d296246c6
  * Author: Jonathan Barrios <jeb2239@columbia.edu>
  * Date: Sun May 8 18:45:30 2016 -0400

stop segfaults alot but the sast is printing some stuff which may be helpful

* commit d6c6dccad0b2ee313323c3d1d43f21bddd0aa566d
  * Author: Jonathan Barrios <jeb2239@columbia.edu>
  * Date: Sun May 8 17:35:12 2016 -0400
compiles but does not work, need to modify codegen of object access

* commit 6997ddecc54e626990dbd036f772712bd42f722a
  Author: Jonathan Barrios <jeb2239@columbia.edu>
  Date: Sun May 8 15:54:21 2016 -0400

  this will not compile

* commit c08cf554e9b314d7d93c2ca9048c93fe7fb5b50d
  Author: Jonathan Barrios <jeb2239@columbia.edu>
  Date: Sun May 8 00:51:38 2016 -0400

  crude string_of_env for debugging sast

* commit 46d4e0ad828cdf4d96fb705d1ed5d9d4f7e3ac5
  Author: Jonathan Barrios <jeb2239@columbia.edu>
  Date: Sun May 8 00:08:14 2016 -0400

  added string_of_classrecord

* commit 67ee0104a5886eea6a9aba4e950419d9ef90455
  Author: Jonathan Barrios <jeb2239@columbia.edu>
  Date: Sat May 7 15:03:48 2016 -0400

  added sizeof

* commit 7afdda3f51556ecf9f1a2db07e99d4cede8ec94af7
  Author: James Stenger <jms2431@columbia.edu>
  Date: Wed May 11 14:31:26 2016 +0000

  trying to resolve function assignment issue that should be working; thought was due to merge but not the case

* commit f52e5f2b3ff55d57e8d050f2e0be474f3e9d89d8
  Author: James Stenger <jms2431@columbia.edu>
  Date: Wed May 11 01:39:55 2016 +0000

  test updates

* commit 5b7a826431870afed8ec592d6f03d5b5b512b4b
  Author: James Stenger <jms2431@columbia.edu>
  Date: Sat May 7 16:28:24 2016 +0000

  stable
* commit 3bc566bcb07c19cade0e7b094962ce142d1945d5
  | Author: James Stenger <jms2431@columbia.edu>
  | Date:   Sat May 7 16:13:59 2016 +0000
  |
  | stable; added calling logic for passing records
  |
  * commit dc2addf21adfe4121a3b8f7c542496fbafeec89c6
  | Author: James Stenger <jms2431@columbia.edu>
  | Date:   Sat May 7 06:05:39 2016 +0000
  |
  | unstable; asymptotically close to finishing nested functions: needed to add access link parameter to record structs and add access resolution in analysis
  |
  * commit 4f9758ae66ba7cd264f2ecf904d04dd02bd10aace
  | Author: James Stenger <jms2431@columbia.edu>
  | Date:   Sat May 7 05:25:38 2016 +0000
  |
  | stable; nested functions functional, everything using activation records properly
  |
  * commit 0fbcc46a999f56613d403f3487fcej07837445a8
  / | Author: James Stenger <jms2431@columbia.edu>
  | Date:   Sat May 7 04:58:07 2016 +0000
  |
  | stable; var function calls are now working again
  |
  * commit 8e28fd122100c0886697f016c63b028b13165cb4
  | Author: James Stenger <jms2431@columbia.edu>
  | Date:   Sat May 7 04:18:28 2016 +0000
  |
  | stable; resolved object allocation issues
  |
  * commit 14c1d1d8e9ba7e5076fb2bf9a17e1b4552a3a76
  | Author: James Stenger <jms2431@columbia.edu>
  | Date:   Sat May 7 02:18:07 2016 +0000
  |
  | unstable; test-temp issue needs to be resolved before progressing (note we're trying to access record.b.a-- what is the problem?
  |
  * commit bffefb55ecbfe50d706f10f9825a1b81892c6417
  | Author: James Stenger <jms2431@columbia.edu>
  | Date:   Sat May 7 01:57:18 2016 +0000
  |
unstable; need to resolve issue with test-temp before finishing nested functions (access struct through struct)

* commit 59cc3a6537d8e4e6a9434b4659c0e76eccc92a06
  Author: Lusa <lusa_zhan@yahoo.de>
  Date: Fri May 6 23:25:07 2016 +0000
  break, continue working now

* commit 09695d08e116d95fb20e330db87daa736ef631
  Author: James Stenger <jms2431@columbia.edu>
  Date: Fri May 6 22:09:38 2016 +0000
  unstable; most features now using records (need to add for calls, object access)

* commit e2aaefe649e11c43341e55e6151929a4cf5f42f6
  Author: James Stenger <jms2431@columbia.edu>
  Date: Fri May 6 21:54:53 2016 +0000
  trying to convert locals to access record instead of new allocs

* commit e7f44af10e3f8d92e67278ccf29ef548cc76e60
  Author: James Stenger <jms2431@columbia.edu>
  Date: Fri May 6 21:33:39 2016 +0000
  stable; all functions now allocate an activation record on start

* commit e8607c7e7125c67aa929353857e3e757cdd229c
  Author: James Stenger <jms2431@columbia.edu>
  Date: Fri May 6 21:16:59 2016 +0000
  stable; analyzing nested functions nearly-correctly

* commit 6c985e529e9e85c2628045a25355efee9f8e28448
  Author: James Stenger <jms2431@columbia.edu>
  Date: Fri May 6 21:06:34 2016 +0000
  stable; analyzing nested functions nearly-correctly

* commit 34d15dce91cb94782d8c538824daefcabe
  Author: James Stenger <jms2431@columbia.edu>
  Date: Fri May 6 19:54:26 2016 +0000
stable; nested2 functional but using vals outside of scope will require a bit more work

* commit 78742bbfbb7fe07408827e2610004cdafl4ded3b
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Fri May 6 19:50:43 2016 +0000

  fixed higher-order function buildup bug in analysis

* commit 1d06f649ca989bf31181f792a7b4fcb90e836924
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Fri May 6 19:43:19 2016 +0000

  stable; nested functions functional, not yet using activation records so named vals in upper scope not accessible

* commit 613ee19891bbdb19bf4c62dedab658e2746ff5ae
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Fri May 6 18:37:59 2016 +0000

  stable; augmented sast printing functions; resolved function call of function var analysis issue; added nested test

* commit f3db0a85620a7d7fc1b9867860006ae96add595a
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Fri May 6 06:38:52 2016 +0000

  added nested function test prog

* commit 6c325ac39b765e138609e90955c9587a6351b995
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Fri May 6 06:29:44 2016 +0000

  resolved increment test issue (added ++ -- parsing)

* commit 7421a561e055b6e76215e61c502e4880b2f0784d
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Fri May 6 06:09:24 2016 +0000

  revised controlflow8; note parse failure -> not language feature (C for instance requires variable declaration outside of loops)

* commit 3020856535b7bbec0bfb2d8c7243c4d9f179f4e2
  
  Merge: 43ed741 68d5d89
  
  Author: James Stenger <jms2431@columbia.edu>
Merge branch 'compiler_updates_everyone' of https://bitbucket.org/nottrainwreck/stop into compiler_updates_everyone

* commit 68d5d89f9d9a5d93a2ce8a17bbf248e6da0342f3
  Author: Lusa <lusa_zhan@yahoo.de>
  Date: Fri May 6 04:07:00 2016 +0000

  adding test for multidim array

* commit b777fc8735495f765a38e94cb7b9525171a502
  Author: Jonathan Barrios <jeb2239@columbia.edu>
  Date: Thu May 5 23:56:22 2016 -0400

  hooray for array

* commit 838d1ccc377d891188f031b5e74a1f16b9652cae
  Author: Lusa <lusa_zhan@yahoo.de>
  Date: Fri May 6 03:13:10 2016 +0000

  array goes through analyzer, only codegen missing now

* commit 43ed74157f2f17f1fb89e2c57028f93a112e04fa8
  Author: James Stenger <jms2431@columbia.edu>
  Date: Thu May 5 21:12:57 2016 +0000

  minor changes

* commit 8abd38d59ba394785338cc02cb2757596d598edc
  Author: Jillian Knoll <jillianknoll@users.noreply.github.com>
  Date: Fri May 6 00:37:31 2016 -0400

  fixed signed integer arithmetic test

* commit e7835b8bd6d537a30d3ce2355c77af70987dc376
  Author: Jillian Knoll <jillianknoll@users.noreply.github.com>
  Date: Thu May 5 23:40:20 2016 -0400

  added arithmetic test

* commit bdfa92784b1ea818c0813b61d276f77ec9afcc
  Author: Lusa <lusa_zhan@yahoo.de>
  Date: Fri May 6 02:21:05 2016 +0000
fixed a couple bugs from before

* commit 020b7e896dbace81f72e6e3fbd0526e2c0d96e67
// Author: Lusa <lusa_zhan@yahoo.de>
Date:   Fri May 6 00:30:23 2016 +0000

added array create to parser, ast, utils

* commit 78b7555f1f82f3b0467ecb4e5c72b53ae1627336
Author: Lusa <lusa_zhan@yahoo.de>
Date:   Thu May 5 20:00:39 2016 +0000

added array tests to array directory in tests

* commit c51c756e9676be808a42e2229410e1fadb160310
Author: James Stenger <jms2431@columbia.edu>
Date:   Thu May 5 19:56:08 2016 +0000

stable; object access actually functional

* commit 061214632dd7f026aa46aa7e91cfeaa21742712a8
Author: James Stenger <jms2431@columbia.edu>
Date:   Thu May 5 19:45:17 2016 +0000

stable; object access somewhat functional

* commit 0ea1fcfcec4916ce4f3178eeb449edcbe86e2c3a
Author: James Stenger <jms2431@columbia.edu>
Date:   Thu May 5 19:40:58 2016 +0000

accessing objects functional

* commit ae3ea4eaa154a39f51855ae11bb861758290b7e
Author: Lusa <lusa_zhan@yahoo.de>
Date:   Thu May 5 19:26:30 2016 +0000

more tests. increment & for(var i:Int=0;..) not working

* commit 4e11be1f04f3d620805627427f6adb5334a9e46b
Author: James Stenger <jms2431@columbia.edu>
Date:   Thu May 5 18:57:21 2016 +0000

stable; working on object & array declaration & accessing
* commit 1a48a9378aa12ab08f4e7ba5b0093b8e3f0b10ad
  Author: Lusa <lusa_zhan@yahoo.de>
  Date:   Thu May 5 18:45:25 2016 +0000
  
  added some tests, not working: controlflow5 and controlflow 6

* commit d547fe22d50275e6a807a1ef543842e72343f084
  Author: Jonathan Barrios <jeb2239@columbia.edu>
  Date:   Thu May 5 14:10:08 2016 -0400

  x86 for test-hang

* commit 76901c1f0b3f1017ddb7e70f6336118dd2619756
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Thu May 5 18:05:08 2016 +0000

  stable; added hang and no_hang problem programs

* commit 67cd77154e111b07d656238f5181eada8f088b72
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Thu May 5 16:24:48 2016 +0000

  stable; commented out fib_dynamic: currently hangs

* commit 5e59ebc6e05a4e1f5fa96bc1efdf299fb91e4beb
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Thu May 5 16:23:27 2016 +0000

  revised test script; now more useful

* commit e1ea637552c469988df3058214f8e1e524ff5e0b
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Thu May 5 16:07:02 2016 +0000

  added unop codegen support

* commit 757a84cb0898aeb4885474e056564f54aa2f0678
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Thu May 5 15:41:31 2016 +0000

  tried to resolve return type issue but having strange error; marked with TODO since still functional without resolve
* commit 3fb43d255f8fae8f77280248a4c570184891eb5e
  Author: James Stenger <jms2431@columbia.edu>
  Date: Thu May 5 15:23:09 2016 +0000
  unstable (issue with fib_rec); resolved arrayaccess type issue

* commit 456e7aad543c8ca1c421c3e15ac378da6fdbe97
  Author: James Stenger <jms2431@columbia.edu>
  Date: Thu May 5 15:06:55 2016 +0000
  stable; added arrayaccess util printing

* commit ef3ca6b089c5d691f466a7ddced20d0f12ef6538
  Author: James Stenger <jms2431@columbia.edu>
  Date: Thu May 5 14:59:09 2016 +0000
  resolved ast generation stmt order issue

* commit 5ad8d23990be3c27b8c9017a0290e5b27d1f383d
  Author: James Stenger <jms2431@columbia.edu>
  Date: Thu May 5 05:37:26 2016 +0000
  stable; roadmark before refactoring nested functions

* commit 9dfa82b91aa8b9076710dd17f86eb3603d5bc899
  Author: James Stenger <jms2431@columbia.edu>
  Date: Thu May 5 04:56:57 2016 +0000
  stable; added previously omitted while semant

* commit 2d531630dcef43a51cc711ec16d803aba9f1d7cd
  Author: James Stenger <jms2431@columbia.edu>
  Date: Thu May 5 04:50:31 2016 +0000
  added all control flow stmts to semant + codegen

* commit 899da272207c686524f524ec3a1c3b09e2645f27
  Author: James Stenger <jms2431@columbia.edu>
  Date: Thu May 5 04:44:06 2016 +0000
  added for semant + codegen

* commit 511dcd66fb11fbc7ce9e9665df48651571ad97ab
  Author: James Stenger <jms2431@columbia.edu>
added tests

* commit cdee173bd6fb4294d2fe5593534d369313d27b09
  Author: Jonathan Barrios <jeb2239@columbia.edu>
  Date:   Thu May 5 15:57:24 2016 -0400
  parsing specs, need to add to sast

* commit c73f43f5a0497a0c23cd52c8116099cc1b26b3e1
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Thu May 5 03:12:10 2016 +0000
  added tests; fixed if with no else issue

* commit c45b65471e9f18ee9ab8d3a40626ac1d91cc074c
  Author: Jonathan Barrios <jeb2239@columbia.edu>
  Date:   Wed May 4 20:25:55 2016 -0400
  will branch from here

* commit 5e0310bc0d492d64ad0bae31f276833f58e9fcede
  Author: Jonathan Barrios <jeb2239@columbia.edu>
  Date:   Wed May 4 20:15:31 2016 -0400
  based on the output of printing the sast it seems that all of the SObjectAccess don't contain a proper object

* commit 3a021b920058a5d8720e8784a597272355c7a8ea
  Merge: e06fe5 e0b800e
  Author: Jonathan Barrios <jeb2239@columbia.edu>
  Date:   Wed May 4 19:04:50 2016 -0400
  merged some stuff I forgot to merge

* commit e0b800ea331e8c84b0b7f51280afe0e941012607
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Wed May 4 20:59:51 2016 +0000
  added desired testall output

* commit 7e97219e0824ce5d2e93c8635236194554016f90
  Author: James Stenger <jms2431@columbia.edu>
Date: Wed May 4 20:55:23 2016 +0000

stable; minor revisions (no added functionality); added tests to indicate what needs to be developed

*  commit e06fef5944ef24e9b32b7a32eb2bae5b2b94e3d5
   Merge: 5d772aa a84c5b0
   Author: Jonathan Barrios <jeb2239@columbia.edu>
   Date: Wed May 4 18:53:19 2016 -0400

  merged new james features, passing fname around, change if so that it works, this will compile

* commit a84c5b047933204c32e69226c832dd1a68e2d978
   Author: James Stenger <jms2431@columbia.edu>
   Date: Wed May 4 03:50:59 2016 +0000

  stable; functions now using heap-allocated activation records

* commit 41462263e902c1c12c8a72853df22d6407b0a4de
   Author: James Stenger <jms2431@columbia.edu>
   Date: Wed May 4 02:53:27 2016 +0000

  test

* commit 5d772aa521b1460465c5b686a544f064078a8a9e
   Author: Lusa <lusa_zhan@yahoo.de>
   Date: Wed May 4 06:02:50 2016 +0000

  added if & test for if

* commit da5253ccfd833117d5f1613189b0322fc39cf448
   Author: Jonathan Barrios <jeb2239@columbia.edu>
   Date: Wed May 4 00:39:29 2016 -0400

  string_of_sprogram, testing merged james changes

* commit 8f09db42ea707ac6fad7476c6a6a51946b7dbf15
   Merge: b6da64b da7f911
   Author: Jonathan Barrios <jeb2239@columbia.edu>
   Date: Wed May 4 00:03:42 2016 -0400

  Merge branch 'compiler_updates_jeb' into compiler_updates_everyone
* commit da7f911fe1788343c631903bec14a00d91aec4e2
  Author: Jonathan Barrios <jeb2239@columbia.edu>
  Date:   Tue May 3 23:52:24 2016 -0400
  changes

* commit c2ae5feaf00f9d45f6b510074c7e55bf4ac20b91
  Author: Jonathan Barrios <jeb2239@columbia.edu>
  Date:   Tue May 3 11:08:31 2016 -0400
  string_of_sprogram works and changed test-class1.stop to reflect correct class syntax

* commit b6da64b44a68fd754eca3ce8416f9daac5f602f8
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Tue May 3 22:35:11 2016 +0000
  unstable; function activation records now being generated

* commit 97d827f4d9ccee7646938597ba7e1dedfc4065cf3
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Tue May 3 18:48:06 2016 +0000
  unstable; correctly getting vars for each function record definition

* commit 4dc8eab5e3cba769d769a22ce7e770703cd6ce6f6
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Tue May 3 06:16:34 2016 +0000
  first order function calls & def. functional (?)

* commit 7d09a4033aa9732e5181228f14e027166fbd3f0e
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Tue May 3 05:58:17 2016 +0000
  stable; getting segfault on test-temp

* commit efc3eb688d2706a99cad12d13820d56564d4068e
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Tue May 3 04:46:37 2016 +0000
  stable; resolved semantic analysis reserved functions issue

* commit 8e284a4033225115d364ce4d2a587ab26f95aa96
  Author: James Stenger <jms2431@columbia.edu>
stable; test-temp pushed through semantic analysis; free to start work on higher-order function codegen

* commit 2c2d3f89fd804e811e985d9f29e436ad008da8cf
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Tue May 3 01:37:58 2016 +0000
  actually removed .swp files...

* commit ea006baec5bf46c5db9524d4f81e07251123e
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Tue May 3 01:36:33 2016 +0000
  removed swp files that shouldn't have been committed

* commit 1c8fab7f11d352b878ecef99ae0841a464935a2f0
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Tue May 3 01:26:50 2016 +0000
  added missing gitignore

* commit 299ab3c98fbc98d753e61fbdec5b77538382be4dd
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Tue May 3 01:14:19 2016 +0000
  unzipped src; branch should be compiling properly

* commit 858d8d34308993837e680b7b42eeb0c7dbea4086
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Tue May 3 01:13:02 2016 +0000
  unzipped src

* commit ce6579175a24907f71eb77f6ad792dd849a246a
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Tue May 3 01:12:09 2016 +0000
  trying to resolve commit issues

* commit 35d3f95004e69045c02e59988f58d6ebb
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Tue May 3 01:10:48 2016 +0000
escape commit

* commit d4ca48b4f424f60b253a4c29876b01e81f429966
  Author: Jonathan Barrios <jeb2239@columbia.edu>
  Date: Fri Apr 8 06:11:01 2016 +0000

  README.md edited online with Bitbucket

* commit 95f31e5f2e9fd6a1533eaaa8bbee5073ce0030fd
  Author: Jonathan Barrios <jeb2239@columbia.edu>
  Date: Mon May 2 22:32:25 2016 -0400

  matching james files structure

* commit 8ca451505b0717f3838ba9e768766ead9a96dbf
  Author: Jonathan Barrios <jeb2239@columbia.edu>
  Date: Mon May 2 18:50:33 2016 -0400

  .

* commit 4a1f5896743a9f261c9a9d9ed0be66711643d4a
  Author: Jonathan Barrios <jeb2239@columbia.edu>
  Date: Sat Apr 30 17:42:14 2016 -0400

  string_of_sast, still not done

* commit 72aad427f8fd55e8f841f3e0e941a78db618c39
  Merge: 9b3b3705fa507e
  Author: Jonathan Barrios <jeb2239@columbia.edu>
  Date: Sat Apr 30 13:04:18 2016 -0400

  merged in james changes, testing string_of_sast

* commit 9b3b370fa306f593d8901c9d6eb39e736635ba4
  Author: Jonathan Barrios <jeb2239@columbia.edu>
  Date: Fri Apr 29 13:06:57 2016 -0400

  compiles but can't get anything through it

* commit c195f1516ed434cf6bf0eae1e478f7bd60c794310
  Author: Jonathan Barrios <jeb2239@columbia.edu>
  Date: Fri Apr 29 12:55:56 2016 -0400
compiles string_of_sprogram but need to test it

* commit 0b140867ab525c8df3ab2672d6ebde3d64341292
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Tue May 3 00:45:49 2016 +0000

  attempted fix

* commit a594dd8e83337bb94ff85d6e3ea50ccc31840ced
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Tue May 3 00:43:46 2016 +0000

  clear

* commit dab91efeb39b3e067623c7a4c6673c5290282ca2
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Tue May 3 00:29:08 2016 +0000

  added correct tests folder

* commit ffd2fc1871c6ee44b3b20344c3a2b90216808051e
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Tue May 3 00:28:28 2016 +0000

  removed tests folder

* commit d09b0aa5b7e67b71296ac6aefc7919ccde812bf8
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Tue May 3 00:21:40 2016 +0000

  trying to resolve commit issue

* commit b8fa2c42790580f3667406f2e57f2787d5767c83
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Mon May 2 22:43:08 2016 +0000

  stable; variety of minor changes & fixes in preparation for getting higher-order functions
  working properly

* commit 4824e96cca8155c1b30ca509b13bf5b0c3bc8a95
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Sat Apr 30 21:39:26 2016 +0000
| stable; organized code and adjusted makefile appropriately; consolidated clean_tests script into makefile, now $ make clean_tests; fixed stop.ml so as to generate stop.native executable which outputs to the appropriate output channel; fixed testall and parse scripts, now both are functional again

| * commit bdc1095fde29e96a3eaca4f7f34b6b52e187f3f7
| Author: James Stenger <jms2431@columbia.edu>
| Date: Sat Apr 30 16:05:07 2016 +0000

| stable; expanded semantic analysis and codegen

| * commit 5fa507ecceadf297e2589d708b9ce92548ac97d4
| Author: James Stenger <jms2431@columbia.edu>
| Date: Sat Apr 30 02:29:07 2016 +0000

| stable; expanded codegen; now handling returns properly

| * commit 8df30bf799c5c70c781d675ea5bb18c2b2191dc6
| Author: James Stenger <jms2431@columbia.edu>
| Date: Fri Apr 29 16:12:47 2016 +0000

| stable; pulled in jeb Core updates; starting class codegen; updated branch to less generic name

| * commit c9d484861562d0a3a87110fb51bf48be4b54fe63
| Author: Jonathan Barrios <jeb2239@columbia.edu>
| Date: Fri Apr 29 01:09:31 2016 -0400

| this will not compile, working on string of sprogram

| * commit 0ecbb116d7e462f3a2b78cf0bd52e087720b4b5d
| Author: Jonathan Barrios <jeb2239@columbia.edu>
| Date: Fri Apr 29 00:01:06 2016 -0400

| also updated james branch to use core, will start string_of_ast from here

| * commit 1f6698e449d021d18c7787f9ea5cee6cee26f825
| Author: James Stenger <jms2431@columbia.edu>
| Date: Fri Apr 29 02:28:25 2016 +0000

| stable; now analyzing functions and methods properly

| * commit 306a15b391a07b9890daac1b88bbd631613bb4e7
| Author: Jonathan Barrios <jeb2239@columbia.edu>
Date: Thu Apr 28 23:35:41 2016 -0400

this compiles using Core.Std, made main and option type

* commit 49e58ba9078501f83798114e5fb0da2ada5b3cf6
  Merge: 154171d 0eb1282
  Author: Jonathan Barrios <jeb2239@columbia.edu>
  Date: Thu Apr 28 22:54:03 2016 -0400

  WIP on post_refactor: 154171d this compiles don't hurt me, starting sast and string_of_sprogram

* commit 0eb1282cc345b9f9780dba834ecdaa69a2abaf12
  Author: Jonathan Barrios <jeb2239@columbia.edu>
  Date: Thu Apr 28 22:54:03 2016 -0400

  index on post_refactor: 154171d this compiles don't hurt me, starting sast and string_of_sprogram

* commit 154171df907b553f637d45be058b8e035fde37ff
  Author: Jonathan Barrios <jeb2239@columbia.edu>
  Date: Thu Apr 28 20:24:42 2016 -0400

  this compiles don't hurt me, starting sast and string_of_sprogram

* commit 4bcd12f37c66b93b055b11d46951b1853e12e3e
  Author: James Stenger <jms2431@columbia.edu>
  Date: Sun Apr 24 15:40:23 2016 +0000

  removed unneeded filepath.ml file

* commit f18b9c0225863b528099a27dc578e229050471a9
  Author: James Stenger <jms2431@columbia.edu>
  Date: Sun Apr 24 15:39:32 2016 +0000

  parsing errors functional; semantic check phase incomplete (most match cases not implemented) but producing type-valid sast; codegen refacto just begun but is taking correct sast type; much TODO (but TODO.txt outdated)

* commit 455d6b3e628a5dc06eb35aa9a58ecc2143945112
  Author: James Stenger <jms2431@columbia.edu>
  Date: Sat Apr 23 00:53:02 2016 +0000

  removed files that shouldn't have been committed
* commit 46c47cde5f06866b63772bde379e04c7101a7263
  Author: James Stenger <jms2431@columbia.edu>
  Date: Sat Apr 23 00:45:34 2016 +0000
  major changes; presently mid complete refactor ala Dice; parsing of most language features functional; TODO: Semant -> Sast (see dice); Refactor Codegen; Complete Codegen

* commit 12f18ca8e4d1a79640f86fac4bdeeb9cccb135338
  Author: Jonathan Barrios <jeb2239@columbia.edu>
  Date: Fri Apr 22 20:28:44 2016 -0400
  forced on parser, but broke codegen obviously, trying to fix codegen

* commit 1a850b8d9d0164dab7c11fbd6bc23c1bab79a9f
  Author: Jonathan Barrios <jeb2239@columbia.edu>
  Date: Fri Apr 22 20:20:07 2016 -0400
  stuff which does not work

* commit 37e51b7bbec28bf62c440a3bc0475c6fa8d56f6e3
  Author: Jonathan Barrios <jeb2239@columbia.edu>
  Date: Wed Apr 20 12:25:02 2016 -0400
  hey

* commit b6ec0f39383f163ecc769ab4c07cadf1e71904fa
  Author: Jonathan Barrios <jeb2239@columbia.edu>
  Date: Sun Apr 17 20:02:16 2016 -0400
  don't push

* commit d7a3e0b23052af95bbc6acdbb897bfd617637d56
  Author: Jonathan Barrios <jeb2239@columbia.edu>
  Date: Sun Apr 17 19:32:30 2016 -0400
  again please don't push this

* commit d77e6787e6f5200d61807b886f4dda3d43ecdafb
  Author: James Stenger <jms2431@columbia.edu>
  Date: Sun Apr 17 21:47:26 2016 +0000
  refactored object_t as Objecttype in datatype in ast
* commit 162c8844b5211c36e4b0a383ecbf24e491b81561
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Sun Apr 17 20:52:18 2016 +0000
  added barios scanner changes; updated tests

* commit 3ffcf25119f4b6357c90d93c41b05fe8e5261777
  Merge: 29c29d9 afc5de9
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Sun Apr 17 20:10:01 2016 +0000
  Merge branch 'stop_compiler' of https://bitbucket.org/nottrainwreck/stop into
  stop_compiler

* commit afc5de9341871f849e7cf976ee70fa71ca269e95
  Author: Jonathan Barrios <jeb2239@columbia.edu>
  Date:   Sun Apr 17 16:09:48 2016 -0400
  i fixed what ever i did

* commit 29c29d904f60b15c26e9efe409bd6111662be7b3
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Sun Apr 17 20:02:10 2016 +0000
  Reverting back one commit
  Revert "added barrios branch files"
  This reverts commit 6b4e99633d072deabfb104cf931a331eba68b13a.

* commit c635f21bec307be0032f12282e81199c72a88276
  Author: Jonathan Barrios <jeb2239@columbia.edu>
  Date:   Sun Apr 17 15:51:23 2016 -0400
  added new Makefile

* commit 6b4e99633d072deabfb104cf931a331eba68b13a
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Sun Apr 17 19:44:29 2016 +0000
  added barrios branch files

* commit a31eb8242244178f4a56bb01ee73b3e0dd838223
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Fri Apr 15 20:58:54 2016 +0000
codegen for stmts stable

* commit a5dcbec913db167519d232b862ab60271e99513b3
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Fri Apr 15 20:20:14 2016 +0000

encapsulated several expr codegen functions

* commit beda0e59d0290391c8668977445df0422353bca6
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Fri Apr 15 20:15:16 2016 +0000

encapsulated printf codegen

* commit 98151246529a47878e5eae03d732e066997c48f
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Fri Apr 15 20:02:04 2016 +0000

added expr codegen % Ids; stable before encapsulating printf codegen

* commit 60bf8b56bb8489e0e798cd1745a3fa325204cc01
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Sat Apr 2 22:40:17 2016 +0000

helloworld functional

* commit 76bd267a3f3a332f9643aa1bf9d276fexc8639827
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Sat Apr 2 22:37:10 2016 +0000

printf working with multiple args

* commit 7c4e2b6b9b313008e7d5d5abac80049b574a087c
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Sat Apr 2 22:25:36 2016 +0000

printf almost at full functionality; revised tests to work w/ format strings; still need multiple args, strings

* commit 74419f581b1a0ed3ed076f4cf819432b926aa809
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Sat Apr 2 22:13:24 2016 +0000
progress towards full printf

* commit 40c1c213edeb777057b7275d1b79856d1631c818
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Sat Apr 2 21:52:26 2016 +0000

  stable before printf func call updates

* commit 723ceedc48db15e3a9511277dd8bb561ffcc0b87
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Sat Apr 2 20:04:23 2016 +0000

  resolved match error (matching with void_t not tenable yields errors)

* commit 12b2e8105104559d0a9d60cee220f317be543da0
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Sat Apr 2 19:49:53 2016 +0000

  stable commit pre-strings

* commit 6eb713457389bbf4f0cc947694fe48638a8f00a7
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Sat Apr 2 18:10:05 2016 +0000

  added some expr features; matching against ltypes causes problems needs fix

* commit f654ebe8ea11d5fe728f0695ab3c013f3c76db3
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Sat Apr 2 17:56:30 2016 +0000

  resolved issue with ltypes pattern matching; solution noted in codegen.ml

* commit fc624c144bc496789e5dbe2aacc26d18136d197
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Fri Apr 1 18:56:46 2016 +0000

  return statements now functional

* commit b8aff0615097ca9eb3e6f9a5cabc7dadbb8a30
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Fri Apr 1 17:27:17 2016 +0000

  small additions from last night
* commit 793b329a6a0e2fb839a8204f77287cf4938263d1
  | Author: James Stenger <jms2431@columbia.edu>
  | Date:   Fri Apr 1 04:35:23 2016 +0000
  |
  | changing testall to not use Run() resolved lli issue; needs closer inspection

* commit ed3de8e345846833cc6e085dc764e00c841f6a4cd
  | Author: James Stenger <jms2431@columbia.edu>
  | Date:   Fri Apr 1 02:35:35 2016 +0000
  |
  | printf(5) now functional; need to fix issue with testall

* commit 1e53ae909a0760b23ce6b9efb17465a75654a922
  | Author: James Stenger <jms2431@columbia.edu>
  | Date:   Thu Mar 31 19:44:09 2016 +0000
  |
  | Roadmark: starting codegen; parsing of function1 and helloworld working correctly

* commit a438b998330cc8054d127ab57bc44ad98823a358
  | Author: James Stenger <jms2431@columbia.edu>
  | Date:   Thu Mar 31 19:39:35 2016 +0000
  |
  | added semant and codegen; not yet actually producing LLVM IR

* commit dd39f8025e838e0d06cd62023ae8875f331883bd
  | Author: James Stenger <jms2431@columbia.edu>
  | Date:   Thu Mar 31 17:34:07 2016 +0000
  |
  | parsing helloworld correctly; moving to codegen

* commit e89cb6303e55ae88618e66fa803040407b9d95c3
  | Author: James Stenger <jms2431@columbia.edu>
  | Date:   Thu Mar 31 17:03:26 2016 +0000
  |
  | working on parsing functions and arrays

* commit 76f643c9af7d519e70dc0e41eeb59f96777a7
  | Author: James Stenger <jms2431@columbia.edu>
  | Date:   Wed Mar 30 23:19:18 2016 +0000
  |
  | testing infrastructure done

* commit 2ec4385464bd4fe74ae54bae6d09ebb18be2d
  | Author: James Stenger <jms2431@columbia.edu>
finishing up testing infrastructure

* commit c01fdfac548f9fd5e89873470c63222b400f6210
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Sat Mar 26 19:51:19 2016 +0000

  added .out files for test-* in tests/ folder

* commit 3c2dba383a1c369227a1fa711b2933ded25222ba
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Sat Mar 26 19:41:28 2016 +0000

  fixed issue with test script

* commit 4267b49216b6365ad41a5acb6dd29fbcbe71c51f
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Sat Mar 26 19:34:45 2016 +0000

  test script complete; working on testall

* commit 12865f87e6489148609516c5f0fcc238fddb7d6e
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Sat Mar 26 19:29:27 2016 +0000

  revised testing setup

* commit 58a346999dfbfeee26482b03a9251b495aec3e03b
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Sat Mar 26 17:49:32 2016 +0000

  added valid programs from barrios to tests/; add vimrc instructions for c-style .stp file extension instructions to t_misc/; updated TODO.txt

* commit 369ecf6b5de0ca33b3a1c18966966bd4ec9d5b7f
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Sat Mar 26 17:15:35 2016 +0000

  updates since break; may or may not be unstable: committing for test suite updates

* commit bd20fb0c6aa3a5de59f59d0d1e897ca885f3635c
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Wed Mar 16 22:10:07 2016 +0000
stable commit. Integrating classes from parser

* commit 241ffae158c7e8e3089ab410110f3f890d7f21e5
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Wed Mar 16 18:59:52 2016 +0000

  stmts now allow variable assignment and declarations; pretty-printing included

* commit 98399a28f425f29a4ed243b48408ef645a09b4f3
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Wed Mar 16 18:34:58 2016 +0000

  fixing up old issues: stmts, scanner. Working towards merge with new features in parser/barrios

* commit 956038eeaecedeb904d23e25c68f825de9d5d8c6
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Wed Mar 16 18:31:38 2016 +0000

  adding necessary features to stmts

* commit 4237f6cd7fde030bd7f1469f0deb4de737a0ae75
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Wed Mar 16 06:18:11 2016 +0000

  added chars, strings, properly scanned to include correct possible characters

* commit b1a982e9ee1a030330b4eb68a14e2e158cde6322
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Wed Mar 16 05:25:55 2016 +0000

  fixed parsing issue

* commit 38c959473cd7a7ffde9c820414e815a20c5a26f0
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Wed Mar 16 04:54:55 2016 +0000

  added test

* commit 76ae9dd1148bf77d2f243f8b69e25c4393d259c9
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Wed Mar 16 04:44:01 2016 +0000
minor scanner adjustments

* commit 22aaf77ae7b03f3250b6e31ed91c084714ad0aa7
* Author: Jonathan Barrios <jeb2239@columbia.edu>
* Date:   Sun Apr 17 13:01:50 2016 -0400

  top level directory is most up to date

* commit 72df8902eb0820561765c2663b2f8792b3a9eb2b
* Author: Jonathan Barrios <jeb2239@columbia.edu>
* Date:   Sun Apr 17 12:56:33 2016 -0400

  my current branch

* commit 8300dae45f3cedce550b715d83823957cc2829ed7
* Author: Jonathan Barrios <jeb2239@columbia.edu>
* Date:   Sun Apr 17 12:27:16 2016 -0400

  array access, array create, object create parsing these not generating code yet

* commit 65b2aca5ea07c61d73028a82616848f08f94fd0e
* Author: Jonathan Barrios <jeb2239@columbia.edu>
* Date:   Sun Apr 17 11:36:07 2016 -0400

  parsing classes is good enough

* commit e4b6e3f88ab4f9a782be1398798f2295a9c32b2
* Author: Jonathan Barrios <jeb2239@columbia.edu>
* Date:   Sun Apr 17 03:34:13 2016 -0400

  parsing classes, but not like pretty printing etc...lot of warning need more work

* commit d5a06231be8ef494836ab3f88da54166d822c4f0
* Author: Jonathan Barrios <jeb2239@columbia.edu>
* Date:   Sun Apr 17 02:25:56 2016 -0400

  adding classes to parser

* commit 33d8367fc83873d910ca554a56e28aee36ba4328
* Author: Jonathan Barrios <jeb2239@columbia.edu>
* Date:   Sat Apr 16 21:12:17 2016 -0400

  this should compile
Date: Sat Apr 2 19:47:22 2016 -0400

my changes

* commit 2d2b109f907b25d127f7af9e07e89b7cea2fa667
  Author: Jonathan Barrios <jeb2239@columbia.edu>
  Date: Thu Mar 31 23:52:51 2016 -0400

will not compile, adding function literals

* commit d7b5330cb0753669fb99110bd5ecd5e5b6ddd905
  Author: Jonathan Barrios <jeb2239@columbia.edu>
  Date: Thu Mar 31 21:27:32 2016 -0400

h

* commit 77a2f997d1b670a1a7d7f1418bd46e7047381eb0
  Author: Jonathan Barrios <jeb2239@columbia.edu>
  Date: Thu Mar 31 21:16:37 2016 -0400

folder for hello world

* commit 60798cdf9bcb073af48617f4aed063944ec6886f
  Author: Jonathan Barrios <jeb2239@columbia.edu>
  Date: Thu Mar 31 20:57:14 2016 -0400

stuff

* commit 7ae6a9a64070de9797c622f7cfc4b01b5a1e3b6a
  Author: Jonathan Barrios <jeb2239@columbia.edu>
  Date: Tue Mar 29 05:07:35 2016 -0400

parsing declarations

* commit a763409976327c6015adc0f8fcec0c8604f39d4
  Author: Jonathan Barrios <jeb2239@columbia.edu>
  Date: Tue Mar 29 03:54:08 2016 -0400

this now compiles and parses minimal expressions correctly, will uncomment code and iteratively test to add features, every new feature must have a pretty printer

* commit f5d9231f053bbf4662676d3a3b195905a0e6a105
  Author: Jonathan Barrios <jeb2239@columbia.edu>
  Date: Sun Mar 27 15:52:14 2016 -0400
started pretty printer

Actually started parser for hello world program very stripped down version

compiles but doesn't work need to add pretty printing to debug

working on making the parser work for at least simple functions

added newStop.stp which is the same example I made a snippit of

more stuff that still will not compile

also ast

more changes, working on establishing function type signatures, still won't compile
* commit 024bd40f2c80f5aaadbef23b6272607402442ae7
  
  Author: Jonathan Barrios <jeb2239@columbia.edu>
  
  Date: Wed Mar 16 15:47:10 2016 -0400
  
  added more stuff, still won't compile, but starting to look like something

* commit cae8af8b270ebf6fa6454596aa7f8a8e18a130b1
  
  Author: Jonathan Barrios <jeb2239@columbia.edu>
  
  Date: Wed Mar 16 14:09:50 2016 -0400
  
  will not compile, working on parser

* commit 2ab7aa7e69166ec2a0c68263997935106c65e88
  
  Author: Jonathan Barrios <jeb2239@columbia.edu>
  
  Date: Sat Mar 5 21:16:23 2016 -0500
  
  integrating classes into grammer, some changes to ast

* commit ae5c34d3881a23d68e8f2f970aea359881a185f6
  
  Author: Jonathan Barrios <jeb2239@columbia.edu>
  
  Date: Sat Mar 5 18:54:20 2016 -0500
  
  much better makefile

* commit 225045da7e4445ae04e3bf1d15ac7f7679aba2c9
  
  Author: Jonathan Barrios <jeb2239@columbia.edu>
  
  Date: Sat Mar 5 18:04:43 2016 -0500
  
  started working on classes and methods

* commit 02563c28850e67c531f94792fc0a20f457cc8847
  
  Author: Jonathan Barrios <jeb2239@columbia.edu>
  
  Date: Sat Mar 5 13:44:06 2016 -0500
  
  added better git ignore

* commit e1faba9eb95a600f6f67a58f6142e1173cb67da4
  
  Merge: 8e71de8 b3bd21c
  
  Author: Jonathan Barrios <jeb2239@columbia.edu>
  
  Date: Sat Mar 5 13:12:21 2016 -0500
  
  Merge branch 'parser' into parser_james
commit b86b271185a92c38b98b1e4f5f26815cc0cd0c2
Author: James Stenger <jms2431@columbia.edu>
Date:   Sun Feb 28 20:55:36 2016 +0000

updated exprs in parser, ast

commit 2c7c340cc12cfef1a711de6598da52221ca53
Author: James Stenger <jms2431@columbia.edu>
Date:   Sun Feb 28 20:40:09 2016 +0000

continued adding to expression grammar, scanner, parser

commit a76f8c388c0f6d8028da54865391604945
Author: James Stenger <jms2431@columbia.edu>
Date:   Sun Feb 28 19:54:42 2016 +0000

added mutli-line, singleline comments, tests

commit 2be67c6e250d984e8ebd69a3b54865391604945
Author: James Stenger <jms2431@columbia.edu>
Date:   Sun Feb 28 18:27:39 2016 +0000

printing now functional; TODO: finish ast, parser

commit 83fd603b9f3afed59a947afcdf36521a1b1340e
Author: James Stenger <jms2431@columbia.edu>
Date:   Sun Feb 28 17:43:47 2016 +0000

compiling; moving towards printable ast

commit 148f09af9e0b9e364eba811e7d2a7581e9e308e3
Author: James Stenger <jms2431@columbia.edu>
Date:   Sun Feb 28 17:00:14 2016 +0000

updated ast; makes and printing now possible

commit feecc497ad65512780e105de7bfb379e43f3e510
Author: Jonathan Barrios <jeb2239@columbia.edu>
Date:   Fri Feb 26 23:23:44 2016 -0500
added dice parser as example

* commit ad9d3200dec9d8c050f295b0f4913a14c68f49d
  Author: James Stenger <jms2431@columbia.edu>
  Date: Sat Feb 27 04:20:35 2016 +0000

  added necessary tokens to parser for Makefile

* commit 44e0e699ed687e7d9027ed5f99df5e2033a8e
  \ Merge: fa59514 2bc387c
  Author: James Stenger <jms2431@columbia.edu>
  Date: Sat Feb 27 04:16:07 2016 +0000

  Merged ast.ml with rest of files
  Merge branch 'parser_james' of https://bitbucket.org/nottrainwreck/stop into parser_james

* commit 2bc387c2c8ba33697522341d80d55e703dd8a58a
  Author: Jonathan Barrios <jeb2239@columbia.edu>
  Date: Fri Feb 26 23:15:14 2016 -0500

  Dice ast added

* commit fa5951446d1ec1224f0ae43b97d457b05c4f802a
  \ Author: James Stenger <jms2431@columbia.edu>
  Date: Sat Feb 27 04:14:56 2016 +0000

  compiler fixed

* commit bb9bcc85211dce9bcde1456bd032803def4be058
  Author: James Stenger <jms2431@columbia.edu>
  Date: Sat Feb 27 03:14:43 2016 +0000

  'else if' token changed to 'elseif'

* commit 512400fc46386ad85cb85912fc65c81e87c809e6
  Author: James Stenger <jms2431@columbia.edu>
  Date: Sat Feb 27 03:13:26 2016 +0000

  fix for error-free make

* commit 7ba11d436fc547faab3ebe736a95a26dc206cffe0
  Author: James Stenger <jms2431@columbia.edu>
  Date: Sat Feb 27 03:01:55 2016 +0000
added tokens (literals, primitives)

commit ac577a8586079723574ff4bd95d9bbeab5dfae78
Author: James Stenger <jms2431@columbia.edu>
Date: Sat Feb 27 02:46:47 2016 +0000
updated scanner, parser w/ more tokens

commit 752161335b7d9d5e7eb54317037dcbaebc64dfab73
Author: James Stenger <jms2431@columbia.edu>
Date: Sat Feb 27 02:29:42 2016 +0000
added tokens to scanner & parser; updated calc

commit 6db04a67e1e40e8e0fcd63bdcaba6c26c424c773
Author: James Stenger <jms2431@columbia.edu>
Date: Sat Feb 27 02:20:17 2016 +0000
brought in Jonathan's updates

commit fd20967c18d05024e91bface5c25c75a3a28db1d
Author: James Stenger <jms2431@columbia.edu>
Date: Sat Feb 27 01:57:48 2016 +0000
updated calc

commit 9b6038832915362017eb70bca1411fe54d5be6b
Author: James Stenger <jms2431@columbia.edu>
Date: Sat Feb 27 01:49:47 2016 +0000
initial commit

commit ba77d07085756513b846798b9dfe4734be6d327c
Author: James Stenger <jms2431@columbia.edu>
Date: Sat Feb 27 01:09:09 2016 +0000
update for friday

commit 6483366f1d664600b7229ee54e95385078067d8
Author: Jonathan Barrios <jeb2239@columbia.edu>
Date: Fri Feb 26 20:45:26 2016 -0500
scanner with more tokens
* commit d4a42dc7964bbd26a8581a2646f9f436b0d5067d
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Wed Feb 24 23:26:54 2016 +0000
  
  fixed Makefile, added tokens, fixed scanner/parser EOF issue

* commit cb00884ce0a33c1514168ce69c066e230dbad37a
  Author: James Stenger <jms2431@columbia.edu>
  Date:   Wed Feb 24 22:57:56 2016 +0000
  
  updated scanner with microc idioms, added Makefile

* commit f474884713958a0f92555ac30b33ec82443c859f7
  Author: Jonathan Barrios <jeb2239@columbia.edu>
  Date:   Sun Feb 21 14:34:17 2016 -0500
  
  starting parser from ocamlyacc tutorial

* commit e19512c15e508c411d54b67eeb32422d1a5c7066
  Author: Jonathan Barrios <jeb2239@columbia.edu>
  Date:   Fri Feb 19 20:46:00 2016 -0500
  
  start of scanner

* commit 7d90cec005e34093c3873aa68c3992335b444d7b
  Author: Jonathan Barrios <jeb2239@columbia.edu>
  Date:   Fri Feb 19 20:43:53 2016 -0500
  
  start of parser

* commit 1a308301f9559b92b9505540621f239566d11653
  Author: Jonathan Barrios <jeb2239@columbia.edu>
  Date:   Fri Feb 12 22:27:08 2016 +0000
  
  README.md edited online with Bitbucket

* commit 39513c83945cd45cd66c59b9a723345a126d68df
  Author: Jonathan Barrios <jeb2239@columbia.edu>
  Date:   Fri Feb 12 22:26:53 2016 +0000
  
  README.md edited online with Bitbucket

* commit 867b065ed221d47df46cd1bb915fb724ea74800
removed that thing

* commit c4df4df6dee038ef3c6dd6a5a1226218e05684c1
  Author: jmstenger <james.m.stenger@columbia.edu>
  Date:   Fri Jan 29 21:15:09 2016 -0500
  added bla

* commit 6e2f3bc683ed5ac52bc1c70b04eфаd3805d7f85
  Author: Jonathan Barrios <jeb2239@columbia.edu>
  Date:   Fri Jan 29 21:02:54 2016 -0500
  hello
10. Full Code Listing
open Core.Std

module A = Analysis
module C = Codegen
module E = Exceptions
module G = Generator
module L = Llvm
module P = Parser
module S = Scanner
module U = Util

(* Compile <src> <destination> *)

let get action = function
  | "-t" -> Tokens
  | "-a" -> Ast
  | "-s" -> Sast
  | "-css" -> CompileStdinStdout
  | "-csf" -> CompileStdinFile
  | "-cfs" -> CompileFileStdout
  | "-cf" -> CompileFile
  | "-h" -> Help
  | as s -> raise (E.InvalidOption s)

let check single argument = function
  | "-n" -> (Help, "")
  | "-tndl" | "-a" | "-s" | "-css" | "-csf" | "-cfs" | "-cf" | "-h" | as s -> raise (E.NoFileArgument) | "-n"

let help_string = {
  "Usage: stop [-option] <source file>
  | [-option] (default): to "-css"
  | -t: Print tokens
  | -a: Print AST
  | -s: Print SAST
  | -css: Compiles stdin to stdout
  | -csf: Compiles stdin to file
  | -cfs: Compiles file to stdout <filename>.<ext> or <filename>.<ext> -> <filename>.ll
  | -cf: Compiles file to <filename>.<ext> -> <filename>.ll
  | -h: Print help
  }

let stop name filename =
  let basename = Filename.basename filename in
  let filesize = Filename.chop_extension basename in
  if filesize = "" then raise (E.InvalidOption s)

try
  (* Get the Appropriate Action *)

  let (action, filename) =
    if Array.length Sys.argv = 1 then
      (Tokens, filename)
    else if Array.length Sys.argv = 2 then
      (CompileStdinStdout, Sys.argv.1)
    else if Array.length Sys.argv = 3 then
      check single argument (Sys.argv.1)
    else
      raise E.InvalidArgc
in

  (* Iterative Application of each Compilation Phase *)

  let file in () = if filename = "" then stdin else open in filename in

  let lexbuf () = Lexing.from channel (file in ()) in
  let token list () = G.build token list filename (lexbuf ()) in
  let ast () = G.build ast filename (token list ()) in
  let sast () = A.analyze filename (ast ()) in
  let llm () = C.codegen sast (sast ()) in

  (* Respond Appropriately to Action *)

match action with
  Tokens -> print_string (U.token list to string (token list ()))
(\* Deal with Exceptions *\)
E.illegalCharacter(file, c, ln) ->
  print_string
  ("illegal character '"^c"' in line"
   string_of_int ln ^ " of " ^ file ^ "\n")
| Parsing.ParseError ->
  print_string
  ("Syntax Error:"
   ^ U.error_string_of_file \G.filename_ref
   ^ c, line " ^ string_of_int \G.lineno_ref
   ^ c, characters " ^ U.error_string_of_cnum \G.cnum_ref \G.last_token_ref
   ^ "Token " ^ U.string_of_token \G.last_token_ref ^ "\n")
| as e -> raise e

/*
Compile in
let lexbuf = Lexing.from_channel stdin in
let ast = Parser.program Scanner.token lexbuf in
Semant.check ast;
match action with
| LLVM IR -> print_string (Llvm.string_of_llmodule (Codegen.translate ast))
| Compile -> let m = Codegen.translate ast in
Llvm analysis.assert valid module m;
print_string (Llvm.string_of_llmodule m)
*/

codegen.ml

(* Code Generation Phase *)

/*
Input: Semantically Checked AST (type sprogram)
Output: LLVM Module

Produces an LLVM IR translation of the source program
LLVM Tutorial:
http://llvm.org/docs/tutorial/index.html
LLVM Documentation:
http://llvm.moe/
http://llvm.moe/ocaml/
*/
open Core.Std
open Sast
open Ast

module A = Analysis
module E = Exceptions
module L = Llvm
module U = Utils

let context = L.global_context ()
let the module = L.create_module context "Stop"
let builder = L.builder context
let i32_t = L.i32_type context
let i64_t = L.i64_type context
let il_t = L.il_type context
let float_t = L.float_type context
(*let double_t = L.double_type_context*)
let void_t = L.void_type context
let str_t = L.pointer_type (L.i8_type context)

(* Control Flow References *)
let br block = ref (L.block_of_value (L.const_int i32_t 0))
let cont block = ref (L.block_of_value (L.const_int i32_t 0))
let is_loop = ref false

let struct types:(string, L.lltype) Hashtbl.t = Hash.tbl.create ()
  ~hashable:String.hashable
  ~size:10

let struct field indexes:(string, int) Hashtbl.t = Hash.tbl.create ()
  ~hashable:String.hashable
  ~size:50

(* Named Values inside current scope *)
let named values:(string, L.llvalue) Hashtbl.t = Hash.tbl.create ()
  ~hashable:String.hashable
  ~size:50

(* Named parameters inside current scope *)
let named parameters:(string, L.llvalue) Hashtbl.t = Hash.tbl.create ()
  ~hashable:String.hashable
  ~size:50

let str_type = Arraytype(Char_t, 1)
let rec get array type array t = match array t with
  | Arraytype(array, i) -> L.pointer_type(get lltype exn (Datatype(array)))
  | Arraytype(array, i, t) -> L.pointer_type(get array type (Arraytype(array, i, t)))
  | _ -> raise(E.InvalidDatatype "Array Type")

and find struct exn name =
  if name = "String" then (L.i8 type context) else
    try
      Hashbl.find exn struct types name
    with
      Not_found -> raise (E.InvalidStructType(name))
    in
      L.pointer_type(L.function_type(find struct exn name)) (Array.of_list lllargs)

and get lltype exn (data t:datatype) = match data t with
  | Datatype(Int t) -> i32_t
  | Datatype(Float t) -> float t /* TODO: Decide what to do a/b doubles & floats */
  | Datatype(Bool t) -> i1_t
  | Datatype(Char t) -> i8_t
  | Datatype(Unit t) -> void_t
  | Datatype(Object t(name)) -> L.pointer_type(find struct exn name)
  | Arraytype(t, i) -> get array type (Arraytype(t, i))
  | Functiontype(dt1, dt2) -> get function type dt1 dt2
  | _ -> raise (E.InvalidDatatype(U.string_of_datatype data t))

let lookup llfunction exn name = match (L.lookup function exn name the module) with
  | None -> raise (E.LLVMFunctionNotFound(name))
  | Some f -> f

let rec codegen sexpr sexpr ~builder:llbuilder = match sexpr with
  | SLit(i) -> L.const int i32 t i
  | SFloatLit(f) -> L.const float float t f
  | SBoolLit(b) -> if b then L.const int i1 t 1 else L.const int i1 t 0
  | SCharLit(c) -> L.const int i8 t (Char.to_int c)
  | SStringLit(s) -> L.build_global_stringptr s "tmp" llbuilder
  | SFunctionLit(s, ) -> codegen function lit s llbuilder
  | SAssign(e, e2) -> codegen assign e1 e2 llbuilder
  | SArrayAccess(se, se1, _ ) -> codegen array access false se se1 llbuilder
  | SObjectAccess(se, se2, d) -> codegen obj access true se se2 d llbuilder
  | SHeapRef(e) -> L.build add (L.let const int i32 t 0) (L.let const int i32 t 0) "nop" llbuilder
  | SId(id, ) -> codegen id false id llbuilder
  | SBinop(e1, op, e2, data t) -> handle binop e1 op e2 data t llbuilder
  | SUnop(op, e) -> handle unop op e llbuilder
  | SCall(action, name, se, se2, data t, ) -> codegen call name se se2 data t llbuilder
  | SArrayCreate(t, e, dl) -> codegen array_create t e dl
  | _ -> raise E.NotImplemented

(* Generate Code for Binop *)

and handle binop el op e2 data t llbuilder =
  (* Get the type of el and e2 *)
  let type1 = A.sexpr_to_type el in
  let type2 = A.sexpr_to_type e2 in
  (* Generate LLVM from el and e2 *)
  let el = codegen sexpr el ~builder:llbuilder in
  let e2 = codegen sexpr e2 ~builder:llbuilder in
  (* Integer LLVM functions *)
  let int ops el op e2 =
    match op with
    | Add -> L.build add el e2 "addtmp" llbuilder
    | Sub -> L.build sub el e2 "subtmp" llbuilder
    | Mul -> L.build mul el e2 "multmp" llbuilder
    | Div -> L.build div el e2 "divtmp" llbuilder
    | Modulo -> L.build srem el e2 "sremtmp" llbuilder
    | Equal -> L.build icmp L.icmp.Eq el e2 "eqtmp" llbuilder
    | Neq -> L.build icmp L.icmp.Neq el e2 "neqtmp" llbuilder
    | Less -> L.build icmp L.icmp.Slt el e2 "lesstmp" llbuilder
    | Leq -> L.build icmp L.icmp.Sle el e2 "leqtmp" llbuilder
    | Greater -> L.build icmp L.icmp.Sgt el e2 "sgtmp" llbuilder
    | Gt -> L.build icmp L.icmp.Sge el e2 "sgtmp" llbuilder
    | And -> L.build and el el e2 "andtmp" llbuilder
    | Or -> L.build or el e2 "ortmp" llbuilder
    | _ -> raise Exceptions.IntOpNotSupported

(* Floating Point LLVM functions *)

let float ops el op e2 =
  match op with
  | Add -> L.build fadd el e2 "fltmp" llbuilder
  | Sub -> L.build fsub el e2 "flsubtmp" llbuilder
  | Mul -> L.build fmul el e2 "flmulttmp" llbuilder
  | Div -> L.build fdiv el e2 "fldivtmp" llbuilder
  | Modulo -> L.build frem el e2 "frsremtmp" llbuilder
  | Equal -> L.build fcmp L.fcmp.Eq el e2 "flcmpeq" llbuilder
  | Neq -> L.build fcmp L.fcmp.Neq el e2 "flrneqtmp" llbuilder
  | Less -> L.build fcmp L.fcmp.Slt el e2 "fllesstmp" llbuilder
  | Leq -> L.build fcmp L.fcmp.Sle el e2 "flesqtmp" llbuilder
  | Greater -> L.build fcmp L.fcmp.Sgt el e2 "flsgtmp" llbuilder
  | Gt -> L.build fcmp L.fcmp.Sge el e2 "flsgtmp" llbuilder
  | And -> L.build and el el e2 "flandtmp" llbuilder
  | Or -> L.build or el e2 "flortmp" llbuilder
  | _ -> raise Exceptions.FloatOpNotSupported
(* Use Integer Arithmetic for Ints, Chars, and Bools *)
(* Use Floating-Point Arithmetic for Floats *)

let type handler data t = match data t with
| Datatype(Int t) -> match data t with
| Datatype(Char t) -> match data t with
| Datatype(Bool t) -> match data t with
| Datatype(Float t) -> match data t with
| _ -> raise E.UnexpectedBinopEvaluationType

in

and handle unop op se data t llbuilder =

let se type = A.sexpr_or_type exn se in
let llvalue = codegen SEXP se llbuilder in

let unops op se type llval = match (op, se type) with
| (Neg, Datatype(Int t)) -> L.build_neg llvalue "int unoptmp" llbuilder
| (Not, Datatype(Bool t)) -> L.build_not llvalue "bool unoptmp" llbuilder
| _ -> raise E.UnexpectedUnopEvaluationType

in

let type handler data t = match data t with
| Datatype(Float t) -> match data t with
| Datatype(Int t) -> unops op se type llvalue
| _ -> raise E.UnexpectedUnopEvaluationType

in

and codegen_call sexpr sexpr l data t llbuilder = match sexpr with

SId(finame, _) ->
  match fname with
  | "printf" -> codegen printf sexpr l llbuilder
  | _ -> codegen_function_call sexpr sexpr l data t llbuilder

and codegen function_call sexpr sexpr l data t llbuilder =

let call function flval =

let fun params = List.map (fun sexpr -> codegen SEXP sexpr l llbuilder) sexpr l in
match data t with
| Datatype(Unit t) -> L.build_call flval (Array.of list params) "temp" llbuilder
| _ -> L.build_call flval (Array.of list params) "temp" llbuilder

in

match sexpr with
| SId(finame, _) ->
  let f = lookup llfunction_exn fname in
  call function f
| S0bfAccess(se1, se2, data t) ->
  let f = codegen_obj_access true se1 se2 data t llbuilder in
  call function f

and codegen printf sexpr l llbuilder =

(* Convert printf format string to llvalue *)

let format str = List.hd exn sexpr l in
let format llstr = match format str with
| SStringLit(s) -> L.build_global_stringptr s "fmt" llbuilder
| _ -> raise E.PrintfFirstArgNotString

in

(* Convert printf args to llvalue *)

let args = List.hd exn sexpr l in
let format llargs = List.map args (fun sexpr -> codegen SEXP sexpr l llbuilder) in
let fun llvalue = lookup llfunction_exn "printf" in
let llargs = Array.of list (format llstr :: format llargs) in
L.build call fun llvalue llargs "printf" llbuilder

and codegen id isDeref id llbuilder =

if isDeref then
try Hashtbl.find_exn named_parameters id
  with | Not_found ->
  try let var = Hashtbl.find_exn named_values id in
    L.build load var id llbuilder
  with | Not_found -> raise E.UndefineId id
else
try Hashtbl.find_exn named_parameters id
  with | Not_found ->
  try Hashtbl.find_exn named_values id
  with | Not_found -> raise E.UndefineId id

and codegen assign se1 se2 llbuilder =

(* Get lhs llvalue; don't emit as expression *)

let lhs = match se1 with
| SId(id, _) ->
  (try Hashtbl.find_exn named_parameters id
    with | Not_found ->
    try Hashtbl.find_exn named_values id
    with | Not_found -> raise E.UndefineId id)
| S0bfAccess(se1, se2, data t) -> codegen_obj_access false se1 se2 data t llbuilder
| SArrayAccess(se, se1, _) ->
  codegen array access true se se1 llbuilder
| _ -> raise E.AssignmentLhsMustBeAssignable
let rhs = match sel2 with
  SObject(se1, se2, data_t) -> codegen_obj_access true sel1 se2 data_t llbuilder
| _ -> codegen_sexpr se2 ~builder:llbuilder
in
let field_name = match rhs with
  SId(_, data_t) -> field
| _ -> codegen_obj_access true le data_t llbuilder
in
let field_type = match rhs with
  SId(_, data_t) -> data_t
| _ -> codegen_obj_access true le data_t llbuilder
in
let search_term = obj_type_name ^ *. ^ field_name in
let field_index = HashTbl.find exn struct field indexes search_term in
let llvalue = L.build_struct gep struct llval field_index field_name llbuilder in
let llvalue = if isAssign
  then L.build load llvalue field_name llbuilder
else llvalue
in
llvalue

and codegen_array_access isAssign e e1 llbuilder =
  let indices = List.map e1 ~f:(codegen_sexpr ~builder:llbuilder) in
  let indices = Array.of_list indices in
  let arr = codegen_sexpr e ~builder:llbuilder in
  let llvalue = L.build_gep arr indices ~typ:tmp llbuilder in
  if isAssign
    then llvalue
  else L.build_load llvalue "tmp" llbuilder

and codegen_function_lit fname llbuilder =
  let f llval = lookup_llvm_function_exn fname in
  f llval

and codegen_return sexpr llbuilder = match sexpr with
  SNoexpr -> L.build_ret void llbuilder
| _ -> L.build_ret (codegen_sexpr sexpr ~builder:llbuilder) llbuilder

and codegen_break llbuilder =
  let b = fun () -> !br block in
  L.build_br (b ()) llbuilder

and codegen_cont llbuilder =
  let b = fun () -> !cont block in
  L.build_br (b ()) llbuilder

(* TODO: Alloca vs. Malloc *)

and codegen_local var name data t sexpr llbuilder =
  let lltype = match data t with
    Datatype(Object t(name)) -> find_struct_exn name
  | _ -> get_lltype_exn data_t
  in
  let alloca = L.build_alloca lltype var name llbuilder in
  (* let alloca = L.build_malloc lltype Var_name llbuilder in *)
  HashTbl.add exn named_values ~key:var_name ~data:alloca;
  let lhs = add(tl(name), data_t) in
  match sexpr with
  | _ -> codegen_assign lhs sexpr llbuilder

and codegen_stmt stmt ~builder:llbuilder = match stmt with
  SBlock(s1) -> List.hd exn (List.map ~f:(codegen_stmt ~builder:llbuilder) s1)
| SExpr(se, _) -> codegen_sexpr se llbuilder
| SReturn(se, _) -> codegen_return se llbuilder
| SLocal(s, data_t, se) -> codegen_local s data_t se llbuilder
| SIf(se, sl, s2) -> codegen_if stmt se sl s2 llbuilder
| SFor(se1, se2, se3, ss) -> codegen_for stmt se1 se2 se3 ss llbuilder
| SWhile(se, ss) -> codegen_while stmt se ss llbuilder
| SBreak -> codegen_break llbuilder
| SContinue -> codegen_continue llbuilder

and codegen_if stmt predicate then stmt else stmt llbuilder =
  let cond val = codegen_sexpr predicate llbuilder in
  let start_bb = L.insertion_block llbuilder in
  let the_function = L.block_parent start_bb in
  let then_bb = L.append_block context "then" the_function in
  L.position at end then_bb lbuilder;
  let _ = codegen_stmt then_stmt llbuilder in
  let new_then_bb = L.insertion_block llbuilder in
  let else_bb = L.append_block context "else" the_function in
  L.position at end else_bb lbuilder;
let new else bb = L.insertion block llbuilder in
let merge bb = L.append block context "ifcont" the function in
L.position at end merge bb llbuilder;

let else bb val = L.value of block new else_bb in
L.position at end start bb llbuilder;

ignore (L.build cond br cond val then bb else_bb llbuilder);
L.position at end new then_bb llbuilder; ignore (L.build br merge_bb llbuilder);
L.position at end new else_bb llbuilder; ignore (L.build br merge_bb llbuilder);
L.position at end merge_bb llbuilder;
else_bb val

and codegen for stmt init se cond se inc_se body stmt llbuilder =
let old_val = is_loop in
is_loop := true;

let the_function = L.block_parent (L.insertion block llbuilder) in
let _ = codegen sexpr init_se llbuilder in

let loop bb = L.append block context "loop" the function in
let inc_bb = L.append block context "inc" the function in
let cond_bb = L.append block context "cond" the function in
let after_bb = L.append block context "afterloop" the_function in

let _ = if not old_val then
  cont block := inc_bb;
  br_block := after_bb;
  in
  ignore (L.build br cond bb llbuilder);

(* Start insertion in loop bb. *)
L.position at end loop_bb llbuilder;

(* Emit the body of the loop. This, like any other expr, can change the *
 * current BB. Note that we ignore the value computed by the body, but *
 * don't allow an error. *)
ignore (codegen stmt body stmt ~builder=llbuilder);

let bb = L.insertion block llbuilder in
L.move block after bb inc_bb;
L.move block after inc_bb cond_bb;
L.move block after cond_bb after_bb;
ignore (L.build_br inc_bb cond_bb llbuilder);

(* Start insertion in loop bb. *)
L.position at end inc_bb llbuilder;

(* Emit the step value. *)
let _ = codegen sexpr inc_se llbuilder in
ignore (L.build_br cond_bb llbuilder);
L.position at end cond_bb llbuilder;

let cond_val = codegen sexpr cond_se llbuilder in
ignore (L.build cond_br cond_val loop_bb after_bb llbuilder);
L.position at end after_bb llbuilder;
is_loop := old_val;
L.const null float t

and codegen while stmt cond_se body stmt llbuilder =
let null sexpr = SIntLit(0) in
codegen for stmt null sexpr cond_se null sexpr body stmt llbuilder

and codegen array create llbuilder t expr type el =
if(List.length el > 1) then raise(Exceptions.ArrayLargerThan1Unsupported)
else
  match expr type with
  Arraytype(Char t, 1) ->
    let e = List.hd expr el in
    let size = (codegen sexpr e llbuilder) in
    let t = get lltype expr t in
    let arr = L.build array malloc t size "tmp" llbuilder in
    let arr = L.build pointercast arr (L.pointer_type t) "tmp" llbuilder in
    (* initialise array arr size (const_int 132 t 0) @ llbuilder; *)
    arr
| ->
    let e = List.hd expr el in
    let t = get lltype expr t in

    (* This will not work for arrays of objects *)
    let size = (codegen sexpr e llbuilder) in
    let size t = L.build intcast (L.size of_t) t "tmp" llbuilder in
    let size t = L.build intcast (L.size of_t) t "tmp" llbuilder in
    let size real = L.build add size (L.const_int 132 t 1) "arr size" llbuilder in
    let arr len = L.build array malloc t size real "tmp" llbuilder in
    let arr len ptr = L.build pointercast arr (L.pointer_type t) "tmp" llbuilder in
    (* Store length at this position *)
    ignore (L.build store size real arr len ptr llbuilder);
    (* initialise array arr len ptr size real (const_int 132 t 0) 0 llbuilder; *)
    arr

(* Codegen Library Functions *)
let codegen library_functions () =
  (* C Std Lib functions (Free with LLVM) *)
  let printf t = L.var_arg function_type i32 t [ L.pointer_type i8 t ] in
  let L.declare_function "printf" printf t the module in
  let malloc t = L.function_type (str t) [ i32 t ] in
  let L.declare_function "malloc" malloc t the module in
  let open t = L.function_type i32 t [ (L.pointer_type i8 t); i32 t ] in
  let L.declare_function "open" open t the module in
  let close t = L.function_type i32 t [ i32 t ] in
  let L.declare_function "close" close t the module in
  let read t = L.function_type i32 t [ i32 t; L.pointer_type i8 t; i32 t ] in
  let L.declare_function "read" read t the module in
  let write t = L.function_type i32 t [ i32 t; L.pointer_type i8 t; i32 t ] in
  let L.declare_function "write" write t the module in
  let lseek t = L.function_type i32 t [ i32 t; i32 t; i32 t ] in
  let L.declare_function "lseek" lseek t the module in
  let exit t = L.function_type void t [ i32 t ] in
  let L.declare_function "exit" exit t the module in
  let realloc t = L.function_type str t [ str t; i32 t ] in
  let L.declare_function "realloc" realloc t the module in
  let getchar t = L.function_type i32 t [ i32 t ] in
  let L.declare_function "getchar" getchar t the module in
  let sizeof t = L.function_type (i32 t) [ i32 t ] in
  let L.declare_function "sizeof" sizeof t the module in

let codegen struct stub s =
  let struct t = L.named_struct_type context s.scname
  in
  Hashtbl.add struct_types
    -key:s.scname
    -data:struct t

let codegen struct s =
  let struct t = Hashtbl.find_exn struct_types s.scname in
  let type list = List.map s.sfields
    -f:(function Field(_, data t) -> get_llifetime data t)
    in
  let name list = List.map s.sfields
    -f:(function Field(_, s, _) -> s)
    in

  (* Add key field to all structs *)
  let type list = i32 t :: type list in
  let name list = "key" :: name list in
  let type array = Array.of_list type list
  List.iteri name_list
    -f:(fun i f ->
      let n = s.scname ^ "\n" ^ f
          in
          Hashtbl.add_exn struct_field_indexes
            -key:n
            -data:i);

  (* Add the struct to the module *)
  L.struct_set_body struct t type array true

let codegen function stub sfdecl =
  let fname = sfdecl.fname in
  let is_var_arg = ref false in
  let params = List.rev (List.fold_left sfdecl.sformals
    -f:(fun l -> (function
        Formal(_, data t) -> get_llifetime data t :: l
        | _, _, is_var_arg := true; l))
    -init:[])
    in
  let ftype =
    if !is_var_arg
      then L.var_arg function_type (get_llifetime sfdecl.sreturn t) (Array.of_list params)
    else L.function_type (get_llifetime sfdecl.sreturn t) (Array.of_list params)
    in
  L.define_function fname ftype the_module

let init params f formals =
  let formals = Array.of_list formals in
  Array.iteri (L.params f)
    -f:(fun i element ->
      let n = U.string_of formal name n in
      L.set value name n element;
      Hashtbl.add_exn named_parameters
        t
        -key:name
        -data:element;
    )

let codegen function sfdecl =
  Hashtbl.clear named values;
  Hashtbl.clear named parameters;
  let fname = sfdecl.fname in
  let f = lookup llfunction exn fname in
  let lbuilder = L.builder_at_end_context (L.entry_block f) in
  let = init params f sfdecl.sformals in
  let = codegen_struct SBlock (sfdecl.sbody) ~builder:llbuilder in

  (* Check to make sure we return; add a return statement if not *)
  let last bb = match (L.block_end (lookup llfunction_exn fname)) with
    L.After(block) -> block
    | L.At_start() -> raise (E.FunctionWithoutBasicBlock(fname))
let codegen main main =
  HashTable.clear named values;
  HashTable.clear named parameters;
let fty = L.function_type i32 t || i32 t; L.pointer_type str_t || in
let f = L.define_function "main" fty the module in
let llbuilder = L.builder at end context (L.entry block f) in

let argc = L.param f 0 in
let argv = L.param f 1 in
L.set_value_name "argc" argc;
L.set_value_name "argv" argv;
HashTable.add exn named parameters -key:"argc" -data:argc;
HashTable.add exn named parameters -key:"argv" -data:argv;

let _ = codegen_stmt (SBlock(main.sbody)) llbuilder in

let codegen sast sast =
  (* Declare the various LLVM Reserved Functions *)
  let _ = codegen_library_functions () in
  (* Generate a map of function names to their respective LLVM Struct Types *)
  let _ = List.map sast.classes ~f:(fun s -> codegen_struct_stub s) in
  (* Generate LLVM IR for classes *)
  let _ = List.map sast.classes ~f:(fun s -> codegen_struct s) in
  (* Define the program functions *)
  let _ = List.map sast.functions ~f:(fun f -> codegen_function_stub f) in
  (* Generate LLVM IR for functions *)
  let _ = List.map sast.functions ~f:(fun f -> codegen_function f) in
  (* Generate LLVM IR for main function *)
  let _ = codegen_main sast.main in
  the module

parser.mly

/* Ocamlyacc Parser for Stop */
%
open Ast
open Core.Std
module E = Exceptions
let lambda_num = ref 0
%
token DOT COMMA SEMI COLON LPAREN RPAREN LBRACE RBRACE LBRACKET RBRACKET

token PLUS MINUS TIMES DIVIDE ASSIGN NOT CARET MODULO

token INCREMENT DECREMENT

token EQ NEQ LT LEQ GT GEO TRUE FALSE AND OR

token IF ELSE FOR WHILE BREAK CONTINUE

token ARROW FATARRROW

token RETURN

token FINAL

token PUBLIC PRIVATE ANON

token SPEC CLASS METHOD

token MATCH CASE

token TYPE VAR THIS

token DEF EXTENDS

token EOF

/* Processor Directives */

.token INCLUDE

token MODULE

/* Primitive Types */
/* Literals */

%nonassoc NOELSE
%nonassoc ELSE
%right ASSIGN
%left EQ NEQ
%left LT GT LEQ GEQ
%left PLUS MINUS
%left TIMES DIVIDE MODULO
%right NOT NEQ
%right RBRACKET
%left LBRACKET
%left INCREMENT DECREMENT
%right DOT
%right ARROW

%start program
%type <Ast.program> program

/* Context-Free Grammar */
/* ------------------------ */

program: constituents EOF { Program(List.rev $1.includes, List.rev $1.specs, List.rev $1.cdecls, List.rev $1.fdecls) }

class constituents: {
    { 
        includes = []; 
        specs = []; 
        cdecls = []; 
        fdecls = []; 
    } 
    | constituents include_stmt { 
        includes = $2 :: $1.includes; 
        specs = $1.specs; 
        cdecls = $1.cdecls; 
        fdecls = $1.fdecls; 
    } 
    | constituents sdecl { 
        includes = $1.includes; 
        specs = $2 :: $1.specs; 
        cdecls = $1.cdecls; 
        fdecls = $1.fdecls; 
    } 
    | constituents cdecls { 
        includes = $1.includes; 
        specs = $2 :: $1.specs; 
        cdecls = $2 :: $1.cdecls; 
        fdecls = $1.fdecls; 
    } 
    | constituents fdecls { 
        includes = $1.includes; 
        specs = $1.specs; 
        cdecls = $1.cdecls; 
        fdecls = $2 :: $1.fdecls; 
    } 

    /* Includes */
    /* -------- */

    include_stmt: 
        INCLUDE STRING_LIT 
            { Include($2) }

    /* Functions */
    /* -------- */

    fdecl:
        DEF ID ASSIGN LPAREN formals_opt RPAREN COLON datatype LBRACE stmts RBRACE { 
            fname = $2;
            ftype = FunctionType(snd $5, $8);
            return_t = $8;
            formals = fst $5;
            body = $10;
            scope = Public;
            overrides = false;
            root_cname = None;
        }

    /* Specs */
    /* ------ */

    sdecl:
        SPEC TYPE ID LBRACE RBRACE { 
            sname = $2;
        }
cdef:
    CLASS_TYPE_ID ASSIGN LBRACE cbody RBRACE { 
        cname = $2;
        extends = NoParent;
    }
}

cbody:
    /* nothing */ { 
        fields = [];
        methods = [];
    }
    | cbody field { 
        fields = $2 :: $1.fields;
        methods = $1.methods;
    }
    | cbody cdef { 
        fields = $1.fields;
        methods = $2 :: $1.methods;
    }
}
cdef:
    scope DEF ID ASSIGN LPAREN formals_opt RPAREN COLON datatype LBRACE stmts RBRACE { 
        fname = $3;
        ftype = Functiontype(snd $6, $9);
        return_t = $9;
        formals = fst $6;
        body = $11;
        scope = $1;
        overrides = false;
        root_cname = None;
    }
}

/* Datatypes */
/* ------------ */
datatype:
    type_tag       { Datatype($1) }
    | array_type    { $1 }
    | function_type { $1 }

type_tag:
    primitive      { $1 }
    | object_type   { $1 }
     /* AST Datatype */

primitive:
    INT
    | FLOAT { Float_t } 
    | CHAR   { Char_t } 
    | BOOL   { Bool_t } 
    | UNIT   { Unit_t } 

object_type:
    TYPE_ID { Object_t($1) }
     /* AST Arraytype */

array_type:
    type_tag LBRACKET brackets RBRACKET { Arraytype($1, $3) }

brackets:
    /* nothing */ { accidentally }
    | brackets RBRACKET LBRACKET { $1 + 1 }
     /* AST Functiontype */

    /* Typel->Type2 is shorthand for (Typel)-->Type2 */
    /* NOTE: ARROW is right-associative */
    function_type:
        LPAREN formal dtypes list RPAREN ARROW datatype { Functiontype($2, $5) }
        | datatype ARROW datatype { Functiontype([$1], $3) }
     /* Fields */
     /* ------- */

field:
    scope VAR ID COLON datatype SEMI { Field($1, $3, $5) }

    /* Formals and Actuals */
    /* ------------------- */

    /* Formal Datatypes -- Nameless for Function Types */
    formal dtypes list:
        formal datatype { [$1] }
        | formal dtypes_list COMMA formal datatype { $3::$1 }

    formal datatype:
        { $1 }

    /* Formals -- Names & Datatypes for Functions */
    /* Returns (f, t), where f = list of formal and t = list of data_t */
    formals opt:
/* nothing */ { (List.rev (fst $1)), List.rev (snd $1)) }

formal_list:
| formal
| formal_list COMMA formal { (fst $3 :: fst $1), (snd $3 :: snd $1) }

formal:
| ID COLON datatype { (Formal($1, $3), $3) }

// Actuals -- Exps evaluated for Function Calls */

actuals_opt:
| /* nothing */ { [] }
| actuals_list { List.rev $1 }

actuals_list:
| expr
| actuals_list COMMA expr { $3::$1 }

/* Scope */
/* ----- */

scope:
| /* nothing */ { Public }
| PUBLIC { Public }
| PRIVATE { Private }

/* Literals */
/* -------- */
literals:
| INT Lit { IntLit($1) }
| FLOAT Lit { FloatLit($1) }
| TRUE { BoolLit(true) }
| FALSE { BoolLit(false) }
| CHAR Lit { CharLit($1) }
| STRING Lit { StringLit($1) }
| function Literal { $1 }
| THIS { This }

function literal:
| ANON LPAREN formsals_opt RPAREN COLON datatype LBRACE stmts RBRACE { lambda num := !lambda_num + 1;
|  functionLit({
|    fnum = "f" ^ string of int !lambda_num;
|    fttype = FunctionType(snd $3, $6);
|    return t = $6;
|    formsals = fst $3;
|    body = "$8; scope = Private;
|    overrides = false;
|    root_chame = None;
|  })
| })

bracket args:
| LBRACKET expr { [$2] }
| bracket_args RBRACKET LBRACKET expr { $4 :: $1 }

/* Statements */
/* --------- */

stmts:
| stmt_list { List.rev $1 }

stmt_list:
| stmt { $1 }
| stmt_list stmt { $2::$1 }

stmt:
| expr SEMI { Expr($1) }
| RETURN SEMI { Return(Noexpr) }
| RETURN expr SEMI { Return($2) }
| LBRACE stmts RBRACE { Block($2) }
| IF LPAREN expr RPAREN stmt ELSE stmt { If($3, $5, $7) }
| WHILE LPAREN expr RPAREN stmt { While($3, $5) }
| VAR ID COLON datatype SEMI { Local($2, $4, Noexpr) }
| VAR ID ASSIGN expr SEMI { Local($2, Any, $4) }
| VAR ID COLON datatype ASSIGN expr SEMI { Local($2, $4, $6) }
| IF LPAREN expr RPAREN stmt NOELSE { If($3, $5, Block([])) }
| IF LPAREN expr_opt SEMI expr opt SEMI expr_opt { For($3, $5, $7, $9) }
| BREAK SEMI { Break }
| CONTINUE SEMI { Continue }

/* Expressions */
/* ------- */

expr_opt:
| /* nothing */ { Noexpr }
| expr { $1 }

expr:
| literals { $1 }
| expr INCREMENT { Binop($1, Add, IntLit(1)) }
| expr DECREMENT { Binop($1, Sub, IntLit(1)) }
| expr PLUS expr { Binop($1, Add, S3) }
| expr MINUS expr { Binop($1, Sub, S3) }
```ocaml
(* Stop Abstract Syntax Tree *)

type op = Add | Sub | Mult | Div | Modulo | And | Or |
        Equal | Neq | Less | Leq | Greater | Geq

type uop = Neg | Not

type primitive = Int_t | Float_t | Bool_t | Char_t | Unit_t | Object_t of string

type scope = Private | Public

type extends = NoParent | Parent of string

(* Functions *)

(* -------- *)

type fdecl = {
  fname : string;
  ftype : datatype;
  return t : datatype;
  formals : formal list;
  body : stmt list;
  scope : scope;
  overrides : bool;
  root cname : string option;
}

(* Specs *)

(* ------ *)

and spec = {
  sname : string;
}

(* Classes *)

(* -------- *)

and cbody = {
  fields : field list;
  methods : fdecl list;
}

and cdecl = {
  cname : string;
  extends : extends;
  cbody : cbody;
}

(* Datatypes, Formals, & Fields *)

(* i.e. Arraytype (a, 2) <= a[][]; (a, 3) <= a[][][] *)

(* Any : used for type of functions that take any datatype e.g. LLVM cast *)

(* NoFunctiontype : used for type of LLVM BuiltIn C Functions *)

datatype and datetype =
  Datatype of primitive
  | Arraytype of primitive * int
  | Functiontype of datatype list * datatype
  | NoFunctiontype
  | Any

(* Many : used for type of variable length functions e.g. LLVM printf *)

and formal = Formal of string * datatype | Many of datatype

and field = Field of scope * string * datatype

(* Fields *)

(* ------ *)

and expr =
  IntLit of int
  | FloatLit of float
  | BoolLit of bool
  | CharLit of char
  | StringLit of string
  | FunctionLit of fdecl


```
and stmt = 
| Block of stmt list
| Expr of expr
| Return of expr
| Local of string * datatype * expr
| If of expr * stmt * stmt
| For of expr * expr * stmt
| While of expr * stmt
| Break
| Continue

and var = Var of datatype * string

and include_stmt = Include of string

(* Program Definition *)
(* ------------------- *)

type constituents = {
  includes : include_stmt list;
  specs : spec list;
  cdecls : cdecl list ;
  fdecls : fdecl list;
}
type program = Program of include_stmt list * spec list * cdecl list * fdecl list

(*
type directive = Include of include_stmt
type constituent = Spec of spec | Class of cdecl | Function of fdecl
type program = Program of directive list * constituent list
*)

util.ml

(* Util *)
(* ------ *)

(* Collection of utilities used in other modules (e.g. pretty printing, tokenization, etc. *)

open Ast
open Parser
open Sast
open Core.Std

module E = Exceptions

(* Tokens *)
(* ------ *)

let string_of_token = function
  | SEMI   -> "SEMI"
  | LBRACK -> "LBRACK"
  | RBRACK -> "RBRACK"
  | LBRACE -> "LBRACE"
  | RBRACE -> "RBRACE"
  | LPAREN -> "LPAREN"
  | RPAREN -> "RPAREN"
  | NOT    -> "NOT"
  | CARET  -> "CARET"
  | MODULO -> "MODULO"
  | EQ     -> "EQ"
  | NEQ    -> "NEQ"
  | LT     -> "LT"
  | LEQ    -> "LEQ"
  | GT     -> "GT"
  | GEQ    -> "GEQ"
  | TRUE   -> "TRUE"
  | FALSE  -> "FALSE"
  | AND    -> "AND"
  | OR     -> "OR"
  | IF     -> "IF"
  | ELSE   -> "ELSE"
  | FOR    -> "FOR"
  | WHILE  -> "WHILE"
let rec token list to string = function
    (token, tail) ->
        string of token token ^ " " ^
        token list to string tail
    | [] -> "\n"

(* Parsing Error Functions *)
(* ----------------------- *)

let error string of file filename =
    if filename = "" then "Stdin"
    else "File " ^ filename ^ "\n"

let error string of cnun cnun token =
    string of int cnun ^ "\n"
    ^ string of int (cnun + String.length (string of token token))

(* Pretty-printing Functions *)
(* -------------------------- *)

let string of op = function
    Add -> "+
    Sub -> "-
    Mult -> "*
    Div -> "/
    Modulo -> "%"
    Ast.Equal -> "=="
    Neq -> "!="
    Ast.Less -> "<"
    Leq -> "<="
    Ast.Greater -> ">
    Geq -> ">="
    And -> "&&"
    Or -> "||"

let string of uop = function
    Neg -> "-"*
    Not -> "!"

let string of primitive = function
    Int t -> "Int"
    Float t -> "Float"
    Bool t -> "Bool"
    Char t -> "Char"
    Unit t -> "Unit"
    Object t(s) -> s

let rec print brackets = function
    1 -> "[
    i -> "[" ^ print brackets (i - 1)

let rec string of datatype = function
    Datatype(p) -> string of primitive p
    | Arraytype(p, i) -> string of primitive p ^ print brackets i
    | Functiontype(formal_dtypes, rtype) ->
        "(" ^
        String.concat -sep": " (List.map -f:string of datatype formal_dtypes) ^ ")" ^
        string of datatype rtype
    | Any -> "Any"

let string of scope = function
    Public -> "public"
    Private -> "private"
let string_of_program = function
Program(includes, spec, cdecls, fdecls) ->
  String.concat -sep:"\n" (List.map -f:string_of_includes includes) ^ "\n"^ 
  String.concat -sep:"\n" (List.map -f:string_of_spec spec) ^ "\n"^ 
  String.concat -sep:"\n" (List.map -f:string_of_cdecl cdecls) ^ "\n"^ 
  String.concat -sep:"\n" (List.map -f:string_of_fdecl fdecls)

(* SAST Printing Functions *)
(* ---------------------- *)

let rec string_of_bracket sexpr = function
  [] -> "" | head :: tail -> "[" ^ (string_of_expr head) ^ "]" ^ (string_of_bracket sexpr tail)

and string_of_sarray Primitive = function
  [] -> "" | [last] -> (string_of_expr last) | head :: tail -> "[" ^ (string_of_sarray Primitive head) ^ "", " ^ (string_of_sarray Primitive tail)

and string_of_expr = function
SIntLit(i) -> string_of_int i | SFloatLit(f) -> string_of_float f | SBoolLit(b) -> if b then "true" else "false" | SCharLit(c) -> Char.escaped c | SStringLit(s) -> "\"" ^ (String.escaped s) ^ "\"" | SFUnLit(s, data t) -> s ^ "": ^ string_of_datatype data t | SId(s, e) -> s | SBinop(op, e1, e2, _ ) -> (string_of_expr e1) ^ "" ^ (string_of_op op) ^ "": ^ (string_of_expr e2) | SApply(op, e1, e2) -> (string_of_op op) ^ "": ^ (string_of_expr e1) ^ "" ^ (string_of_expr e2) | SObjAccess(e1, e2, data t) -> (string_of_expr e1) ^ "": ^ (string_of_datatype data t) | SOpCall(ss, e1, _ , _ ) -> (string_of_expr ss) ^ "": ^ (string_of_bracket sexpr e1) | SArrayAccess(se, se_l, l) -> string_of_expr se ^ "": ^ string_of_bracket sexpr se_l ^ ""]" | SArrayCreate(d, se_l, _ , _ ) -> string_of_datatype d ^ "": ^ string_of_bracket sexpr se_l ^ ""]" | SMethod -> "" | SThis() -> "this"

and string_of_local sexpr = function
SMethod -> "" | e -> "": ^ string_of_expr e

and string_of_stmt indent =
let indent_string = String.make indent \"\t\" in
let get_stmt_string = function
SBlock(stmts) -> indent_string ^ "\n" ^ (String.concat -sep:"\n" (List.map -f:string_of_stmt indent+1) stmts) ^ indent_string ^ "\n" | SExpr(expr, data t) -> indent_string ^ "": ^ string_of_datatype data_t ^ "\n" | SReturn(expr, _ ) -> indent_string ^ "return" ^ "": ^ string_of_expr ^ "\n" | SIf(expr, SBlock(SExpr(SExpr, _ ))), -> indent_string ^ "if" ^ (string_of_expr e ^ ")\n" ^ (string_of_stmt indent+1) s | SIf(e, s1, s2) -> indent_string ^ "if" ^ (string_of_expr e ^ ")\n" ^ (string_of_stmt indent+1) s1 ^ indent_string ^ "else\n" ^ (string_of_stmt indent+1) s2 | SFor(e1, e2, e3, _ ) -> indent_string ^ "for" ^ (string_of_expr e1 ^ "": ^ (string_of_expr e2 ^ "": ^ (string_of_expr e3 ^ "\n" ^ (string_of_stmt indent) s | SWhile(e, _ ) -> indent_string ^ "while" ^ (string_of_expr e ^ "\n" ^ (string_of_stmt indent) s | SBreak -> indent_string ^ "break;" | SContinue -> indent_string ^ "continue;" | SLocal(s, d, e) -> indent_string ^ s ^ "": ^ string_of_datatype d ^ string_of_local sexpr e ^ "\n" in
get_stmt_string

and string_of_sdecl sdecl =
  "function" ^ "": ^ sdecl.sfname ^ "": ^ (string_of_formal sdecl.sfforms) ^ "\n" ^ (string_of_datatype sdecl.sdtype) ^ (string_of_stmt & (SBlock(sdecl.sbody)) ^ "")\n"

and string_of_sdecl sdecl =
  "class" ^ "": ^ sdecl.sname ^ "": ^ (string_of_field sdecl.sfields) ^ "\n" ^ (string_of_datatype sdecl.sdtype) ^ (string_of_stmt & (SBlock(sdecl.sbody)) ^ "")\n"

and string_of_main main = match main with
  Some(sdecl) -> string_of_sdecl sdecl | None -> ""
sast.ml

(* Semantically Checked AST *)
(* ------------------------ *)
(* Resolves datatypes in exprs, stmt s *)

open Ast

type fgroup = User | Reserved

type sfdecl = {
  sfname : string;
  sreturn t : datatype;
  srecord_vars : (string * datatype) list;
  sformals : formal list;
  sbody : stmt list;
  fgroup : fgroup;
  overrides : bool;
  source : string option;
  sftype : datatype;
}

and sfdecl = {
  sname : string;
  sffields : field list;
  sfdecls : sfdecl list;
}

and sprogram = {
  classes : sfdecl list;
  functions : sfdecl list;
  main : sfdecl;
}

and sexpr =
  SIntLit of int |
  SFloatLit of float |
  SBoolLit of bool |
  SCharLit of char |
  SStringLit of string |
  SFunctionLit of string * datatype |
  SId of string * datatype |
  SUnop of uop * sexpr * datatype |
  SBinop of sexpr * op * sexpr * datatype |
  SAssign of sexpr * sexpr * datatype |
  SCall of sexpr * sexpr list * datatype * int |
  SObjAccess of sexpr * sexpr * datatype |
  SArrayAccess of sexpr * sexpr list * datatype |
  SArrayCreate of datatype * sexpr list * datatype |
  SThis of datatype |
  SNoexpr |

and stmt =
  SBlock of stmt list |
  SExpr of sexpr * datatype |
  SReturn of sexpr * datatype |
  SIf of sexpr * stmt * stmt |
  SFor of sexpr * sexpr * sexpr * stmt |
  SWhile of sexpr * stmt |
  SLocal of string * datatype * sexpr |
  SBreak |
  SContinue

scanner.ml

(* Ocamlllex scanner for Stop Language *)

{ open Core.Std
 open Parser
 module E = Exceptions

 let lineno = ref 1
 let depth = ref 0
 let filename = ref ""

 let unescape s =
   Scanf sscanf ("\n\n" ^ s ^ "\n") "%S%" (fun x -> x)
}

(* Helper Regexes *)

 let whitespace = [' ' | '	' | '']

 let alpha = ['a'-'z' | 'A'-'Z']

 let upper alpha = ['A'-'Z']
let lower_alpha = ['a'..'z']

let digit = ['0'..'9']
let exp = (['E' | 'e'] | ['-'] | ['+'])?digit+
let ascii = ['\' | '#'] | ['\' | ' ' | 'n' | 'r' | 't']

(* Literals *)
let int_lit = digit* as lit
let float_lit = (digit* exp?) | (digit *? digit exp) | digit* as lit
let char_lit = ascii as lit
let escape_char_lit = 'l' (escape_char as lit)
let string_lit = (ascii | escape_char)* as lit
let id = lower_alpha | digit | '_'* as lit
let typeid = upper_alpha | (alpha | digit | '_')* as lit

token = parse

  whitespace = 
    | /\/\* { single_comment lexbuf }
    | /\* { incr_depth; multi_comment lexbuf }
    | /\* \n { incr_linen; token lexbuf }
    | /\* (' ' | RPAREN | ']' | LBRACE | '
' | LBRACKET | '}' | RBRACKET | COLON | SEMI | COMMA | DOT |

(* Operators *)

  INCREMENT | DECREMENT | PLUS | MINUS | TIMES | DIVIDE | ASSIGN | CARET | MODULO | EO | NÉO | LT | LEQ | GT | GEQ | AND | OR | NOT |

(* Misc *)

  ARROW | FATARROW | PUBLIC | PRIVATE | ANON |

(* Conditionals *)

  IF | ELSE | FOR | WHILE | BREAK | CONTINUE | RETURN |

(* Reserved Keywords *)

  SPEC | CLASS | METHOD | DEF | VAR | TYPE | FINAL | THIS | EXTENDS | MATCH | CASE |

(* Processor Directives *)

  INCLUDE | MODULE |

(* TYPES *)

  INT | FLOAT | BOOL | CHAR | UNIT |

(* PRIMITIVE LITERALS *)

  TRUE | FALSE |
int_lit = INT LIT(int of string lit)
float_lit = FLOAT LIT(float of string lit)
char_lit = CHAR LIT(lit)
and single comment = parse
\n'\n'   { incr lineno; token lexbuf }
|_   { single_comment lexbuf }

and multi comment = parse
\n'\n'   { incr lineno; multi_comment lexbuf }
|**   { incr depth; multi_comment lexbuf }
|**   { decr depth; if (depth > 0) then multi_comment lexbuf
|_   { multi_comment lexbuf }

* Generator.ml *

open Parser
module E = Exceptions

type token attr = {
  lineno : int;
  cnum : int;
}

let filename_ref = ref ""
let lineno_ref = ref 1
let cnum_ref = ref 1
let last_token_ref = ref EOF

(* Build an OCaml List of the tokens returned from the Scanner *)

let build_token_list filename lexbuf =
  Scanner.filename := filename;
  let rec helper lexbuf token list =
    let token = Scanner.token lexbuf in
    let lineno = Scanner.lineno in
    let cnum = (Lexing.lexeme_start_p lexbuf).Lexing.pos_cnum in
    match token with
    | t  -> (t, {lineno = lineno; cnum = cnum}) :: token list
    | EOF as eof -> (eof, {lineno = lineno; cnum = cnum}) :: token list
    in
  helper lexbuf []

(* Build an AST by feeding the Scanner's tokens into the Parser *)

let build ast filename token list =
  let token_list = ref(token_list) in
  let tokenizer =
    match !token_list with
    | (head, attr) :: tail ->
      let filename_ref := filename;
      let lineno_ref := attr.lineno;
      let cnum_ref := attr.cnum;
      let last_token_ref := head;
      let token_list := tail;
      let token =
        match head with
        | L.Value (start, end, t, v) -> t
        | L.Iface -> 11
        | L.Impl -> 18
        | _ -> L.error
        in
      (head, attr) :: tail
    | [] -> raise E.MissingEOF
  in
  let program = Parser.program tokenizer (Lexing.from_string "") in
  program

* Legacy Code *

(* Old Codegen *)

let translate ast = match ast with
  | A.Program(include, specs, classes, functions) ->
    let context = L.global context () in
    let the_module = L.create module context "Stop"
    and 132 t = L.132 type context
    and 18 t = L.18 type context
    and 11 t = L.11 type context
    and str t = L.pointer type (L.18 type context)
    and void t = L.void type context

    let str_type = A.Array type(A.Char t, 1) in

    let ltye of prim = function
    | A.Int t -> 132 t
    | A.Float t -> 132 t
    | A.Bool t -> 11 t
    | A.Char t -> 18 t
    (* TBD: Implement final struct function for Object_t *)
    | A.Unit t -> void_t

    in

    let rec ltye of array type array type = match array_type with
    A.Array type(p, l) -> L.pointer type (ltye of prim p)
let ltype of datatype = function
A.Datatype(p) -> ltype of prim p
| A.ArrayType(p, i) -> ltype of arraytype (A.ArrayType(p, i-1))
in

let ltype of formal = function
A.Datatype(p) -> p
| A.ArrayType(p, i) -> ltype of arraytype (A.ArrayType(p, i))
in

let atype of datatype = function
A.Datatype(p) -> ltype of datatype data_t in

(*) Declare print(), which the print built-in function will call *)
(* print() is already implemented in LLVM *)
let printf_t = L.var arg function type i32 t [ | L.pointer type i8 t | ] in
let printf_func = L.declare function “printf” printf_t the module in

(*) Define each function (arguments and return type) so we can call it *)
let function decls =
  let function decl = fdecl =
    let name = fdecl.A.fname
    and formal types =
      Array.of list (List.map (fun formal -> ltype of formal formal) fdecl.A.formals)
    in
    let ftype = L.function type (ltype of datatype fdecl.A.return t) 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, ...) = StringMap.find fdecl.A.fname function decls in
  let builder = L.builder_at_end context (L.entry_block the function) in
  let int_format_str = L.build global stringptr “\n” “fmt” builder in

(*) Construct code for an expression; return its value *)
let rec expr builder = function
  A.IntLit i -> L.const int i32 t i
  | A.FloatLit f -> L.const float i32 t f
  | A.BoolLit b -> L.const int i1 t (if b then 1 else 0)
  | A.CharLit c -> L.const int i8 t (Char.code c)
  | A.StringLit s -> L.build global stringptr s “tmp” builder
  | A.Id s -> raise E.NotImplemented
  | A.Binop (el, op, e2) -> build binop el op e2
  | A.Unop (op, e) -> build unop op e
  | A.Call (“printf”, e) -> build printf e
  | A.Call (s, e) -> raise E.NotImplemented
  | A.Noexpr -> L.const int i32 t 0

  and build binop el op e2 =
    let el1 = expr builder el
    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)
    el1 e2 “tmp” builder

  and build unop op e =
    let e' = expr builder e in
    (match op with
      | A.Neg -> L.build neg
      | A.Not -> L.build_not)
    e' “tmp” builder

  and build printf e =
    let format_str = match e with
      | [] -> A.Noexpr
      | hd :: tl -> hd
    and args = match e with
      | [] -> []
      | hd :: tl -> tl
    in
    let first arg = match args with
      | [] -> A.Noexpr
      | hd :: tl -> hd
    in
    let format lstr = match format_str with
      | A.StringLit(s) -> L.build global stringptr s “fmt” builder
      | _ -> raise E.PrintFirstArgNotString
    in
    let l_format_args_list = List.map (expr builder) args
    in
    let l_full_args_list = [format lstr] @ l_format_args_list
    in
    let l_args_arr = Array.of list l_full_args_list
    in
L.build_call print func l_args_arr "printf" builder

(* Invoke "f builder" if the current block doesn’t already
  have a terminal (e.g., a branch). *)

let add terminal builder f =
  match L.block terminator (L.insertion_block builder) with
  Some e -> ignore (expr builder e); builder
  | None -> ignore (f builder) in

(* Build the code for the given statement; return the builder for
the statement’s successor *)

let rec stmt builder = function
  A.Block sl -> List.fold_left stmt builder sl
| A.Expr e -> ignore (expr builder e); builder
| A.Return e -> build return e
| A.If (predicate, then stmt, else stmt) -> build if predicate then stmt else stmt
| A.While (predicate, body) -> build while predicate body
| A.For (el, e2, e3, body) -> build for el e2 e3 body
  and build return e =
    ignore (match fdecl.A.return t with
      A.Datatype (A.Unit t) -> L.build ret void builder
    | _ -> L.build ret (expr builder e) builder);

build if predicate then stmt else stmt =
  let bool_val = expr builder predicate in
  let merge bb = L.append block context "merge" the function in
  let then bb = L.append block context "then" the function in
  add terminal (stmt (L.builder at end context then_bb) then_stmt)
  (L.build br merge bb);
  let else bb = L.append block context "else" the function in
  add terminal (stmt (L.builder at end context else_bb) else_stmt)
  (L.build br merge bb);
  ignore (L.build cond br bool_val then_bb else_bb builder);
  L.builder at end context merge_bb

build while predicate body =
  let pred_bb = L.append block context "while" the function in
  ignore (L.build br pred_bb builder);
  let body_bb = L.append block context "while body" the function in
  add terminal (stmt (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
  let merge_bb = L.append block context "merge" the function in
  ignore (L.build cond br bool_val body_bb merge_bb pred_builder);
  L.builder at end context merge_bb

build for el e2 e3 body =
  stmt builder (A.Block [A.Expr el]; A.While (e2, A.Block [body; A.Expr e3]));

(* Build the code for each statement in the function *)

let builder = stmt builder (A.Block fdecl.A.body) in

(* Add a return if the last block falls off the end *)

add terminal builder (match fdecl.A.return t with
  A.Datatype (A.Unit t) -> L.build ret void
  | data_t -> L.build ret (L.const int (type_of_datatype data_t) 0)
) in

List.iter build function body functions;
the_module

analysis.ml

(* Semantic Analyzer for Stop Language *)

open Core.Std
open Ast
open Sast

module E = Exceptions
module G = Generator
module U = Util

module StringMap = Map.Make(String)
module StringsSet = Set.Make(String)

let seed_index = ref 0;

(* General String of List Function *)

let string_of_list string of item l = 
  "[" ^ String.concat ~sep; "", "(List.map ~f:string_of_item l) ^ "]"

let higher_order_sfdecls = ref StringMap.empty

(* Type of access link to pass to function *)

let access link types:(string, datatype) Hashtbl.t = Hashtbl.create ()
  ~hashable:STRING.hashable
  ~size:10
let access_link fnames: (string, string) Hashtbl.t = Hashtbl.create ()
- hashtbl: string hashtbl
- size: 10

(* Record which contains information re: Classes *)
type class record = {
  field map : field StringMap.t;
  method map : fdecl StringMap.t;
  cdecl : cdecl;
  (* constructor_map : Ast.fdecl StringMap.t; *)
}

(* Analysis Environment *)
(* Named vars = vars in scope *)
(* Record vars = vars to be placed in function activation record *)
type env = {
  env cname : string option;
  env creford : class record option;
  env cmmap : class record StringMap.t;
  env fname : string option;
  env fnmap : fdecl StringMap.t;
  env named vars : datatype StringMap.t;
  env record vars : datatype StringMap.t;
  env record_to_pass : (string * datatype) StringMap.t;
  env return t : datatype;
  env in for : bool;
  env in while : bool;
}

let update_env_cname env_cname env =
{
  env cname = env_cname;
  env creford = env.env creford;
  env cmmap = env.env cmmap;
  env fname = env.env fname;
  env fnmap = env.env fnmap;
  env named vars = env.env named vars;
  env record vars = env.env record vars;
  env record_to_pass = env.env record_to_pass;
  env return t = env.env return t;
  env in for = env.env in for;
  env in while = env.env in while;
}

let update_call_stack in for in while env =
{
  env cname = env.env cname;
  env creford = env.env creford;
  env cmmap = env.env cmmap;
  env fname = env.env fname;
  env fnmap = env.env fnmap;
  env named vars = env.env named vars;
  env record vars = env.env record vars;
  env record_to_pass = env.env record_to_pass;
  env return t = env.env return t;
  env in for = in for;
  env in while = in while;
}

let get_fname_exn fname_option = match fname_option with
  Some(s) -> s
| None -> raise E.UnexpectedNoFName

(* Name all methods <cname>.-<fname> *)
let get_method name cname fdecl =
let name = fdecl.fname in
  cname = "\".*" ^ name

let build_reserved_map =
  (* Note: type/for: printf has no functional equivalent *)
let reserved_stub fname return t formal s =
{
  sfname = fname;
  sreturn t = return t;
  ssformals = formals;
  ssrecord vars = [];
  sboody = [];
  sfgroup = Sast.Reserved;
  sourc = None;
  stype = NoFunctionType;
}

in
let i32 t = Datatype(Int t) in
let void t = Datatype(Unit t) in
let unit t = Arraytype(Char, t) in
let int t = FormaSize(t) in
let reserved list = [
  reserved stub "printf" void t [Many[Any]];
  reserved stub "malloc" str t [f "size" i32 t];
  reserved stub "calloc" Any [f "in" Any];
  reserved stub "sizeof" i32 t [f "in" Any];
  reserved stub "open" i32 t [f "path" str t; f "flags" i32 t];
  reserved stub "close" i32 t [f "fd" i32 t; f "buf" str t; f "nbyte" i32 t];
  reserved stub "read" i32 t [f "fd" i32 t; f "buf" str t; f "nbyte" i32 t];
  reserved stub "write" i32 t [f "fd" i32 t; f "buf" str t; f "nbyte" i32 t];
  reserved stub "lseek" i32 t [f "fd" i32 t; f "offset" i32 t; f "whence" i32 t];
  reserved stub "exit" (void t) ([f "status" i32 t]);
  reserved stub "getchar" (i32 t) ([]);
]
let rec expr_to_sexp e env = match e with
 (* Literals *)
 | IntLit(i) -> (SIntLit(i), env)
 | FloatLit(f) -> (SFloatLit(f), env)
 | BoolLit(b) -> (SBoolLit(b), env)
 | CharLit(c) -> (SCharLit(c), env)
 | StringLit(s) -> (SStringLit(s), env)
 | Id(s) -> (check_record_access s env, env)
 | This -> (Sid("this", get_this_type env), env)
 | Noexpr -> (SNoexpr, env)

 (* Operations *)
 | Unop(op, e) -> (check_unop op e env, env)
 | Binop(op, e1, e2) -> (check_binop e1 op e2 env, env)
 | Assign(e1, e2) -> (check_assign e1 e2 env, env)
 | Call(s, e l) -> (check call s e l env, env)
 | ArrayAccess(e, e l) -> (check array access e e l env, env)
 | ArrayCreate(d, e l) -> (check array create d e l env, env)
 | FunctionLit(f) -> (check function literal f env, env)
 | ObjAccess(e1, e2) -> (check obj access e1 e2 env, env)

(* Return Datatype for Binops with an Equality Operator (=, /=) *)
and get_equality_binop_type sel op se2 =
let type1 = expr_to_type exn sel in
let type2 = expr_to_type exn se2 in
match (type1, type2) with
| (Datatype(Char_t), Datatype(Int_t)) ->
  SBinop(sel, op, se2, Datatype(Bool_t))
| _ ->
  if type1 = type2
    then SBinop(sel, op, se2, Datatype(Bool_t))
    else raise (E.InvalidEqualityBinop(type1, type2))

(* Return Datatype for Binops with a Logical Operator (&&, ||) *)
and get_logical_binop_type sel op se2 =
let type1 = expr_to_type exn sel in
let type2 = expr_to_type exn se2 in
let operable = Set.of list [Datatype(Int_t); Datatype(Char_t); Datatype(Bool_t)]
  ~comparator: Comparator.Poly.comparator
in
if Set.mem operable type1 && Set.mem operable type2
then SBinop(sel, op, se2, Datatype(Bool_t))
else raise E.InvalidBinaryOperation

(* Return Datatype for Binops with a Comparison Operator (<, <=, >, >=) *)
and get_comparison_binop_type sel op se2 =
let type1 = expr_to_type exn sel in
let type2 = expr_to_type exn se2 in
let numerics = Set.of list [Datatype(Int_t); Datatype(Float_t); Datatype(Char_t)]
  ~comparator: Comparator.Poly.comparator
in
if Set.mem numerics type1 && Set.mem numerics type2
then SBinop(sel, op, se2, Datatype(Bool_t))
else raise E.InvalidBinaryOperation

(* TODO: Handle casting *)

(* Return Datatype for Binops with an Arithmetic Operator (+, *, -, /, %) *)
and get_arithmetic_binop_type sel op se2 =
let type1 = expr_to_type exn sel in
let type2 = expr_to_type exn se2 in
match (type1, type2) with
| (Datatype(Int_t), Datatype(Int_t)) -> SBinop(sel, op, se2, Datatype(Int_t))
| (Datatype(Float_t), Datatype(Float_t)) -> SBinop(sel, op, se2, Datatype(Float_t))
| _ -> raise E.InvalidBinaryOperation

(* Return Datatype for ID *)
and get_id_type s env =
  try StringMap.find exn.env.env_named_vars s
  with | Not_found ->
    ("
    StringMap.iter env.env_named_vars
    (fun k ->print_string (k ^ "="));
    raise (E.UndefinedId s)
  end

and get_this_type env = match env.env.cname with
| Some cname -> Datatype(Object_t(cname))
| None -> raise E.ThisUsedOutsideClass

and check_unop op e env =
let check_num unop op data t = match op with
...
Neg -> data t
  |   -> raise E.InvalidUnaryOperation
in
let check bool unop op = match op with
  Not -> Datatype(Bool t)
  |   -> raise E.InvalidUnaryOperation
in
let (se, env) = expr_to_sexpr e env
in
let data t = sexpr_to_type_exn se in
match data t with
  Datatype(Int t)
  Datatype(Char t) -> SUnop(op, se, check num unop op data t)
  Datatype(Bool t) -> SUnop(op, se, check BOOL unop op)
  |   -> raise E.InvalidUnaryOperation

and check binop el op e2 env =
  (* NOTE: may want to keep returned env *)
let (sel, _) = expr_to_sexpr el env
in
let (se2, _) = expr_to_sexpr e2 env
in
match op with
  Equal -> get_arity binop_type sel op se2
  And
  Or -> get_logical binop_type sel op se2
  Less
  Leq
  Greater
  Geq -> get_comparison binop_type sel op se2
  Add
  Mul
  Sub
  Div
  Modulo -> get_arithmetic binop_type sel op se2
  |   -> raise E.InvalidBinaryOperation

and check assign el e2 env =
  (* NOTE: may want to keep returned env *)
let (sel, _) = expr_to_sexpr el env
in
let (se2, _) = expr_to_sexpr e2 env
in
let type1 = sexpr_to_type_exn sel in
let type2 = sexpr_to_type_exn se2 in
match (type1, type2) with
  if type1 = type2
    then Assign(sel, se2, type1)
    else
      let str1 = U.string of datatype typel in
      let str2 = U.string of datatype typel2 in
      raise (E.AssignmentTypeMismatch(str1, str2))

(* TODO: Investigate Dice differences *)

and check call e l env =
  (* Add the correct activation record if the function takes one *)
let se l = expr_to_list e l env
in
let record_to_pass = StringMap.find env.env_record_to_pass s in

let se l = match record_to_pass with
  Some(tupple) ->
    let record_name = fst tupple in
    let record_type = snd tupple in
    let se = SID(record_name, record_type) in
    se :: se l
  | None -> se l
in
try
  (* Call the function if it is not a var *)
  let fdecl = StringMap.find exn env.env_fmap s in
  let return t = fdecl.return t in
  let sid = SID(s, fdecl.ttype) in
  SCall(sid, se l, return t, 0)
with | Not_found ->
  try
    (* Get the function pointer if it is a var *)
    let rhs_type = StringMap.find exn env.env_named_vars s in
    let return t = match rhs_type with
      FunctionType(_, return t) -> return t
    in
data t ->
      let data t = U.string of datatype data t in
      raise (E.CallFailedOnType data t)
    in
    let env fname = get fname env.env_fname in
    let record_type = Datatype(Object t {env_fname = "record"}) in
    let record_type_name = env fname ^ "record" in
    let record_class = StringMap.find exn env.env_cmap record_type_name in
    let lhs = SID(record_name, record_type) in
    let rhs = SID(s, rhs_type) in
    let sstmt = SObjAccess(lhs, rhs, rhs type) in
    SCall(sstmt, se l, return t, 0)
with | Not_found -> raise (E.UndefineFunction s)

and expr list to sexpr_list e l env = match e l with
  hd :: tl ->
    let (se, env) = expr_to_sexpr hd env
    in
    se :: expr list to sexpr_list tl env
  | [] -> []
let check access params = List.map se l
  ~f:(fun se -> match (sexpr_to_type exn se) with
    Datatype(Int t) -> ()
    | _ -> raise (E.ArrayAccess "Passed non-Int Indice Argument")
  )

let arr num indices = List.length e l
let arr num dims = match data t with
  Arraytype(_, n) -> n
  | _ -> raise (E.ArrayAccess "Passed non-Arraytype Variable")

let check nums_indices = if arr num dims <> arr num indices
  then raise (E.ArrayAccess "Number Indices != Number Dimensions")

SArrayAccess(se, se_l, data_t)

and check array create d e l env =
  let se_l = expr_list_to_seexpr_list e_l env in

  (* Check that the indice parameters are all Int_t *)
  let check access params = List.map se_l
    ~f:(fun se -> match (sexpr_to_type exn se) with
      Datatype(Int t) -> ()
      | _ -> raise (E.NonIntegerArraySize)
    )
  in

  let arr num indices = List.length e l
  let convert_d to arraytype = function
    Datatype(x) -> Arraytype(x, arr num indices)
    | _ -> raise (E.NonArraytypeCreate)
  in

  let sexpr_type = convert_d to arraytype d in
  SArrayCreate(d, se_l, sexpr_type)

and check function literal fdcl env =
  let f = StringMap.find exn env.env fmap (get_fname exn env.env.fname) in
  let link_type = Some(DatatypeObject t(f.fname ^ "record")) in

  let fdcld = convert fdcl to sfdecl env.env fmap.env.env fmap fdcl env.env named vars link_type env.env.record_to_pass

  let higher_order sfdecls = StringMap.add higher_order sfdecls <- key:fdcl.fname =data:sfdecl;
  SFunctionLit(sfdecl.sfname, sfdecl.sftype)

and check obj access e1 e2 env =
  let get cname exn = function
    Some(cname) -> cname
    | None -> raise E.CannotUseThisKeywordOutsideOfClass
  in

  let check lhs = function
    This -> SID("this", Datatype(Object t(get cname exn env.env.cname)))
    Id(s) -> check record access s env ("s IIs, get id_type s env")
    | _ as e -> raise E.LHSofObjectAccessMustBeAccessible
  in

  let check rhs e2 =
    let id = match e2 with
      Id s -> s
    | _ as e -> raise E.RHSofObjectAccessMustBeAccessible
  in

  let cname = match (check lhs e1) with
    SID(_, data t) -> (match data t with
      Datatype(Object t(name)) -> name)
    | SObjAccess(_, data t) -> (match data t with
      Datatype(Object t(name)) -> name)
    | _ -> raise E.RHSofObjectAccessMustBeAccessible
  in

  let crecord = StringMap.find exn env.env fmap cmap cname in

  try
    match StringMap.find exn crecord.field map id with
      Field(_, s, data t) -> SID(t, data t)
    with | Not_found -> raise E.UnknownClassVar
  in

  let lhs = check lhs e1 in
  let rhs_type = sexpr to type exn lhs in
  let rhs = check rhs e2 in

  let rhs t = match rhs with
    SID(_, data t) -> data t
  in

  SObjAccess(lhs, rhs, rhs_t)

(*
  StringMap.iter record_class.field map
  ~f:(fun ~key:s ~data:t -> print_string (s ^ "\n");

  let link_type = Hashtbl.find access link types fname in
  let print =match link type with
    Some(dt) ->
      print_string ("\n fname: " ^ fname ^ "\n");
      print_string ("type: " ^ U.string_of_datatype dt ^ "\n");
      print_string "===>\n"
    | None -> ()
  in
  print;
*)

(* Follow access links if var defined outside of function *)

and check record access s env =
let f_name = get_f_name_env env.env_f_name in

let rec build_rhs_helper f_name inner =
  let record_type = f_name ^ ".record" in
  let record_class = StringMap.find_env env.env_map record_type_name in
  if StringMap.mem record_class.field_map s then
    inner
  else
    let access_link = f_name ^ "+@link*" in
    let access_link_type = Hashtbl.find_env access_link_types f_name in
    let outer_name = Hashtbl.find_env access_link_names f_name in
    let inner = SObjAccess(inner, SId(access_link_name, access_link_type), access_link_type) in
    build_rhs_helper outer_f_name inner

in

let build_rhs f_name =
  let record_type = f_name ^ ".record" in
  let record_class = StringMap.find_env env.env_map record_type_name in
  let record_type = Datatype(Object.t(record_type_name)) in
  try
    ("Access the item if it is the current record."
    let = StringMap.find_env record_class.field_map s in
    let result = SId(record_name, record_type) in
    result

  with | Not_found ->
    (* Access the item through access links otherwise *)
    let access_link = f_name ^ "+@link*" in
    let access_link_type = Hashtbl.find_env access_link_types f_name in
    let outer_name = Hashtbl.find_env access_link_names f_name in
    build_rhs_helper outer_f_name
    (SObjAccess(SId(record_name, record_type), SId(access_link_name, access_link_type), access_link_type))

  in

let rhs_type = StringMap.find_env.env_env.named_vars s in
let rhs = SId(s, rhs_type) in
SObjAccess(lhs, rhs, rhs_type)

and arraytype_to_access_type data_t = match data_t with
  ArrayType(_, _) -> Datatype(p)
| _ -> raise E.UnexpectedType

and sexpr_to_type sexpr = match sexpr sexpr with
  SIntLit(_) -> Some(Datatype(Int.t))
| SFloatLit(_) -> Some(Datatype(Float.t))
| SBoolLit(_) -> Some(Datatype(Bool.t))
| SCharLit(_) -> Some(Datatype(Char.t))
| SStringLit(_, data_t) -> Some(data_t)
| SId(_, data_t) -> Some(data_t)
| SBinop(_, _, data_t) -> Some(data_t)
| SUnary(_, data_t) -> Some(data_t)
| SCall(_, data_t, _) -> Some(data_t)
| SObjAccess(_, data_t) -> Some(data_t)
| SAssign(_, data_t) -> Some(data_t)
| SArrayAccess(_, data_t) -> Some(data_t)
| SArrayCreate(_, data_t) -> Some(data_t)
| SThis(data_t) -> Some(data_t)
| SNoexpr -> None

and sexpr_to_type_exn sexpr = match (sexpr_to_type sexpr) with
  Some(t) -> t
| None -> raise E.UnexpectedNoexpr

(* Statement to SStatement Conversion *)
and check_sblock sl env = match sl with
  [] -> ([SBlock([SExpr(SNoexpr, Datatype(Unit.t))])], env)
| _ -> let (sl, l) = convert_stmt_list_to_sstmt_list sl env in
  ([SBlock(sl)], env)

and check_expr stmt e env
let se, env = expr_to_sexp e env in
let data_t = sexpr_to_type_exn se in
([SExpr(se, data_t)], env)

and check_return e env
let (se, ) = expr_to_sexp e env in
let data_t = sexpr_to_type_exn se in
match data_t, env.env_return_t with
  | Datatype(Unit.t), Datatype(Object.t()) ->
  (* TODO: See if this makes sense for Unit_t... *)
  Datatype(Unit.t), ArrayType() -> ([SReturn(se, data_t)], env)
| _ ->
  if data_t = env.env_return_t
  then ([SReturn(se, data_t)], env)
  else raise E.ReturnTypeMismatch
  (U.string of datatype data_t, U.string of datatype env.env_return_t, env.env_f_name)

and local_handler s data_t e env
  if StringMap.mem env.env.named_vars s
  then raise (E.DuplicateVar(s))
  else
    let (se, ) = expr_to_sexp e env in
    if se = SNoexpr then

let named_vars = StringMap.add env.env_named_vars
   ~key:s
   ~data:data_t;
in
let record_vars = StringMap.add env.env_record_vars
   ~key:s
   ~data:data_t;
in
let new_env = {
  env cname = env.env cname;
  env crecord = env.env crecord;
  env cmap = env.env cmap;
  env fname = env.env fname;
  env fmap = env.env fmap;
  env named vars = named vars;
  env record vars = record vars;
  env record to pass = env.env record to pass;
  env return t = env.env return t;
  env in for = env.env in for;
  env in while = env.env in while;
}

in
let save_obj_with_storage = (* Add the temp var as a local *)

  let var name = "tmp malloc var" (string_of_int !seed_index) in
  let var type = data_t in
  let ststm l = [SLocal(var name, var type, SNoexpr)] in
  let ststm id = SId(var name, var type) in
  let ststm record var = check record access s new env in
  let seexpr = SAssign(ststm record var, ststm id, var_type) in
  let ststm l = SExpr(seexpr, var_type :: ststm l in
  (List.rev ststm l, new_env)

in
(* Only allocate locals if they need to be allocated (pointer in activation record *))

seed_index := !seed_index + 1;

match data_t with
  | Datatype(Object t( )) -> save_obj_with_storage
  | _ -> ([SExpr(SNExpr, Datatype(Unit_t)]), new_env)
else
  let se_data_t = seexpr to type exn se in
  let is_assignable = function
    NoFunctionType
    Any -> false
    _ -> true
  in
  let valid assignment = function
    (Any, _) -> is_assignable se_data_t
    (data_t, se_data_t) -> if data_t = se_data_t
    then true else false
  in
  if valid assignment (data_t, se_data_t) then
    let named_vars = StringMap.add env.env_named_vars
       ~key:s
       ~data:se_data_t;
    in
    let record_vars = StringMap.add env.env_record_vars
       ~key:s
       ~data:se_data_t;
    in

    (* Record to pass *)
    let record to pass = match se with
      | SFunctionLit((), _) -> StringMap.add env.env_record to pass
        ~key:s
        ~data:se_data
      | _ -> env.env_record to pass
    in

    let new_env = {
      env cname = env.env cname;
      env crecord = env.env crecord;
      env cmap = env.env cmap;
      env fname = env.env fname;
      env fmap = env.env fmap;
      env named vars = named vars;
      env record vars = record vars;
      env record to pass = record to pass;
      env return t = env.env return t;
      env in for = env.env in for;
      env in while = env.env in while;
    }

    in
    let save object no storage =
      let lhs = check record access s new env in
      let seexpr = SAssign(lhs, se, se_data_t) in
      let ststm = SExpr(seexpr, se_data_t) in
      ([ststm], new_env)
    in
    save object no storage
    (* (SLocal(s, se_data_t, se), new_env) *)
else
  let type1 = U.string of datatype data_t in
  let type2 = U.string of datatype se data_t in

and parse_stmt stmt env = match stmt with
  | Block sl -> check_block sl env
  | Expr e -> check_expr stmt e env
  | Return e -> check_return e env
  | Local(s, data_t, e) -> local_handler s data_t e env
  | If(e, s1, s2) -> check_if e s1 s2 env
  | For(e1, e2, e3, s) -> check_for e1 e2 e3 s env
  | While(s, s) -> check_while e s env
  | Break -> check_break env
  | Continue -> check_contiue env

/* Semantically check a list of stmts: Convert to ssstms */
and convert_stmt_list to ssstmlist sl env =
  let rec iter = function
    | [] -> []
  |
    let ssstmlist = ((iter tail), !env_ref in
      ssstmlist)

and check if e s1 s2 env =
  let (se, _) = expr_to_seexpr e env in
  let t = sexpr_to_type_exn se in
  let (ifbody, _) = parse_stmt s sl env in
  let (elsebody, _) = parse_stmt s2 env in
  if t = Datatype(Bool t)
    then [[SIf(se, SBlock(ifbody), SBlock(elsebody))], env]
  else raise E.InvalidIfStatementType

and check for e1 e2 e3 s env =
  let old_in_for = env.env_in_for in
  let old_in_exists = env.env_in_for in
  let env = update_call_stack true env.env_in_for while env env in
  let (se1, _) = expr_to_seexpr e1 env in
  let (se2, _) = expr_to_seexpr e2 env in
  let (se3, _) = expr_to_seexpr e3 env in
  let (sboby, _) = parse_stmt s env in
  let conditional_t = sexpr_to_type_exn se2 in
  let sfor =
    if conditional_t = Datatype(Bool t)
      then SFor(se1, se2, se3, SBlock(sboby))
    else raise E.InvalidWhileStatementType
  in
  let env = update_call_stack old_in_for env.env_in_for while env in
  ([sfor], env)

and check while e s env =
  let old_in_for = env.env_in_for in
  let old_in_exists = env.env_in_for in
  let env = update_call_stack true env.env_in_for while env env in
  let (se, _) = expr_to_seexpr e env in
  let conditional_t = sexpr_to_type_exn se in
  let (sboby, _) = parse_stmt s env in
  let while =
    if conditional_t = Datatype(Bool t)
      then SWhile(se, SBlock(sboby))
    else raise E.InvalidWhileStatementType
  in
  let env = update_call_stack env.env_in_for old_in_while env env in
  ([while], env)

and check break env =
  if env.env_in_for [] env.env_in_while then
    ([SBreak], env)
  else raise E.BreakOutsideOfLoop

and check continue env =
  if env.env_in_for [] env.env_in_while then
    ([SContinue], env)
  else raise E.ContinueOutsideOfLoop

(* Map Generation *)
(* ============ *)
(* Generate StringMap: cname -> record *)
and build_record_map map cdecls fdecls =
  (* Check each constituent of a class: fields, member functions, constructors *)
  let helper m cdecl : Ast.cdecl =
    (* Check Fields *)
    let check_fields m field = match field with
      Field(scope, s, data_t) ->
        if StringMap.mem m s then raise (E.DuplicateField s)
        else StringMap.add m ~key:~data:(Field(scope, s, data_t))
      |
      (* Check Methods *)
    let method_name = get_method_name cdecl.cname in
    let check_methods m fdecl =
      if StringMap.mem m (method_name fdecl)
        then raise (E.DuplicateFunctionName (method_name fdecl))
      else if (StringMap mem fmap fdecl.fname)
        then raise (E.FunctionNameReserved fdecl.fname)
      else StringMap.add m ~key:~method_name fdecl ~data:fdecl
    in
    (* Check Class Name *)
    if (StringMap mem cdecl.cname) then raise (E.DuplicateClassName cdecl.cname)

(* Add Class Record to Map *)
else StringMap.add m
    ~key:cdeclcname
    ~data:
        field_map = List.fold_left cdecl.cbody.fields ~f:check fields ~init:StringMap.empty;
        method_map = List.fold_left cdecl.cbody.methods ~f:check methods ~init:StringMap.empty;
        cdecl = cdecl )
    in
    let record_map = List.fold_left cdecls ~f:helper ~init:StringMap.empty
    in
(* Add function Records *)
let discover named vars fdecl =
    let field_map = List.fold fdecl.formals ~f:(fun m formal -> match formal with
            Formal(s, d) -> (StringMap.add m ~key:s ~data:(Field(Public, s, d))))) ~init:StringMap.empty
    in
    let helper stmt = match stmt with
        Local(_, _, _) -> Some(s, Field(Public, s, d))
        _ -> None
    in
    List.fold fdecl.body ~f:(fun m stmt -> match (helper stmt) with
            Some(t) -> StringMap.add m ~key:(fst t) ~data:(snd t)
            _ -> m) ~init:field_map
    in
    let helper m (fdecl : Ast.fdecl) =
        let field_map = discover named vars fdecl in
        let field_map = try
            let link_type = Hashbl.find_exn access_link_types fdecl.fname in
            let link_name = fdecl.fname ^ ^ @link^ . in
            let field = Field(Public, link_name, link_type) in
            StringMap.add field_map ~key:link_name ~data:field
            with | Not_found -> field_map
        in
        let temp_class = {
            field_map = field_map;
            method_map = StringMap.empty;
            cdecl = {
                cfname = fdecl.fname ^ ^ .record^ ;
                extends = NoParent;
                cbody = {
                    fields = [];
                    methods = [];
                }
            }
        } in
        StringMap.add m ~key:(fdecl.fname ^ ^ .record^ )
        ~data:temp_class
    in
    List.fold_left fdecls ~f:helper ~init:record_map
(* Generate StringMap: fname -> fdecl *)
and build fdecl map reserved sfdecl map first_order fdecls =
    (* Check whether each function is already defined before adding it to the map *)
    let check functions m fdecl =
        if StringMap.mem m fdecl.fname
            then raise (E.DuplicateFunctionName fdecl.fname)
        else if StringMap.mem reserved sfdecl map fdecl.fname
            then raise (E.FunctionNameReserved fdecl.fname)
        else StringMap.add m ~key:(fdecl.fname) ~data:fdecl
    in
(* Add all the first order functions to the map *)
let map = List.fold_left first_order cdecls ~f:check functions ~init:StringMap.empty;
    in
(* DFS to discover all higher order functions *)
let rec discover higher order l fdecl =
    let check higher order helper l stmt = match stmt with
        Local(_, _, e) -> (match e with
            FunctionLit(nested fdecl) ->
                let link_t = Datatype(Object t(fdecl.fname ^ ^ .record^ )) in
                Hashbl.add exn access link types ~key:nested fdecl.fname
                ~data:link_t;
                Hashbl.add exn access link fnames ~key:nested fdecl.fname
                ~data:fdecl.fname;
                nested fdecl :: discover higher order l nested fdecl
            _ -> l)
        _ -> l
    in
    List.fold_left fdecl.body ~f:check higher order helper ~init:l
let higher_order_fdecls = List.fold_left first_order_fdecls
  \f -> discover_higher_order
  \f -> []

(* Add all the higher order functions to the map *)
let map = List.fold_left higher_order_fdecls
  \f -> check functions
  \f -> init:map

(* Add reserved functions to the map *)
let add reserved fdecls m key =
  let sdecl = StringMap.find_exn reserved_sdecl_map key in
  let fdecl = {
    fname = key;
    ftype = sdecl.sftype;
    return_t = sdecl.sreturn_t;
    formals = sdecl.sformals;
    body = [];
    scope = Public;
    overrides = false;
    root_cname = None;
  }
  StringMap.add m ~key:key ~data:fdecl

let fdecl_map = List.fold_left (StringMap.keys reserved_sdecl_map)
  \f -> add reserved fdecls
  \f -> init:map

let fdecls_to_generate = first_order_fdecls @ higher_order_fdecls

(* Conversion *)
(* =========== *)

(* Convert a method to a semantically checked function *)
(* Name = <root class>::<fname> *)
(* Propose instance of class to function parameters *)
and convert method to sdecl fmap cmap cname fdecl =
  let crecord = StringMap.find_exn cmap cname
  in
  let root_cname = match fdecl.root_cname with
    Some(c) -> c
    | None -> c
  in
  (* The class that the function takes as an additional formal *)
  let class formal =
    if fdecloverrides then
      Ast.Formal("this", Datatype(Object t(root_cname)))
    else
      Ast.Formal("this", Datatype(Object t(cname)))
  in
  let env param helper m formal = match formal with
    Formal(s, data_t) -> (StringMap.add m ~key:s ~data:formal)
    _ -> m
  in
  let env params = List.fold_left (class formal :: fdecl.formals)
    \f -> env param helper
  in
  let env = {
    env_cname = Some(cname);
    env_crecord = Some(crecord);
    env_cmap = cmap;
    env_fname = None;
    env_fmap = fmap;
    env_named vars = StringMap.empty;
    env_record vars = StringMap.empty;
    env_record to pass = StringMap.empty;
    env_return_t = fdecl.return_t;
    env_in for = false;
    env_in while = false;
  }

(* Assign fname to <fname> or <class>::<fname> appropriately *)
let fname = get_method name cname fdecl
  \f -> class formal :: fdecl.formals

(* Prepend the class as the first parameter to the function if it is a method *)
in
let fdecls_formals = class formal :: fdecl.formals

(* Check the global in the fbody *)
let (fbody, env) = convert stmt_list to stmt_list fdecl.body env

let record vars = StringMap.fold env.env.env.record vars
  \f -> for k v -> (k, data_t :: l)
  \f -> init:[]

{ sfname = fname;
  sreturn_t = fdecl.return_t;
  srecord vars = record vars;
  sformals = fdecl.formals;
  sbody = fbody;
  sgroup = Sast.Literal;
  overrides = fdecl.overrides;
(* Convert a function to a semantically checked function *)
and convert_fdecl_to_sfdecl fmap cmap fdecl named_vars link_type record_to_pass =

(* Add link, if the function is not first class *)
let sfmodels = match link_type with
  Some(t) -> let access_link = Formal(fdecl.fname ^ "@link", t) in access_link :: fdecl.formals
| None -> fdecl.formals
in
(* let sfmodels = fdecl.formals in *)

(* Add named values to env *)
let env param helper m formal = match formal with
  Formal(s, data t) ->
    if StringMap.mem named vars s
    then raise (E.DuplicateVar s)
    else StringMap.add m ~key:s ~data:data t
  | _ -> m
in
let named_vars = List.fold_left sfmodels
  -f:env param helper
  -init:named_vars
in
let record_vars = List.fold_left sfmodels
  -f:env param helper
  -init:StringMap.empty
in
let env = {
  env cname = None;
  env_crecord = None;
  env_cmap = cmap;
  env_fname = Some(fdecl.fname);
  env_fmap = fmap;
  env_named vars = named vars;
  env_record vars = record vars;
  env_record_to_pass = record_to_pass;
  env_return_t = fdecl.return_t;
  env_in for = false;
  env_in while = false;
}

(* Check the stmts in the fbody *)
let (sfbody, env) = convert_stmt_list_to_stmt_list fdecl.body env
in
let record_vars = StringMap.fold env.env record vars
  -f:(fun ~key:k ~data:(data t l) -> (k, data t :: l))
  -init:[]
in
let record_vars = match link_type with
  Some(t) -> let access_link = (fdecl.fname ^ "@link", t) in access_link :: record_vars
| None -> record_vars
in
(* Assign any parameters to their corresponding activation record vars *)
let field helper l f = match f with
  Formal(s, data t) ->
    let stmtt id = Sid(s, data t) in
    let stmtt_record_var = check_record_access s env in
    let sexpr = SAssign(stmtt_record_var, stmtt_id, data t) in
    SExpr(sexpr, data t :: l)
  | _ -> l
in
let sfbody = List.fold_left sfmodels
  -f:field helper
  -init:sfbody
in
(* Add activation record *)
let record_type = Datatype(Object t(fdecl.fname ^ ".record")) in
let record_name = fdecl.fname ^ "Record" in
let sfbody = SLocal(record_name, record_type, SNoexpr) :: sfbody in

(* Make sure the function has the correct type (prepend access link) *)
let sftype = match link_type with
  Some(t) -> (match fdecl.fctype with
    Functiontype(dt l, dt) -> Functiontype(t :: dt l, dt)
  )
| None -> raise E.FTypeMustBeFunctiontype
in

{ sfname = fdecl.fname;
  sreturn t = fdecl.return_t;
  srecord vars = record vars;
  sfmodels = sfmodels;
  sbody = sfbody;
  fgroup = Sast.User;
  overrides = fdecl.overrides;
  source = None;
  sftype = sftype; }
/* Generate activation records for fdecls */
let generate_sfdecl_records sfdecl =
  let fields = List.map sfdecl.record_vars
  in
  { sname = sfdecl.sfname ^ ".record";
    sfields = fields;
    sfdecls = [];
  }

/* Convert cdecls to sfdecls */
let convert_cdecl_to_sfdecl sfdecls (c:Ast.cdecl) =
  { sname = c.cname;
    sfields = c.cbody.fields;
    sfdecls = sfdecls;
  }

/* Generate Sast: sprogram */
let convert ast to sast
crecord map (cdecls : cdecl list)
fdecl map ((first order fdecls : fdecl list) (higher order fdecls : fdecl list))
let is_main = 
  (fun f ->
    match f.sfname with
    | s -> s = "main"
  )
let get main fdecls =
  let mains = List.filter (
    f:is_main fdecls
  )
  in
  if List.length mains < 1 then
    raise E.MissingMainFunction
  else if List.length mains > 1 then
    raise E.MultipleMainFunctions
  else
    List.hd exn mains
  in
let remove main fdecls =
  List.filter
    (fun f -> not (is_main f))
  fdecls
  in
let handle cdecl cdecl =
  let crecord = StringMap.find exn crecord_map cdecl.cname in
  let sfdecls = List.fold_left cdecl.cbody.methods
    ~f:(
      fun l f ->
        convert_method_to_sfdecl fdecl_map crecord_map cdecl.cname f
      )
    ~init:l
  in
  let sfdecls = remove_main sfdecls in
  let sfdecls = convert_cdecl_to_sfdecl sfdecls cdecl in
  (sfdecls, sfdecls)
  in
let iter cdecls t c =
  let sfdecls = handle cdecl c in
  (fst sfdecls :: fst t, snd sfdecls @ snd t)
  in
let (sfdecl_list, sfdecl_list) = List.fold_left cdecls
  ~f:iter cdecls
  ~init:([], [])
  in

(* Append first order fdecls to the tuple *)
let sfdecls = List.fold_left first_order fdecls
  ~f:(
    fun l f ->
      convert_fdecl_to_sfdecl fdecl_map crecord_map f StringMap.empty None StringMap.empty
    )
  ~init:l
  in

(* Append higher order fdecls to the tuple *)
let sfdecls = StringMap.fold
  higher_order sfdecls
  ~f:(
    fun ~key:k ~data:sfdecl l ->
      sfdecl :: l
    )
  ~init:sfdecls
  in
let (sfdecl_list, sfdecl_list) = (sfdecl_list, sfdecls @ sfdecl_list) in

(* Add Activation Record structs to the tuple *)
let sfdecls = List.fold_left sfdecl_list
  ~f:(
    fun l f ->
      (generate_sfdecl_records f)
    )
  ~init:l
  in
let (sfdecl_list, sfdecl_list) = (sfdecls @ sfdecl_list, sfdecl_list) in

let main = get main sfdecl_list in
let sfdecl_list = remove_main sfdecl_list in
{ classes = sfdecl_list;
  functions = sfdecl_list;
  main = main;
}

(* Analyze *)
(* TODO: Include code from external files *)
let analyze filename ast = match ast with
  | Program(includes, specs, cdecls, fdecls) ->
    let reserved_map = build_reserved_map in
    let (sfdecl_map, fdecls, first, higher) = build_fdecl_map reserved_map fdecls in
    let record_map = build_record_map reserved_map cdecls fdecls in
    let sast = convert ast to sast record_map record_map cdecls fdecl_map first higher in
    sast
/* Stop Exceptions */
exception InvalidOption of string
exception InvalidArg
exception NoFileArgument

/* Scanner Exceptions */
exception IllegalCharacter of string * string * int

/* Parser Exceptions */
exception CannotDefineVariableLengthArgFunction

/* Generator Exceptions */
exception MissingEOF

/* Sema Exceptions */
exception FTypeMustBeFunctiontype
exception ThisUsedOutsideOfClass
exception MissingMainFunction
exception MultipleMainFunctions
exception InvalidUnaryOperation
exception UnexpectedNoexpr
exception UnexpectedType
exception UnexpectedNoName
exception UnexpectedDatatype
exception UnexpectedNonBodyStmt
exception InvalidBinaryOperation
exception LHSofObjectAccessMustBeAccessible
exception RHSofObjectAccessMustBeAccessible
exception UnknownClassVar
exception CannotUseThisKeywordOutsideOfClass
exception InvalidIfStatementType
exception InvalidForStatementType
exception InvalidWhileStatementType
exception NonIntegerArraySize
exception NonArrayTypeCreate
exception CallFailedOnType of string
exception InvalidEqualityBinop of string * string
exception UndefinedId of string
exception DuplicateField of string
exception DuplicateClassName of string
exception DuplicateVar of string
exception FunctionNameReserved of string
exception ReturnTypeMismatch of string * string * string option
exception AssignmentTypeMismatch of string * string
exception LocalAssignmentTypeMismatch of string
exception LocalAssignmentTypeNotAssignable of string
exception ArrayAccess of string
exception UndefinedFunction of string
exception BreakOutsideOfLoop
exception ContinueOutsideOfLoop

/* Util Exceptions */
exception UtilError of string

/* Codegen Exceptions */
exception FieldIndexNotFound
exception PrintFirstArgNotString
exception PrintMissingArgs
exception NotImplemented
exception FloatOpNotSupported
exception IntOpNotSupported
exception UnopNotSupported
exception InvalidUnopEvaluationType
exception InvalidBinopEvaluationType
exception InvalidObjAccessType
exception InvalidStructType of string
exception InvalidStructType of string
exception InvalidDatatype of string
exception LLVMFunctionNotFound of string
exception FunctionWithoutBasicBlock of string
exception AssignmentLhsMustBeAssignable
exception ArrayLargerThan1Unsupported