ARBOL Programming Language

A High-Level Programming Language that Provides Easy Access for Tree Manipulation

Andrea Mary McCormick {amm2497}

Anthony Palmeira Nascimento {asp2199}

Derek Hui Zhang {dhz2104}

26 April, 2021

Table of Contents

1. Introduction

1.1 Inspiration

The world we live in today and the data we often seek are composed of intricate relationships and connections. As such, we recognized the importance of efficiency in terms of building and expressing this web of relationships among entities. Hence, we sought to exploit the versatility of the tree data structure within our programming language in order to facilitate an environment where users have the freedom to build, manipulate, and organize node-based connections with ease. Hence, not only did we decide to pursue a programming language that accomplishes just that, but we also named our language ARBOL, which is the Spanish word for "tree" – an appropriate reinforcement of the allurement of our language.

1.2 Overview

In the realm of programming there is a plethora of languages that enable the implementation of trees. However, in many of those cases such implementations are not native to the language, which poses particular challenges to inexperienced programmers. Thus, the purpose of the ARBOL language is to abstract much of the logic behind the tree data structure in order to reduce the complexity and promote easy access to such useful functionality.

Our language – which is primarily based upon the C programming language – includes built-in syntax in order to provide ease and usefulness for trees by comprising node data structures as well as elements of node-traversal abstraction behind-the-scenes. Therefore, it follows the paradigm of a primarily imperative programming language, where a user's code is written as a set of instructions that are executed sequentially as opposed to relying on function execution as the fundamental means of data manipulation and transformation in order to reach a desired outcome. The most fundamental operations provided through our tree-specific syntax include: node declaration, child assignment, node dereferencing, node assignment, and a get-child functionality that provides easy access to a given node's immediate children. With this new tree-specific syntax and the underlying layer of abstraction offered by ARBOL, users will be able to solve algorithmic tree data structure problems without having to worry about properly implementing the tree data structure itself.

1.3 Program Execution

ARBOL is syntactically similar to C, but utilizes syntactic sugar in order to promote efficient implementation and manipulation of trees which are composed of individual node data structures. Within the src folder, type **make clean**. Then, type **make**. This creates the ARBOL to LLVM compiler. Next, utilizing the shell script run_arbol.sh, type **bash ./run_arbol.sh [.arbol file name]**, which will successfully resolve the paths to the LLVM interpreter, LLVM compiler, C compiler, ARBOL compiler, and essentially feed a given test file to the ./arbol executable. The special void function named "main" in the given arbol file name will be run. The output of the given test file will then output to the standard output.

1.4 ARBOL Code Example

The following sample ARBOL code demonstrates the following features:

- Function declaration semantics of a user-declared (recursive) function called "**get_height**" which returns an integer and takes a node of integer type as an argument
- Various node-specific built-in syntax including: get right child operator (`[`), get left child operator (`]`), node initialization operator (`@`), node assignment operator (=>), and set child operators (`<--` , `-->`)
- The mandatory "**main**" function which doesn't take any parameters and doesn't include a return statement
- Calling the built-in **print_int** function, which takes an integer argument and prints its corresponding value to standard output
- Multiple node declarations (using `@` and => operators)

test-height1.arbol

```
function int get_height(int @root) {
    int lheight = 0;
    if ([root != null) {
        lheight = get_height([root);
    }

    int rheight = 0;
    if (]root != null) {
        rheight = get_height(]root);
    }
```

```
        if (lheight > rheight) {
            return lheight + 1;
        } else {
            return rheight + 1;
        }
    }

    function void main() {
        int @a => 1;
        int @b => 2;
        int @c => 3;
        int @d => 4;
        int @e => 5;

        a <-- b;
        a --> c;
        b <-- d;
        b --> e;

        print_int(get_height(a));
    }
```

In order to compile the sample code above, the user must type the following (assuming the file is saved to a .arbol file in the current directory):

```
$ make clean

$ make

$ bash ./run_arbol.sh ./test-height1.arbol
```

Then, the following expected output will appear in standard output, representing the height of the tree:

```
3
```

2. Language Reference Manual

2.1 Lexical Conventions

An ARBOL program is first tokenized by the lexical analyzer (scanner.ml) before passing along this stream of tokens to the parser (parser.mly). The lexer is responsible for tokenizing the input program into delimited identifiers, keywords, and literals. This section describes how the lexical analyzer carries out the process of tokenizing an ARBOL program. See sections 8.2 and 8.3 for additional details.

2.1.1 Comments

The character **#** introduces a single-line comment, which is terminated by the end of the line (‘\n’). The character **#*** introduces a multi-line comment, which is terminated with a corresponding ***#**.

2.1.2 Identifiers (Names)

An identifier includes some sequence of upper/lowercase letters, digits, and underscores ‘_’. The first character of an identifier must be alphabetic. Identifiers are case-sensitive.

`Examples:`

```
foo
foo_bar
tempChar
```

2.1.3 Keywords

The following identifiers are reserved for use as keywords, and may not be used otherwise (i.e. as variable or function names):

```
function
 return
   if
  else
   for
  while
continue
```

```
break
int
float
boolean
char
string
null
true
false
```

2.1.4 Literals

Literals are notations for constant values of some built-in types, and are as follows:

i. Integer Literal

An integer literal is a sequence of digits. An integer literal may begin with the optional unary operator minus sign (‘-’), which denotes a negative value, or it may begin with a digit. An integer literal must not begin with a zero if it contains more than one digit. All integer literals are interpreted as base-10 (decimal) numbers.

`Examples:`

```
89
-9
2537
0
```

ii. Float Literal

A float literal consists of a signed integer part, a decimal part, or a fraction part. Both the integer part and decimal part of a float literal include a sequence of digits. Either the integer part, or the fraction part (but not both) may be missing.

`Examples:`

```
-2.46
62.0
```

```
.98
2.3
```

iii. Boolean Literal

A Boolean literal has one of the two values: `true` or `false`.

iv. Char Literal

A char literal consists of a single character enclosed in single-quotes (‘ ’). The following escape sequences exist in the Arbol Language and are preceded by a back-slash ‘\’ :

\n	Newline
\t	Tab
\\	Backslash

`Examples:`

```
‘b’
‘’
‘P’
‘9’
```

v. String Literal

A string literal consists of a sequence of characters surrounded by double quotes (“ ”). A string has the type: array-of-characters. The compiler places a null byte (\0) at the end of each string in order to specify the end.

`Examples:`

```
“sunshine”
“”
“968”
```

2.2 Punctuation

The following punctuation conventions are recognized within the Arbol Language and are as follows:

`@`	Create new node object
`~`	Dereference node object to get the node value
`[`	Get left child
`]`	Get right child
`=>`	Node assignment
`<--`	Set left child
`-->`	Set right child
`()`	Expression precedence, conditional parameters, function arguments
`{}`	Statement block
`;`	End of statement
`,`	Separate arguments in function declaration

2.3 Operators

The following operators are listed from top to bottom in order of precedence with their corresponding associativity:

Top has the highest precedence:

Precedence	**Associativity**	**Description**	**Operator**

1	Left-to-right	Get left child, Get right child	[ ]
2	Left-to-right	Node dereference	~
3	Right-to-left	Logical NOT	!
3	Left-to-right	Left bracket	[
4	Left-to-right	Multiplication, Division, Modulo (remainder)	* / %
5	Left-to-right	Addition, Subtraction	+ −
6	Left-to-right	Relational greater than, greater than or equal to, less than, less than or equal to	> >= < <=
7	Left-to-right	Relational equal, not equal to	== !=
8	Left-to-right	Logical And	&&
9	Left-to-right	Logical Or	||
10	Right-to-left	Assignment equals, Node assignment	= =>

2.4 Whitespace

Instances of whitespace includes the following:

‘ ’ Blank character
\t Tab character
\n Newline character
\r Carriage return

3. Types

3.1 Primitive Types

There are five primitive types:

`int`	32-bit integer
`float`	32-bit single precision floating point type
`char`	8-bit data type used to store ASCII characters
`string`	sequence of chars
`boolean`	1-bit data type that is either ‘true’ or ‘false’
`void`	No value (usually used to signify the return type of a function that returns nothing)

3.2 Derived Types: `node` Type

All nodes in a tree have the derived `node` data type. This recursive data type specifies how we enter data and what type of data we enter. Each `node` data type includes a name, primitive type, value, and pointers to left/right children (if applicable).

This `node` type is built into our language. Just as most other languages have built-in syntax for creating arrays, our language has built-in syntax for creating trees. All nodes in a tree must have the same type.

The following code creates an integer node named `a` using the node initializer `@` that denotes a node object is being declared. Then, using the node assignment operator => the node object is assigned the value of 5. Remember, `a` is not of type integer; `a` is a `node` type and thus the value cannot be accessed directly. In order to get the value of `a`, the dereference symbol ~ must be used before the expression. However, this differs from

C-based pointer syntax because ARBOL doesn't support dereferencing and value re-assignment syntax within a single line (i.e. `~a = 5`). A user must use the node assignment operator => for value reassignment for a given node. This operator (=>) essentially includes both the dereferencing and node-value assignment functionalities in one.

```
# A node can be created as follows:

int @a => 5;

# Dereferencing node 'a' and value assignment not supported in the same line

~a = 5; # ERROR
a => 6;
```

In the following code, the dereference symbol (~) in the last line indicates that we are getting the value of `a`, rather than just the node itself. The assignment equals operator (=) is restricted to variable assignment for primitive types and for node-pointer assignment (discussed below).

```
# Dereferences node 'a' in order to be used in variable assignment:

int x = ~a;
int y = ~a + 3;
```

In the following code, the first line creates a new node of `int` type named `b` with a value of 5. However, the second line uses the assignment equals operator (=) for node-pointer assignment, where node `a` is initialized and assigned to point to the same node that was created in node `b`.

```
# Create a new node b and assigning node a to point to the same node that
# was created in node b:

int @b => 5;
int @a = b;
```

In the following code, the "`-->`" operator assigns the node on the right side as the right child of the node on the left side of the operator. The "`<--`" operator assigns the node on the left side as the left child of the node on the right side of the operator. Note that the nodes on both sides must have the same type, or an error will be returned.

```
# Create three nodes; assign c as the left child of a and b as the right child of a.

int @a => 1;
int @b => 2;
int @c => 3;
a --> b;
c <-- a;
```

Further, as described below, the] operator is used to get the right child of the current node, and the `[` operator is used to get the left child of the current node. These two operators return the node object, not the value, of the specified child node. In order to get the value, the node must also be dereferenced.

It's important to note that ARBOL supports node initialization using node-pointer assignment to a given node's right or left child-node (i.e. `int @b = [a;`). However, ARBOL doesn't support get-child operators in addition to node dereferencing within the same line (i.e. `int val = ~[a;`). A get-child operator must be used and pointed to by a node object before it can later be dereferenced in later steps in order to obtain its specific node-value.

```
#Given node a, gets the left and right child nodes of a

[a;
]a;

#Initializes nodes b, and sets node c to point to the same thing the left
child-node of node a points to


int @b = [a;

#Initializes node c and uses node-pointer assignment to assign node c to point
#to the the right child-node of a
```

```
#Gets the value of node c (which is also equal to the right child-node of a) and
#initializes integer val to its node value

int @c;
c = ]a;
int val = ~c;

#Initializes nodes x and y, and sets y as the right child-node of node x

int @x => 1;
int @y => 2;
x --> y;
```

4. Expressions

4.1 Literal Expression

```
literal
```

Literals are integer literals, float literals, boolean literals, character literals, and string literals.

4.2 Variable Expression

```
ID
```

A variable expression is the name of a variable defined previously in the same block, or in a block of higher scope (such as global variables).

4.3 Binary Operator Expressions

```
expression + expression
| expression - expression
| expression * expression
| expression / expression
| expression % expression
| expression == expression
| expression != expression
| expression < expression
| expression <= expression
| expression > expression
| expression >= expression
| expression && expression
| expression || expression
```

A binary operator expression takes in an expression on the left, and expression on the right, and returns a single expression in response. The following are expressions that include arithmetic, relational, logical, or assignment operators:

4.3.1 Equality Operators

This == and != operators tests for equality or non-equality between two expressions of any type, as long as they are the same type, and returns a boolean in response. Note that it is also possible to compare the left or right child of a node to the special variable “null,” which represents a null pointer.

```
expression == expression
expression != expression
```

The other relational operators can be used to compare between two ints or floats of the same type:

```
expression > expression
expression >= expression
expression < expression
expression <= expression
```

4.3.2 Arithmetic Operators

The following arithmetic operators (addition, subtraction, multiplication, division, and modulo, respectively) operate between ints, floats, and chars of the same type:

```
expression + expression
expression - expression
expression * expression
expression / expression
expression % expression
```

4.3.3 Logical Operators

Logical AND and OR operators can be used if both expressions are boolean type, and returns a boolean in response:

```
expression && expression
expression || expression
```

4.3 Unary Operator Expressions

```
-expression
| !expression
```

Unary expressions include the minus (-) operator, denoting a negative value, and the not (!) operator, which produces the opposite of the current value of the boolean expression.

Examples:

```
boolean a = true;
int b = 1;


#Boolean c is initialized to the opposite value of variable a (false)
#Integer d is initialized to the negative value of variable b (-1)

boolean c = !a;
```

```
int d = -b;
```

4.4 Assignment Expression

```
ID = expression
| ID => expression
```

An assignment and node assignment expression consists of a string on the left type, representing the ID to which to assign the value, and a value on the right side. The value returned from an assignment operation is the value of the right side of the expression. An assignment expression assigns a value to the variable on the left, while a node assignment expression specifically assigns a value to the node variable on the right.

Examples:

```
int a;
int @b;

int c = (a = 2) + 3;
int d = (b => 2) + 3;
```

In this example, variables c and d have the value of 5, because a 2 is returned from both the assignment operator (a = 2) and (b => 2). Moreover, due to the assignment operation, a has the value of 2, and b is a node which now points to the value of 2 as well.

4.5 Node Operation Expression

```
~ID
| [ID
| ]ID
```

A node operation expression takes in an operator and the variable name of a node, and returns either a node or a value, depending on the operation.

4.5.1 Dereference Operator

The dereference operator (~) dereferences the value of a node. For instance, in the following example, c is 8, as it is the value of b (5) added to 3.

```
int @b => 5;

int c = ~b + 3;
```

4.5.2 Get Child Operators

The get child operator gets the left ([) or right (]) child of a node. Note that the operator is always used before the variable name. In the following code, ~d will have the value of "c" as the left child of d is c, which has a value of "c." ~e will have the value of "b" as b is the right child of a, and b has a value of "b." Note that while the dereference operator returns a value that is the type of the node, the get child operators return the node itself.

```
string @a => "a";
string @b => "b";
string @c => "c";
a <-- c;
a --> b;
string @d = [a;
string @e = ]a;
```

4.6 Call Function Expression

```
ID(args_optional)
```

A call function expression calls a function and returns the value of the result. Note that the function can be anywhere in the same file; it does not have to appear chronologically before the function call in the same document. For instance, the below function will work, even though the max() function appears below its invocation:

```
function void main() {
    print_int(max(3, 4));
}
function int max(int a, int b) {
    if (a > b) {
        return a;
    }
    return b;
}
```

5. Functions

5.1 Function Declaration

```
function return_type ID(params_optional){
    statement_list
}
```

A function in ARBOL can take zero or more arguments. In this example, `add` takes two parameters of type `int`. Notice that the type comes *before* the variable name.

```
function int add(int x, int y) {
    return x + y;
}
```

Moreover, functions can take in nodes as the return value or as arguments to the function. The following function takes in a node of type string and returns the value of its left child.

```
function string get_leftchild(string @t) {
    string @x = [t;
    return ~x;
```

```
}
```

5.2 Built-in Functions

ARBOL comes with the following standard functions, which are linked via print.c, for convenience.

`print`	Prints a string with newline to standard output
`print_int`	Prints an integer to standard output
`print_float`	Prints a float to standard output

6. Statements

Statements are executed sequentially.

6.1 Expression Statements

```
expression;
```

Expression statements are mostly used as assignments or function calls.

6.2 Conditional Statements

```
if (expression) statement
| if (expression) statement else statement
```

Conditional statements are executed as follows (note that the else statement is optional):

```
if (x == 5) {
```

```
        return true;
    } else {
        return false;
    }
```

6.3 While Statements

```
    while (expression) statement
```

While statements can be written as follows::

```
    while(x < 5) {
        x = x + 1;
    }
```

6.4 For Statements

```
    for (expression; expression; expression)
    statement
```

For statements can be executed as follows:

```
    int i;
    for (x = 0; x < 20; x++) {
        print("Hi");
    }
```

The first expression in the for statement is an expression that is evaluated before the for loop is called. The second statement is called on every iteration of the loop. The third expression is called after each loop iteration, and before the second statement is called. Note that each section of code in the parentheses is expecting an expression, and thus no

variable creation can be done; instead, any variables must be created beforehand, as shown in the picture.

6.5 Break Statements

```
    break;
```

Like in C, break statements can be used to exit out of loops (for loops and while loops) statements early:

```
    for (int x = 0; x < 20; x++) {
        if (x == 5) {
            break;
        }
    }
```

6.6 Continue Statement

```
    continue;
```

Like in C, the continue keyword is used to indicate that we should continue to the next iteration of the control statement, as opposed to exiting the control statement early.

```
    function void main(){
        int i;
        for(i = 0; i < 5; i = i + 1) {
            if (i == 3) {
                continue;
            }
            print_int(i);
        }
    }
```

When the above code is executed, it produces the following result

```
0
1
2
4
```

6.7 Return Statement

```
return expression;
```

Always, only one value can be returned from a function. A return statement with an expression shall not appear in a function whose return type is void. A return statement without an expression shall only appear in a function whose return type is void.

6.8 Node Child Statement

```
ID --> expression;
ID <-- expression;
```

A node child statement causes the node variable on the left hand side to be assigned to have the left (<--) or right (-->) child of the right hand side, which is an expression that evaluates to a node variable. For instance, the following expression creates three nodes: a, b, and c. It assigns c to to be the left child of a, and b to be the right child of a.

```
function void main() {
    string @a => "a";
    string @b => "b";
    string @c => "c";
    a <-- c;
    a --> b;
}
```

6.9 Variable Initialization Statement

```
type ID = expression;
| type @ID => expression;
| type ID;
```

A variable initialization statement creates a new variable and adds it to the stack. If a variable is not a node, it can be assigned a value on the same line. If a variable is a node, it can either be assigned to point to another node, or be assigned a value on the same line as well using the node assignment syntax. Alternatively, a variable does not need to be assigned a value on the same line. Note that a node will be initialized to have a default value of 0 or an empty string, and the left and right children will be assigned the value of "null" on creation, as they initially point to null pointers.

```
function void main() {
    string @a => "a";
    string @b = a;
    string c = "c";
    if ([b == null) {
        print("hello");
    }
}
```

This program will print "hello," as the left child of b has not yet been initialized. It creates three variables: a, b, and c. Variable a is created a string node type and is assigned a value of a. Variable b points to the string node of a. Variable c is created as a string and is assigned the value of "c." ([b == null) is true because the left child of b has not yet been assigned.

Note that variables are scoped according to the block in which they are created. Variables created globally are viewable in any functions. Variables from the formal arguments and those created in the main body of a function are visible throughout the function. Variables created in an if/else, for loop, or while loop are only visible within that particular loop. Variables within the same block cannot share a common name, but variables in different blocks can have the same name.

Variable initialization is the only statement that can be made outside any other function, so that users can define global variables. However, global variables can only be assigned a value within functions; they can only be initialized outside of any functions.

6.9 Statement Block

```
    { statement_list }
```

A statement block is used to indicate a series of statements, used for control blocks like if statements, while loops, and for loops.

7. Project Plan

7.1 Planning Process

Throughout this process, our team met virtually to discuss this project one to two times a week in addition to constant email or text-messaging communication about minor implementation details. Given that our team is short one member, we decided to delegate individual responsibilities in pursuit of clearly-defined milestone deadlines for building the ARBOL compiler as opposed to strictly upholding our given roles within the team (i.e. Manager, Architect Design, Test-Suite Specialist, etc.). By maintaining flexibility in terms of our individual tasks, we were able to utilize an iterative approach in creating short-term goals in pursuit of our larger milestones. The specific milestones were defined based on upcoming deadlines that Professor Edwards imposed in order to help with the organization of this evolving process.

7.2 Specification Process

Initially, we envisioned the ARBOL Programming Language to include an expansive list of syntactic sugar, encouraging efficiency in terms of tree manipulation, as well as multiple built-in functions that would also be applicable. Such a list included binary-search tree capabilities, extensive child-manipulation capabilities that extended beyond a given node's direct child, built-in node traversal functionalities, and more. However, as we continued to iterate through this process, we realized that honing in on the primary building blocks that would provide users the most useful tree-based functionalities would be far more important. In other words, we decided that simplicity was key and quality over quantity in terms of the built-in node syntax would allow the user to utilize their own ingenuity in a broader sense.

Our first detailed specification included the lexical and syntactic qualifications that included keywords, primitive data types, various operator syntax, literals, expression and statement syntax, variable and function syntax, and more. All of these specifications were outlined within the lexer (scanner.mll) and parser (parser.mly) while we simultaneously outlined their importance within our Language Reference Manual. As a team, we collaborated and delegated tasks in discussing, composing, and refining the specifications outlined within our reference manual. Following the submission of our LRM, we frequently re-assessed the necessity of various specifications and prioritized our collective capacity to implement each. In particular, we decided that arrays, C-based structs, global variables, additional postfix operators, and nuances in our node syntax were all changes that needed to be made.

7.3 Development Process

We began this project by defining the primary specifications of the ARBOL programming language, which were implemented in the lexer and parser. Then, we proceeded to construct the semantics checker, where the specified types for each element within our program were clearly defined. Finally, we implemented the code generator, which took the semantic checker, and constructed LLVM IR to conclude the extent of our end-to-end compiler. We found that the type requirements outlined and maintained within the SAST proved to be the most tedious. Thus, we decided to prioritize this middle stage above most other steps before ensuring that the concluding code generator was implemented successfully.

7.4 Testing Process

Throughout the process of completing this project, we ensured that unit testing was central to our definition of success, and that it was clearly outlined within our collective milestones. In particular, we built out an extensive test-suite, inspired by the examples provided by Professor Edwards' MicroC program test suite. We continually added additional tests for functionalities as we saw fit. This allowed us to maintain unit testing at each stage within the compilation process. In particular, to overcome the grueling nature of implementing the SAST, we sought to analyze the intermediate data-structures (i.e. Abstract Syntax Tree) for each individual test, which allowed us to visually pinpoint any inconsistencies. As our program continued to evolve, we honed in on creating more concise test programs that would allow us to diagnose bugs most efficiently.

7.5 Team Responsibilities

Our team responsibilities were broadly divided across three members, and are outlined below. Although the specified roles represent the primary responsibilities expected of

each individual member, we allowed a range of flexibility in terms of delegating tasks and facilitating a more cohesive collaboration attempt in the creation of this project.

Team Member	Responsibility
Andrea McCormick	Compiler Front End, Code Generation, Automated Testing Suite
Anthony Palmeira Nascimento	Compiler Front End, Automated Testing Suite, CLI tool
Derek Hui Zhang	Compiler Back End, Semantic Checking, Code Generation

7.6 Project Timeline

February 3rd	Submitted *Project Proposal*
February 24th	Submitted *Language Reference Manual*
February 24th	Finished S*canner*
March 12th	Finished *Parser* and *AST*
March 24th	Submitted *Hello World Milestone*
April 16th	Revised and finished *SAST* and *Code Generator*
April 25th	Project Presentation
April 26th	Submission of *Final Project Report*

7.7 Development Environment

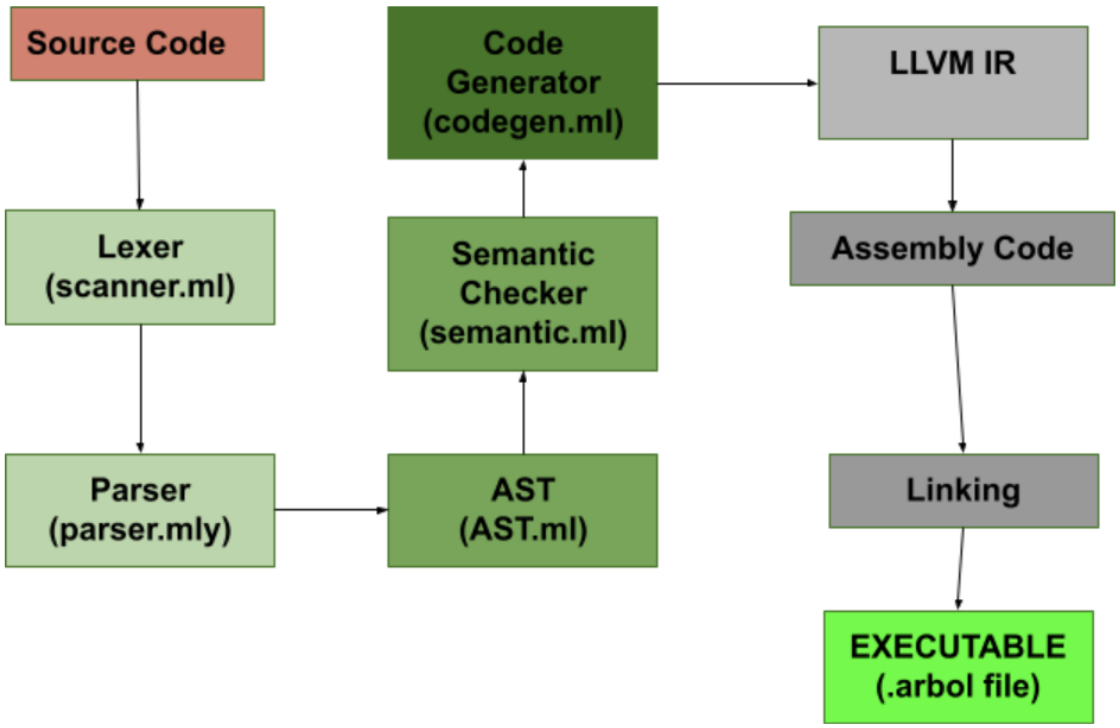
The following programming language and software infrastructure were utilized throughout the creation of our project.

- **Github-Hosted Git Repository** - for the purposes of version control, and it encompasses our compiler code, test-suite, Makefile, and shell script to facilitate efficient collaboration and testing
- **Python 3.8.5** - for writing our test program, which enables us to go through and check all our tests against the expected output
- **OCaml 4.2.01** - for parsing and implementing the semantic checker
  - Ocamlyacc and Ocamllex extensions were used for compiling the scanner and parser
- **LLVM 11.0.0** - for building the IR for our compiler in order to produce output code
- **VSCode** - for the development environment that was universally used by our team (also useful for Live Share via VSCode)

8. Architectural Design

8.1 The Compiler (Block Diagram)

The architecture of the ARBOL programming language compiler consists of the following components: Lexer, Parser, Semantic (Type) Checker, and Code Generator. The lexer and parser constitute the front-end of our compiler, and the semantic checker and code generator constitute the back-end.



8.2 Scanner

- Relevant Files: `scanner.mll`

The scanner is written in OCamllex, and it tokenizes an input ARBOL source program into identifiers, keywords, and literals. Also, the scanner removes all whitespace, and signifies an error for any tokens that are not syntactically valid (i.e. identifiers or literals). The tokens produced by the scanner are then fed into the Parser in order to produce an *Abstract Syntax Tree*.

8.3 Parser and AST

- Relevant Files: `parser.mly, ast.ml`

The parser is written in OCamlyacc, and it takes the tokenized data from the scanner and builds an *Abstract Syntax Tree* data structure that is built from the grammar defined in the `parser.mly` file and the specified data-types defined in the `ast.ml` file.

The grammar defined in `parser.mly` details production rules that seek to eliminate any ambiguities within the language. Successfully compiling the parser means that the given code for the language is syntactically (but not necessarily semantically) correct.

8.4 Semantic Checking

- Relevant Files: `semantic.ml, codegen.ml`

The semantic checking phase of compilation takes the *Abstract Syntax Tree* generated from the Parser, and recursively traverses the data-structure in order to transform it into a semantically-checked abstract syntax tree (*SAST*). Through this process, `semantic.ml` also builds a symbol table from all of the identifiers. A unique symbol table is created for each new block that is encountered, with a pointer to the outer block it is in. This symbol table verifies that each identifier that gets called has been formally declared, maintains its correct scope within the program, and has the correct data type. If a variable cannot be found in the current symbol table, the semantic checker looks for it in the outer block, up until the global scope. One of the most important functionalities of the semantic checker is its analysis of binary operators, where it ensures that the operands within a given expression have the same corresponding data type and there are no scoping violations, considering ARBOL doesn't support type-casting. We also make sure that all functions have the proper return type, that variables cannot be defined with type void, that all statements that require predicates (if, while, and for blocks) have a predicate expression (expression that evaluates to a boolean) in the right location, that breaks and continues

only occur within loops, and that all function calls call functions that have been defined and use the right type and number of arguments. Finally, we make sure that a main function has been defined.

8.5 Code Generator

- Relevant Files: `codegen.ml`

The code generator represents the final step in this compilation process, where it takes a semantically-checked Abstract Syntax Tree and translates each node in this tree into LLVM Intermediate Representation by traversing and generating code in a bottom-up fashion. A symbol table is maintained for each block; when a variable is called, codegen looks for it in a similar fashion as semantic, so we can get the proper address for a variable by looking for it in the innermost block possible. Throughout this process, basic blocks are constructed for control-flow statements. Also, this stage generates the code necessary to initialize a function instance first so that we can call them when we build their bodies.

9. Test Plan

9.1 Test Suites

As we continued to refine our compiler, we implemented individual test programs that pinpointed specific features that were added or modified in order to ensure their implementations were successful. For most of our implementations, we were intentional in creating two different test programs: one intended to pass and one intended to fail. As our program continued to expand, our tests seemingly became smaller and more concise. This allowed us to hone in on features such as binary and unary operators, variable assignment, function declaration and calls, scoping rules, etc.

Lastly, as a culminating step in our testing process, we created large ARBOL programs that served as an amalgamation of previously-tested small programs in order to verify that our implementations were well-suited for more sophisticated functionalities. Some of these larger programs we created to fail, while others were useful in testing the overall success of our program.

9.2 Test Automation

We created a CLI tool in Python for the purpose of unit-testing our language. This tool made it easier for us to compile, and check the outputs of all programs in the tests folder at once, or one at a time. We included a **fixtures.json** file that contained the expected

output for each of our tests. The CLI tool ran each given file and ultimately asserts the result against the fixtures file containing expected output.

Follow the instructions to install tool (need to be in the root directory):

1) First create a virtual environment

```
python3 -m venv env
```

2) Activate the virtual environment.

```
source env/bin/activate
```

3) Install CLI tool.

```
pip3 install --editable .
```

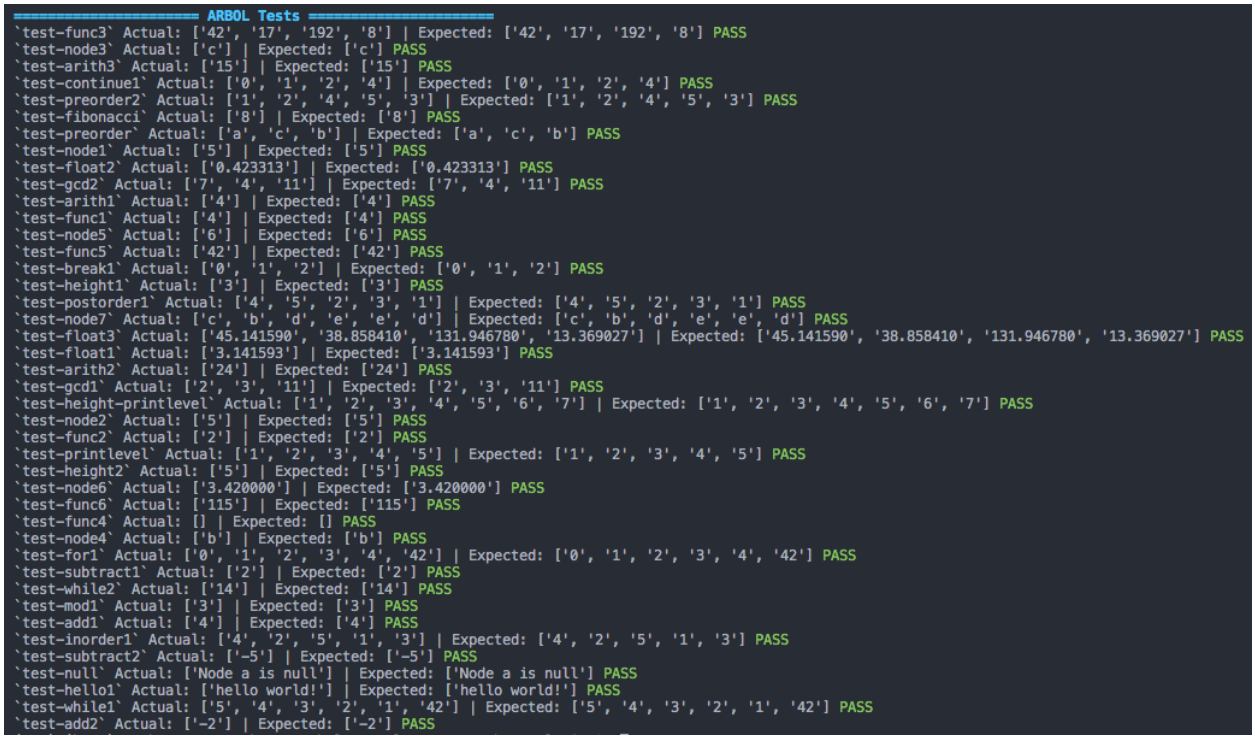
CLI tool runs all .arbol programs in tests folder by default:

```
arbol_test
```

To run a single .arbol file from tests folder you can add the following flag:

```
arbol_test -t <file_name>
```

CLI Tool Running Unit Tests:



10. Lessons Learned

10.1 Andrea McCormick

I learned that meeting to discuss taks, milestones, and deadlines for given stages of implementation for our project proved to be the bare-minimum that had to be accomplished. Instead, constant communication, instantaneous feedback, reports of challenges and realizations, and unexpected or last-minute collaboration or instructional sessions were all incredibly essential aspects in carrying out this project as a cohesive unit. Also, I learned that starting early was one of the most critical pieces of advice that I could've received (and should've taken to heart a little more) prior to embarking on this journey. Finally, I actually found functional programming (specifically OCaml) to be far more useful and (dare I admit) cooler than I anticipated.

10.2 Anthony Palmeira Nascimento

In working in OCaml I learned a completely new paradigm of programming. Shifting from traditional object-oriented programming to functional programming was not trivial, but the effort proved to be incredibly useful in terms of thinking recursively. I'd say,

however, one of the most challenging parts was understanding the LLVM, especially given that there is limited documentation available. In addition, debugging proved to be a grinding, sometimes cruel, yet valuable experience in understanding the underlying concepts of building a programming language. At the end of the day, seeing the expected outputs from a file containing your language's name was immensely rewarding.

10.3 Derek Hui Zhang

For this project, my primary responsibility was handling the semantic analysis and code generation portions. I realized that gaining a deeper understanding of LLVM, and specifically the code generation parts that we used as a reference for MicroC, was key to being able to implement all our desired features. Initially, I jumped in and tried to implement all the additional features without really understanding how LLVM worked, which caused me to waste a fair amount of time. In the end, I realized that understanding the architecture first, before trying to write any code, was the key to using LLVM effectively. It was quite interesting as LLVM's underlying code is written in C, yet it can be used in a functional programming language in OCAML. Moreover, though I had some previous experience with functional programming, most notably writing a Tetris AI in Haskell in a previous class taught by Professor Edwards, writing the semantic analysis and code generation portions gave me a better understanding of how to fully leverage OCAML's functional aspects.

11. Appendix

11.1 Scanner

`scanner.ml`

```
(* ARBOL Scanner *)
{ open Parser }

let digit = ['0'-'9']
let identifier = ['a'-'z''_']+['a'-'z''A'-'Z''_''0'-'9']*
rule token = parse
   (* Whitespace *)
   | [' ' '\t' '\n'] { token lexbuf }
   (* Comments *)
 | "#" { comment lexbuf }
 | "#*" { comment_multi_line lexbuf }

 (* Punctuation *)
```

```
(* Check for - comma, period  - usage *)
| '(' {LPAREN}      | ')' {RPAREN}
| '{' {LBRACE}      | '}' {RBRACE}
| ';' {SEMI}
| ',' {COMMA}
| '~' {DEREF}
| '@' {NODE_INIT}
| "[" {GET_CHILD_L}
| "]" {GET_CHILD_R}
| "=>" {NODE_ASSIGN}
| "<--" {LCHILD}
| "-->" {RCHILD}

(* Function Keywords *)
| "function"  {FUNCTION}
| "return"    {RETURN}

(* Conditionals and Loops Keywords *)
| "if"      {IF}
| "else"    {ELSE}
| "for"     {FOR}
| "while"   {WHILE}
| "continue"  {CONTINUE}
| "break"   {BREAK}

(* Types *)
| "int"     {INT}
| "float"     {FLOAT}
| "boolean"   {BOOL}
| "char"    {CHAR}
| "void"    {VOID}
| "string"    {STRING}


(* Arithmetic Operators *)
| '+' {PLUS}    | '-' { MINUS }
| '%' {MOD}     | '*' { TIMES }
| '/' { DIVIDE }

(* Assignment Operators *)
| "=" {ASSIGN}
(*| "--" {MINUS_MINUS}
```

```
 | "++" {PLUS_PLUS}*)

 (* Relational Operators *)
 | "==" {EQUALS}    | "!=" {NOT_EQUALS}
 | '>' {GREATER_THAN}  | ">=" {GREATER_THAN_EQUALS}
 | '<' {LESS_THAN}   | "<=" {LESS_THAN_EQUALS}

 (* Logical Operators *)
 | "&&" {AND}
 | "||" {OR}
 | "!" {NOT}

 (* Literals *)
 | "true" { BOOL_LIT(true)}
 | "false" { BOOL_LIT(false)}
 | identifier as lit {ID(lit)}
 | digit+ as lit {INT_LIT(int_of_string lit)}
 | ('-'? digit+ '.' digit* | '-'? '.' digit+) as lit {FLOAT_LIT(lit)}
 | '"' + [',''_''%''^''/''[']''(')''*''&''$''#''@''!''=''+''-'' ' '\''
'0'-'9''a'-'z''A'-'Z''\\']* + '"' as lit {
   let strip_quotes str = String.sub str 1 ((String.length str) - 2) in
STRING_LIT(strip_quotes lit)
   }
 | '\'' ([^'\''] as lit) '\'' {CHAR_LIT(lit)}
 | "\'\\n\'" {CHAR_LIT('\n')}
 | "\'\\t\'" {CHAR_LIT('\t')}
 | "\'\\\\\'" {CHAR_LIT('\\')}

 (* End of File *)
 | eof {EOF}

 (*Illegal characters*)
 | _ as c { raise (Failure("Illegal character: " ^ Char.escaped c))}

and comment = parse
 | '\n' {token lexbuf}
 | eof  {EOF}
 | _    {comment lexbuf}
and comment_multi_line = parse
 | "*#"  {token lexbuf}
 | _   {comment_multi_line lexbuf}
```

11.2 Parser

Parser.mly

```
/* Compile --
* ocamlyacc -v parser.mly)
*/

/*
*
* ARBOL Parser
*
*/

%{
 open Ast;;
%}

/* Punctuation Tokens */
%token LPAREN RPAREN LBRACE RBRACE
%token SEMI COMMA
%token LCHILD RCHILD DEREF NODE_INIT NODE_ASSIGN GET_CHILD_L GET_CHILD_R
%token EOF
/* LBRACK RBRACK */

/* Keywords */
%token FUNCTION RETURN
%token IF ELSE FOR WHILE CONTINUE BREAK
%token INT FLOAT BOOL CHAR VOID STRING
/* Added for Array inplementation */
// %token ARRAY NODE

/* Arithmetic, Assignment, Relational, and Logical Operators */
%token PLUS MINUS MOD TIMES DIVIDE
%token ASSIGN
%token EQUALS NOT_EQUALS GREATER_THAN GREATER_THAN_EQUALS LESS_THAN
LESS_THAN_EQUALS
%token AND OR NOT
```

```
/* Literals */
%token <bool> BOOL_LIT
%token <string> ID
%token <int> INT_LIT
%token <string> FLOAT_LIT
%token <char> CHAR_LIT
%token <string> STRING_LIT

%start entry
%type <Ast.program> entry
%type <Ast.program> program
%type <Ast.expr> expr
%type <Ast.vtype> vtype

/* Precedence and Associativity */
%nonassoc NOELSE
%nonassoc ELSE
%nonassoc LPAREN
%right ASSIGN NODE_ASSIGN
%left OR
%left AND
%left EQUALS NOT_EQUALS
%left GREATER_THAN GREATER_THAN_EQUALS LESS_THAN LESS_THAN_EQUALS
%left PLUS MINUS
%left TIMES DIVIDE MOD
%right NOT
%left GET_CHILD_L GET_CHILD_R

/* UNder NOT: %left DEREF */
%%

entry:
 program EOF         {$1}

program:
   /* nothing */       {[], []}
 | program vdecl SEMI      {($2 :: fst $1), snd $1}
 | program function_dec      {fst $1, ($2 :: snd $1)}

/*
```

```
* STATEMENTS
*/

statement_list:
 /* nothing */         {[]}
 | statement_list statement  {$2::$1}

statement:
   expr SEMI                 {Expr($1)}
 | if_else_statement          {$1}
 | iteration_statement          {$1}
 | BREAK SEMI             {Break}
 | CONTINUE SEMI            {Continue}
 | RETURN expr_optional SEMI     {Return($2)}
 | LBRACE statement_list RBRACE    {Block(List.rev $2)} /* Check */
 | node_declare_child SEMI       {$1}
 /*| assign_statement          {$1}*/
 | var_initialize SEMI           {Variable($1)}

if_else_statement:
   IF LPAREN expr RPAREN statement %prec NOELSE{ If($3, $5, Block([]))}
 | IF LPAREN expr RPAREN statement ELSE statement {If($3, $5, $7)}

iteration_statement:
   FOR LPAREN expr_optional SEMI expr_optional SEMI expr_optional RPAREN
statement { For($3, $5, $7, $9)}
 | WHILE LPAREN expr_optional RPAREN statement { While($3, $5)}

/*assign_statement:
     ID ASSIGN expr SEMI { Assign($1, $3) }
   | ID LBRACK expr RBRACK ASSIGN expr SEMI { SetElementAssign($1, $3, $6)} /*
Add this rule to AST */

/*array_assign_statement:
   var_type LBRACK expr RBRACK ASSIGN expr SEMI {ArrayAssign($1, $3, $6)}
 | var_type LBRACK RBRACK ASSIGN expr SEMI {ArrayAssign($1, Noexpr, $5)} */

/* Function Declaration: */
function_dec:
   FUNCTION vtype ID LPAREN params_optional RPAREN LBRACE statement_list
RBRACE {
```

```
    { rtype = $2;
      fname = $3;
      args_list = $5;
      body = List.rev $8;
    }}

/* Formal Args: */
params_optional:
   {[]} //nothing
 | params    {List.rev $1}

params:
   vtype ID        {[{
                     v_type = $1;
                     v_name = $2;
                     v_val = Noexpr;
                     node_val = false;
                    }]}
 | params COMMA vtype ID   {{
                     v_type = $3;
                     v_name = $4;
                     v_val = Noexpr;
                     node_val = false;
                    } :: $1}



/*
* EXPRESSIONS
*/
expr_optional:
 /* nothing */     {Noexpr}
 | expr            {$1} /* Just expr ? */

expr:
   literal              {$1}
 | ID                {Id($1)}
 | LPAREN expr RPAREN       {$2} /* new */
 | expr PLUS expr         {Binop($1, Plus, $3)}
 | expr MINUS expr          {Binop($1, Minus, $3)}
 | expr TIMES expr          {Binop($1, Times, $3)}
 | expr DIVIDE expr         {Binop($1, Divide, $3)}
```

```
 | expr MOD expr          {Binop($1, Mod, $3)}
 | expr EQUALS expr         {Binop($1, Equals, $3)}
 | expr NOT_EQUALS expr       {Binop($1, Not_Equals, $3)}
 | expr GREATER_THAN expr     {Binop($1, Greater, $3)}
 | expr GREATER_THAN_EQUALS expr  {Binop($1, Greater_Eq, $3)}
 | expr LESS_THAN expr        {Binop($1, Less, $3)}
 | expr LESS_THAN_EQUALS expr   {Binop($1, Less_Eq, $3)}
 | expr AND expr          {Binop($1, And, $3)}
 | expr OR expr           {Binop($1, Or, $3)}
 | NOT expr             {Unop(Not,$2)}
 | MINUS expr %prec NOT       {Unop(Neg, $2)} /* new */
 | node_ops                  {$1}
 | ID ASSIGN expr           {Assign($1, $3)} /* new */
 | ID LPAREN args_optional RPAREN {Call($1, $3)}


args_optional:
   /* nothing */ { [] }
 | args_list  { List.rev $1 }

args_list:
   expr                    { [$1] }
 | args_list COMMA expr { $3 :: $1 }

/*
* TREE SYNTAX
*/

node_declare_child:
   ID LCHILD expr         {Node_child($1, Set_left_child, $3)}
 | ID RCHILD expr         {Node_child($1, Set_right_child, $3)}

node_ops:
   DEREF ID            {Nodeop(Dref, $2)}
 | GET_CHILD_R ID      {Nodeop(Get_right_child, $2)}
 | GET_CHILD_L ID      {Nodeop(Get_left_child, $2)}
 | ID NODE_ASSIGN expr     {Node_assign($1, $3)}

/*
* VARIABLES
*/
```

```
vtype:
   literal_type      {$1}
 | vtype NODE_INIT {Node($1)}


literal_type:
   INT               {Int}
 | FLOAT             {Float}
 | BOOL              {Bool}
 | CHAR              {Char}
 | VOID              {Void}
 | STRING            {String}

var_initialize:
   vdecl             {$1}
 | vtype ID ASSIGN expr {
   { v_type = $1;
     v_name = $2;
     v_val = $4;
     node_val = false;
     }}
 | vtype ID NODE_ASSIGN expr { (* Creates node, deref, and node_assign to val
(i.e. 3) *)
   { v_type = $1;
     v_name = $2;
     v_val = $4;
     node_val = true;
    }}

vdecl:
   vtype ID      {
                 if ($1 == Void) then
                   raise Variable_of_void
                 else
                 {
                   v_type = $1;
                   v_name = $2;
                   v_val = Noexpr;
                   node_val = false;
                   }
               }
```

```
literal:
   INT_LIT           {Int_lit($1)}
 | FLOAT_LIT           {Float_lit($1)}
 | BOOL_LIT            {Bool_lit($1)}
 | CHAR_LIT            {Char_lit($1)}
 | STRING_LIT          {String_lit($1)} /* Added */
```

11.3 AST

ast.ml

```
* ARBOL Syntax Tree *)
(* Binary operators *)
type op = Plus   | Minus | Times | Divide | Mod
   | Equals | Not_Equals | Greater | Greater_Eq
   | Less   | Less_Eq | And | Or

(* Unary operators *)
type unop = Neg | Not

(* Post fix operators *)

(* Tree (Node) operators *)

type node_op = Get_left_child | Get_right_child | Dref

type node_child_op = Set_left_child | Set_right_child

type node_assign_op = Node_assign

(* Assignment Operators *)
type vtype =
 | Int
 | Float
 | Bool
```

```
 | Char
 | Void
 | String
 | Node of vtype

(* Expressions *)
type expr =
   Int_lit of int
 | Float_lit of string
 | Bool_lit of bool
 | Char_lit of char
 | String_lit of string
 | Id of string
 | Assign of string * expr
 | Unop of unop * expr
 | Binop of expr * op * expr
 | Call of string * expr list
 | Nodeop of node_op * string
 | Node_assign of string * expr
 | Noexpr
 (*
 | Not of expr
 | Array of expr list;
 | ArrayAccess of string * expr
 | Value of expr * expr
 *)

(* Variable Declaration *)

type vdecl = {
 v_type: vtype;
 v_name: string;
 v_val: expr;
 node_val: bool;
}

(* Statements *)
type stmt =
   Block of stmt list
 | Expr of expr
 | Return of expr
```

```
 | If of expr * stmt * stmt
 | For of expr * expr * expr * stmt
 | While of expr * stmt
 | Continue
 | Break
 | Node_child of string * node_child_op * expr
 (* | ArrayAssign of string * expr * expr *)
 | Variable of vdecl

(* Function Declaration *)
type fdecl = {
 rtype: vtype;
 fname: string;
 args_list: vdecl list; (* formal arguments*)
 body: stmt list;
}

(* type program = vdecl * func_decl *)
type program = vdecl list * fdecl list

(* Need this? *)
exception Variable_of_void

(*
* Prints out Abstract Syntax Tree
*)

let string_of_op = function
   Plus    -> "+"
 | Minus   -> "-"
 | Times   -> "*"
 | Divide -> "/"
 | Mod     -> "%"
 | Equals   -> "=="
 | Not_Equals -> "!="
 | Less -> "<"
 | Less_Eq -> "<="
 | Greater -> ">"
 | Greater_Eq -> ">="
 | And -> "&&"
 | Or -> "||"
```

```
let string_of_unop = function
 | Neg -> "-"
 | Not -> "!"

let string_of_node_op = function
 | Get_left_child -> "<^"
 | Get_right_child -> "^>"
 | Dref -> "~"

let string_of_node_child_op = function
 | Set_right_child -> "-->"
 | Set_left_child -> "<--"

let string_of_node_assign_op = "=>"

let rec string_of_expr = function
     Int_lit(l) -> string_of_int l
   | Float_lit(l) -> l
   | Bool_lit(true) -> "true"
   | Bool_lit(false) -> "false"
   | Char_lit(l) -> String.make 1 l
   | String_lit(l) -> l
   | Id(s) -> s
   | Assign(v, e) -> v ^ " = " ^ string_of_expr e
   | Unop(o, e) -> string_of_unop o ^ string_of_expr e
   | Binop(e1, o, e2) ->
       string_of_expr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_expr e2
   | Call(f, el) -> f ^ "(" ^ String.concat ", " (List.map string_of_expr el)
^ ")"
   | Nodeop(o, n1) -> string_of_node_op o ^ n1
   | Node_assign(n1, e) -> n1 ^ " " ^ string_of_node_assign_op ^ " " ^
string_of_expr e
   | Noexpr -> ""

let rec string_of_vtype = function
 | Int -> "int"
 | Bool -> "bool"
 | Float -> "float"
 | Char -> "char"
 | String -> "string"
```

```
 | Void -> "void"
 | Node(x) -> string_of_vtype x ^ " @"

let string_of_vdecl var =
 match var.v_val with
 | Noexpr -> string_of_vtype var.v_type ^ " " ^ var.v_name
 | _ ->
 match var.node_val with
 | true -> string_of_vtype var.v_type ^ " " ^ var.v_name ^ " " ^
string_of_node_assign_op ^ " " ^ string_of_expr var.v_val
 | false -> string_of_vtype var.v_type ^ " " ^ var.v_name ^ " = " ^
string_of_expr var.v_val

let string_of_fdecl var = string_of_vtype var.v_type ^ " " ^ var.v_name

let rec string_of_stmt = function
   Block(stmts) ->
     "{\n" ^ String.concat "" (List.map string_of_stmt stmts) ^ "}\n"
 | Expr(expr) -> string_of_expr expr ^ ";\n";
 | Return(expr) -> "return " ^ string_of_expr expr ^ ";\n";
 | If(e, s, Block([])) -> "if (" ^ string_of_expr e ^ ")\n" ^ string_of_stmt s
 | If(e, s1, s2) ->  "if (" ^ string_of_expr e ^ ")\n" ^
   string_of_stmt s1 ^ "else\n" ^ string_of_stmt s2
 | For(e1, e2, e3, s) ->
   "for (" ^ string_of_expr e1  ^ " ; " ^ string_of_expr e2 ^ " ; " ^
   string_of_expr e3  ^ ") " ^ string_of_stmt s
 | While(e, s) -> "while (" ^ string_of_expr e ^ ") " ^ string_of_stmt s
 | Continue -> "continue" ^ ";\n"
 | Break -> "break" ^ ";\n"
 | Node_child(n1, o, e) -> n1 ^ " " ^ string_of_node_child_op o ^ " " ^
string_of_expr e ^ ";\n"
 | Variable(v) -> string_of_vdecl v ^ ";\n"

let string_of_fdecl fdecl =
 "function " ^ string_of_vtype fdecl.rtype ^ " " ^
 fdecl.fname ^ "(" ^
 String.concat ", " (List.map string_of_fdecl fdecl.args_list) ^
 ")\n{\n" ^
 String.concat "" (List.map string_of_stmt fdecl.body) ^
 "}\n"
```

```
let string_of_program (vlist, flist) =
 "START\n" ^
   (String.concat "\n" (List.map string_of_vdecl vlist)) ^ "\n\n" ^
   (String.concat "\n\n" (List.map string_of_fdecl (List.rev flist) )) ^
 "\nEND\n"
```

11.4 Semantic Checker

Semantic.ml

```
(* Semantic checking for the MicroC compiler *)

open Ast
open Sast

module StringMap = Map.Make(String)

type environment = {
 vars: svdecl list;
 parent: environment option;
 in_loop: bool;
}

(* Builtin functions *)
let builtins = [
 {
   fname = "print";
   rtype = Void;
   args_list = [{
     v_type = String;
     v_name = "str";
     v_val = Noexpr;
     node_val = false;
   }];
   body = [];
 };
```

```
 {
   fname = "print_int";
   rtype = Void;
   args_list = [{
     v_type = Int;
     v_name = "int";
     v_val = Noexpr;
     node_val = false;
   }];
   body = [];
 };
 {
   fname = "print_float";
   rtype = Void;
   args_list = [{
     v_type = Float;
     v_name = "float";
     v_val = Noexpr;
     node_val = false;
   }];
   body = [];
 };
]

let built_in_decls =
 let add_bind map f = StringMap.add f.fname f map
 in List.fold_left add_bind StringMap.empty builtins

let is_node n = match n with
 | Node(_) -> true
 | _ -> false

let check_types s (t1, t2) e =
match t2 with
| Void -> t1
| _ ->
match t1 = t2 with
| true -> t1
| false -> raise (Failure ("type error in " ^ s
 ^ ": expected type " ^ string_of_vtype t1
```

```
 ^ " but received type " ^ string_of_vtype t2 ^ " in expr " ^ string_of_sexpr
e))

(* Verify a list of bindings has no void types or duplicate names *)
let check_binds (kind : string) (types : vdecl list) =
 let type_to_name = List.map (fun vdecl -> (vdecl.v_type, vdecl.v_name)) types
 in
 List.iter (function
   (Void, name) -> raise (Failure ("illegal void " ^ kind ^ " " ^ name))
   | _ -> ()) type_to_name;
 let rec dups = function
     [] -> ()
   | ((_,n1) :: (_,n2) :: _) when n1 = n2 ->
       raise (Failure ("duplicate " ^ kind ^ " " ^ n1))
   | _ :: t -> dups t
 in dups (List.sort (fun (_,a) (_,b) -> compare a b) type_to_name)

(* Add function name to symbol table *)
let add_func map fd =
 let built_in_err = "function " ^ fd.fname ^ " may not be defined"
 and dup_err = "duplicate function " ^ fd.fname
 and make_err er = raise (Failure er)
 and n = fd.fname (* Name of the function *)
 in match fd with (* No duplicate functions or redefinitions of built-ins *)
       _ when StringMap.mem n built_in_decls -> make_err built_in_err
     | _ when StringMap.mem n map -> make_err dup_err
     | _ ->  StringMap.add n fd map

(* Look for function or var *)
let find_func funcs s =
 try StringMap.find s funcs
 with Not_found -> raise (Failure ("could not find function " ^ s))

(* Check expressions *)

let check_function func_decls global_env func =
 (* Make sure no formals or locals are void or duplicates *)
 check_binds "formal" func.args_list;
 let formals = List.map (fun v -> {
   sv_type = v.v_type;
   sv_name = v.v_name;
```

```
   sv_val = SNoexpr;
   sv_node_val = false;
 }) func.args_list in

 let local_env = { vars = formals; parent = Some(global_env); in_loop = false
} in

 let rec var_exists vars name = match vars with
 | var :: rest -> if var.sv_name = name then Some(var) else var_exists rest
name
 | [] -> None in

 let rec find_var env_option name = match env_option with
   | None -> raise (Failure ("variable " ^ name ^ " is not in scope"))
   | Some(env) ->
  match var_exists env.vars name with
   | Some(var) -> var
   | None -> find_var env.parent name in

 let rec check_expr (e: Ast.expr) env = match e with
     Int_lit(l) -> SInt_lit(l), Int
   | Float_lit(l) -> SFloat_lit(l), Float
   | Bool_lit(l) -> SBool_lit(l), Bool
   | Char_lit(l) -> SChar_lit(l), Char
   | String_lit(l) -> SString_lit(l), String
   | Unop(op, e1) as e ->
     let (e1', t) = check_expr e1 env in
     let ty = match op with
       Neg when t = Int || t = Float -> t
     | Not when t = Bool -> Bool
     | _ -> raise (Failure ("illegal unary operator " ^
                            string_of_unop op ^ string_of_vtype t ^
                            " in " ^ string_of_expr e))
     in (SUnop(op, e1', ty), ty)
   | Binop(e1, op, e2) as e ->
           let (e1', t1) = check_expr e1 env
           and (e2', t2) = check_expr e2 env in
           (* All binary operators require operands of the same type *)
           let same = t1 = t2 in
           (* Determine expression type based on operator and operand types *)
           let ty = match op with
```

```
          | Plus | Minus | Times | Divide | Mod when same && t1 = Int    ->
Int
          | Plus | Minus | Times | Divide | Mod when same && t1 = Float ->
Float
          | Plus | Minus | Times | Divide | Mod when same && t1 = Char ->
Char
          | Not_Equals | Equals                  when same               ->
Bool
          | Greater | Greater_Eq | Less  | Less_Eq
                    when same && (t1 = Int || t1 = Float) -> Bool
          | And | Or when same && t1 = Bool -> Bool
          | Not_Equals | Equals when (is_node t1 && t2 = Void) || (is_node t2
&& t1 = Void) -> Bool
          | _ -> raise (
        Failure ("illegal binary operator " ^
                      string_of_vtype t1 ^ " " ^ string_of_op op ^ " " ^
                      string_of_vtype t2 ^ " in " ^ string_of_expr e))
          in (SBinop(e1', op, e2', t1, t2, ty), ty)
   | Call(fname, args) as call ->
      let fd = find_func func_decls fname in
      let param_length = List.length fd.args_list in
      if List.length args != param_length then
            raise (Failure ("expecting " ^ string_of_int param_length ^
                            " arguments in " ^ string_of_expr call))
      else let check_call vdecl e =
             let (e', et) = check_expr e env in
             let _ = check_types "function call" (vdecl.v_type, et) e'
             in e'
           in
           let args' = List.map2 check_call fd.args_list args
           in SCall(fname, args', fd.rtype), fd.rtype
   | Id(s) -> let sv = find_var (Some env) s in SId(s, sv.sv_type), sv.sv_type
   | Assign(s, e) -> let sv = find_var (Some env) s in let (e', et) =
check_expr e env in
      let t = check_types "assignment" (sv.sv_type, et) e' in SAssign(s, e',
t), t
   | Nodeop(node_op, s) -> let sv = find_var (Some env) s in (match sv.sv_type
with
       | Node(t) -> (match node_op with
           | Get_left_child | Get_right_child -> SNodeop(node_op, s,
sv.sv_type), sv.sv_type
```

```
          | Dref -> SNodeop(node_op, s, t), t
        )
      | _ -> raise (Failure ("expected Node type for var " ^ s)))
  | Node_assign(s, e) -> let sv = find_var (Some env) s in let (e', et) =
check_expr e env in
      (match sv.sv_type with
        | Node(t1) -> let t = check_types "node assign" (t1, et) e' in
SNode_assign(s, e', t), t
        | _ -> raise (Failure ("expected Node type for var " ^ s
          ^ " for node assignment in expression " ^ string_of_sexpr e'))
      )
  | Noexpr -> SNoexpr, Void in
 let to_svdecl v env = let (e', et) = check_expr v.v_val env in match
v.node_val with
  | true -> (match v.v_type with
    | Node(t1) -> let _ = check_types ("node assignment" ^ v.v_name) (t1, et)
e' in
      {
        sv_type = v.v_type;
        sv_name = v.v_name;
        sv_val = e';
        sv_node_val = true
      }
    | _ -> raise (Failure ("internal parsing error"))
    )
  | false -> let t = check_types ("variable declaration " ^ v.v_name)
(v.v_type, et) e' in {
      sv_type = t;
      sv_name = v.v_name;
      sv_val = e';
      sv_node_val = false
    } in

 let add_id env v =
  match v.v_type with
  | Void -> raise (Failure ("illegal void type for variable " ^ v.v_name))
  | _ ->
  match var_exists env.vars v.v_name with
  | Some(_) -> raise (Failure ("variable " ^ v.v_name ^ " declared previously
in this scope"))
  | None ->
```

```
   match v.node_val with
   | false -> { env with vars = to_svdecl v env :: env.vars }
   | true ->
   match v.v_type with
   | Node(_) -> { env with vars = to_svdecl v env :: env.vars }
   | _ -> raise (Failure ("illegal declaration for variable " ^ v.v_name
       ^ ": expected node type with node assignment operator")) in

 let check_bool_expr e env =
   let (e', t') = check_expr e env
     and err = "expected Boolean expression in " ^ string_of_expr e
   in if t' != Bool then raise (Failure err) else e' in

 let add_var s curr_env = match s with
   | Variable(v) -> add_id curr_env v
   | _ -> curr_env in

 (* Return a semantically-checked statement i.e. containing sexprs *)
 let rec check_stmt (s: Ast.stmt) env = match s with
   | If(p, b1, b2) -> SIf(check_bool_expr p env, check_stmt b1 env, check_stmt
b2 env)
   | For(e1, e2, e3, st) ->
     let new_env = { vars = []; parent = Some(env); in_loop = true } in
     SFor(fst (check_expr e1 new_env), check_bool_expr e2 new_env, fst
(check_expr e3 new_env), check_stmt st new_env)
   | While(p, s) ->
     let new_env = { vars = []; parent = Some(env); in_loop = true } in
     SWhile(check_bool_expr p new_env, check_stmt s new_env)
   | Return e -> let (e', t) = check_expr e env in
     if t = func.rtype then SReturn (e', t)
     else raise (
       Failure ("return gives " ^ string_of_vtype t ^ " expected " ^
         string_of_vtype func.rtype ^ " in " ^ string_of_expr e))
   | Continue -> (match env.in_loop with
     | true -> SContinue
     | false -> raise (Failure "continue statement must occur within loop"))
   | Break -> (match env.in_loop with
     | true -> SBreak
     | false -> raise (Failure "break statement must occur within loop"))
   | Block sl ->
```

```
      let new_env = { vars = []; parent = Some(env); in_loop = env.in_loop }
in
      let rec check_stmt_list s curr_env : Sast.sstmt list = (match s with
        | [Return _ as s] -> [check_stmt s curr_env]
        | Break as s :: _ -> [check_stmt s curr_env]
        | Continue as s :: _ -> [check_stmt s curr_env]
        | Return _ :: _   -> raise (Failure "nothing may follow a return")
        | Block _ :: _  -> raise (Failure "cannot nest block inside a block")
        | s :: ss         -> (check_stmt s curr_env) :: check_stmt_list ss
(add_var s curr_env)
        | []              -> [])
      in SBlock(check_stmt_list sl new_env)
   | Expr e -> let (sexpr, _) = check_expr e env in SExpr(sexpr)
   | Variable(v) -> (match v.v_name with
     | "null" -> raise (Failure ("not allowed to create variable named null"))
     | _ -> SVariable(to_svdecl v env))
   | Node_child(n, o, e) -> let (e', et) = check_expr e env in (match et with
     | Node(_) -> let sv = (find_var (Some env) n) in
      let t = check_types "node child operation" (sv.sv_type, et) e' in
SNode_child(n, o, e', t)
     | _ -> raise (Failure ("expected node type for expression " ^
string_of_sexpr e')))

 in (* body of check_function *)
 { srtype = func.rtype;
   sfname = func.fname;
   sargs_list  = List.map (fun x -> (x.v_type, x.v_name)) func.args_list;
   sbody = match (check_stmt (Block func.body) local_env) with
     SBlock(sl) -> sl
     | _ -> raise (Failure ("internal error: block didn't become a block?"))
 }


(* Semantic checking of the AST. Returns an SAST if successful,
  throws an exception if something is wrong.

  Check each global variable, then check each function *)

let check (globals, functions) =

 (**** Check global variables ****)
```

```
 check_binds "global" globals;

 let global_svdecls = List.map (fun vdecl -> {
   sv_type = vdecl.v_type;
   sv_name = vdecl.v_name;
   sv_val = SNoexpr;
   sv_node_val = false;
 }) globals
 in
  let null_var = {
   sv_type = Void;
   sv_name = "null";
   sv_val = SNoexpr;
   sv_node_val = false;
 }
 in
  let global_env = { vars = null_var::global_svdecls; parent = None; in_loop =
false }
 in
 (**** Check functions ****)

 (* Collect all function names into one symbol table *)
 let function_decls = List.fold_left add_func built_in_decls functions
 in

 let _ = find_func function_decls "main"
 in
 (global_svdecls, List.map (check_function function_decls global_env )
functions)
```

11.5 Code Generator

Codegen.ml

```
(* Code generation: translate takes a semantically checked AST and
produces LLVM IR
```

```
/*
LLVM tutorial: Make sure to read the OCaml version of the tutorial

http://llvm.org/docs/tutorial/index.html

Detailed documentation on the OCaml LLVM library:

http://llvm.moe/
http://llvm.moe/ocaml/

*)

module L = Llvm
module A = Ast
open Sast

module StringMap = Map.Make(String)

type l_environment = {
 vars: L.llvalue StringMap.t;
 parent: l_environment option;
}

type l_types = {
 mutable nodes: L.lltype StringMap.t;
}

let get_opt o = match o with
 | Some(x) -> x
 | None -> raise (Failure ("failed to extract value (should have been caught
by semant)"))

(* translate : Sast.program -> Llvm.module *)
let translate (globals, functions) =
 let context    = L.global_context () in
  (* Create the LLVM compilation module into which
    we will generate code *)
 let the_module = L.create_module context "Arbol" in
  let mytypes = { nodes = StringMap.empty } in
  (* Get types from the context *)
 let i32_t      = L.i32_type    context
```

```
 and i8_t       = L.i8_type     context
 and i1_t       = L.i1_type     context
 and float_t    = L.double_type context
 and str_t      = L.pointer_type (L.i8_type context)
 and void_t     = L.void_type   context in

 (* Return the LLVM type for a MicroC type *)
 let rec ltype_of_typ = function
     A.Int   -> i32_t
   | A.Bool  -> i1_t
   | A.Float -> float_t
   | A.Char -> i8_t
   | A.String -> str_t
   | A.Void  -> void_t
   | A.Node(t) ->
     try StringMap.find (A.string_of_vtype t) mytypes.nodes
     with Not_found ->
       let ntype = L.named_struct_type context ("node_" ^ A.string_of_vtype t)
in
       let ptype = L.pointer_type (ltype_of_typ t) in
       let ctype = L.pointer_type ntype in
       let _ = L.struct_set_body ntype [|ptype; ctype; ctype|] true in
       mytypes.nodes <- StringMap.add (A.string_of_vtype t) ntype
mytypes.nodes; ntype
 in
  let default_vals t = match t with
   | A.Float -> L.const_float (ltype_of_typ t) 0.0
   | A.Int -> L.const_int (ltype_of_typ t) 0
   | A.Bool -> L.const_int (ltype_of_typ t) 0
   | A.Char -> L.const_int (ltype_of_typ t) 0
   | A.String -> L.const_pointer_null (ltype_of_typ t)
   | A.Node(t) -> let pnull = L.const_pointer_null (ltype_of_typ t) in
       let cnull = L.const_pointer_null (ltype_of_typ t) in
       L.const_struct context [|pnull; cnull; cnull|]
   | A.Void -> raise (Failure "illegal void type (should have been checked by
semant)") in

 let fill_null_node node node_t t builder =
   let pnull = L.build_malloc (ltype_of_typ t) "default_val" builder in
   let cnull = L.const_pointer_null (L.pointer_type (ltype_of_typ node_t)) in
   let data_ptr = L.build_struct_gep node 0 "result" builder in
```

```
   let lchild_ptr = L.build_struct_gep node 1 "lchild_ptr" builder in
   let rchild_ptr = L.build_struct_gep node 2 "rchild_ptr" builder in
   let _ = L.build_store (default_vals t) pnull builder in
   let _ = L.build_store pnull data_ptr builder in
   let _ = L.build_store cnull lchild_ptr builder in
   let _ = L.build_store cnull rchild_ptr builder in builder in
  (* Create a map of global variables after creating each *)
 (* Takes a binding (type and name) and creates global constants initialized
to 0 *)
 let global_vars : L.llvalue StringMap.t =
   let global_var m sv =
     let init = default_vals sv.sv_type
     in StringMap.add sv.sv_name (L.define_global sv.sv_name init the_module)
m in
   List.fold_left global_var StringMap.empty globals in
  let global_env = { vars = global_vars; parent = None } in

 let printf_t : L.lltype =
     L.var_arg_function_type i32_t [| L.pointer_type i8_t |] in
 let printf_func : L.llvalue =
     L.declare_function "print" printf_t the_module in

 (* Adding print_int implementation from semantic *)
 let printf_int : L.lltype =
       L.var_arg_function_type i32_t [| i32_t |] in
 let printf_int_func : L.llvalue =
       L.declare_function "print_int" printf_int the_module in

 (* print_float implementation *)
 let printf_float : L.lltype =
   L.var_arg_function_type float_t [| float_t |] in
 let printf_float_func : L.llvalue =
   L.declare_function "print_float" printf_float the_module in

 (* built-in function 'pre-order' *)
 (* let preorder_node : L.lltype =
   L.var_arg_function_type node [| node |] in
 let preorder_node_func : L.llvalue =
   L.declare_function "preorder" preorder_node the_module in *)

 let unwrap_sargs = function (t,_) -> ltype_of_typ t in
```

```
 (* Define each function (arguments and return type) so we can
    call it even before we've created its body *)
 let function_decls : (L.llvalue * sfdecl) StringMap.t =
 let function_decl m fdecl =
   let name = fdecl.sfname
   and formal_types = Array.of_list (List.map unwrap_sargs fdecl.sargs_list)
   in let ftype = L.function_type (ltype_of_typ fdecl.srtype) 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 global_env fdecl =
   let (the_function, _) = StringMap.find fdecl.sfname function_decls in
   let builder = L.builder_at_end context (L.entry_block the_function) in

 (* Return the value for a variable or formal argument, starting from lowest
scope *)
 let rec lookup n env = try StringMap.find n env.vars
   with Not_found -> match env.parent with
                   | Some(p) -> lookup n p
                   | None -> raise (Failure ("var " ^ n ^ " not found
(semantic analysis failed!)")) in

 (*
   Add variable to env
 *)
 let add_formal m (t, n) p =
   L.set_value_name n p;
   let local = L.build_alloca (ltype_of_typ t) n builder in
   ignore (L.build_store p local builder);
   StringMap.add n local m in

 let add_local m sv builder = match sv.sv_type with
   | A.Node(t) as node_t -> let node_var = L.build_alloca (ltype_of_typ
sv.sv_type) sv.sv_name builder in
     let _ = fill_null_node node_var node_t t builder in
     StringMap.add sv.sv_name node_var m
   | _ -> let local_var = L.build_alloca (ltype_of_typ sv.sv_type) sv.sv_name
builder in
     StringMap.add sv.sv_name local_var m in
```

```
 (* Add formals *)
 let formals = List.fold_left2 add_formal StringMap.empty fdecl.sargs_list
     (Array.to_list (L.params the_function)) in
 let local_env = { vars = formals; parent = Some(global_env) } in

 (* Construct code for an expression; return its value *)
 let rec expr env builder (e : sexpr) = match e with
     SInt_lit i  -> L.const_int i32_t i, None
   | SChar_lit c  -> L.const_int i8_t (Char.code c), None
   | SBool_lit b  -> L.const_int i1_t (if b then 1 else 0), None
   | SString_lit s -> L.build_global_stringptr s "string" builder, None
   | SFloat_lit l -> L.const_float_of_string float_t l, None
   | SUnop(op, e, t) ->
     let (e', _) = expr env builder e in
       (match op with
         A.Neg when t = A.Float -> L.build_fneg
       | A.Neg                  -> L.build_neg
       | A.Not                  -> L.build_not) e' "tmp" builder, None
   | SBinop (_, op, e2, A.Void, _, _) ->
     let (_, ptr) = expr env builder e2 in
     (match op with
       | A.Equals -> L.build_is_null
       | A.Not_Equals -> L.build_is_not_null
       | _ -> raise (Failure "internal error: semant should have rejected
invalid op on void")
     ) (get_opt ptr) "tmp" builder, None
   | SBinop (e1, op, _, _, A.Void, _) ->
     let (_, ptr) = expr env builder e1 in
     (match op with
       | A.Equals -> L.build_is_null
       | A.Not_Equals -> L.build_is_not_null
       | _ -> raise (Failure "internal error: semant should have rejected
invalid op on void")
     ) (get_opt ptr) "tmp" builder, None
   | SBinop (e1, op, e2, A.Float, _, _) ->
   let (e1', _) = expr env builder e1
   and (e2', _) = expr env builder e2 in
     (match op with
       A.Plus     -> L.build_fadd
     | A.Minus     -> L.build_fsub
```

```
    | A.Times    -> L.build_fmul
    | A.Divide     -> L.build_fdiv
    | A.Mod     -> L.build_frem
    | A.Equals   -> L.build_fcmp L.Fcmp.Oeq
    | A.Not_Equals     -> L.build_fcmp L.Fcmp.One
    | A.Less    -> L.build_fcmp L.Fcmp.Olt
    | A.Less_Eq     -> L.build_fcmp L.Fcmp.Ole
    | A.Greater -> L.build_fcmp L.Fcmp.Ogt
    | A.Greater_Eq    -> L.build_fcmp L.Fcmp.Oge
    | A.And | A.Or ->
        raise (Failure "internal error: semant should have rejected and/or on
float")
    ) e1' e2' "tmp" builder, None
  | SBinop (e1, op, e2, _, _, _) ->
  let (e1', _) = expr env builder e1
  and (e2', _) = expr env builder e2 in
    (match op with
      A.Plus     -> L.build_add
    | A.Minus     -> L.build_sub
    | A.Times    -> L.build_mul
    | A.Divide     -> L.build_sdiv
    | A.Mod     -> L.build_srem
    | A.And     -> L.build_and
    | A.Or      -> L.build_or
    | A.Equals   -> L.build_icmp L.Icmp.Eq
    | A.Not_Equals     -> L.build_icmp L.Icmp.Ne
    | A.Less    -> L.build_icmp L.Icmp.Slt
    | A.Less_Eq     -> L.build_icmp L.Icmp.Sle
    | A.Greater -> L.build_icmp L.Icmp.Sgt
    | A.Greater_Eq     -> L.build_icmp L.Icmp.Sge
    ) e1' e2' "tmp" builder, None
  | SCall ("print", [e], _) ->
      let (v, _) = expr env builder e in L.build_call printf_func [| v |]
        "print" builder, None
  | SCall ("print_int", [e], _) ->
      let (v, _) = expr env builder e in L.build_call printf_int_func [| v |]
        "print_int" builder, None
  | SCall ("print_float", [e], _) ->
      let (v,_) = expr env builder e in L.build_call printf_float_func [| v
|]
      "print_float" builder, None
```

```
   (* | SCall ("preorder", [e], _) ->
       let (v,_) = expr env builder e in L.build_call preorder_node_func [| v
|]
       "preorder" builder, None *)
   | SCall (f, args, _) ->
       let (fdef, fdecl) = StringMap.find f function_decls in
       let llargs = List.rev (List.map (fun arg -> fst (expr env builder arg))
(List.rev args)) in
       let result = (match fdecl.srtype with
                       A.Void -> ""
                     | _ -> f ^ "_result") in
       L.build_call fdef (Array.of_list llargs) result builder, None
   | SAssign (s, e, _) -> let (e', _) = expr env builder e in
       ignore(L.build_store e' (lookup s env) builder); e', None
   | SId (s, _)       -> let ptr = lookup s env in L.build_load ptr s builder,
Some(ptr)
   | SNode_assign(s, e, t) -> let (e', _) = expr env builder e in
     let ptr = L.build_malloc (ltype_of_typ t) (s ^ "_val") builder in
     let _ = L.build_store e' ptr builder in
     let node = lookup s env in
     let ptr_ptr = L.build_struct_gep node 0 "result" builder in
     let _ = L.build_store ptr ptr_ptr builder in e', Some(ptr)
   | SNodeop(nodeop, s, _) -> (match nodeop with
     | A.Get_left_child ->
       let node = lookup s env in
       let ptr_ptr = L.build_struct_gep node 1 "result" builder in
       let ptr = L.build_load ptr_ptr (s ^ "_ptr") builder in
       L.build_load ptr (s ^ "_lchild") builder, Some(ptr)
     | A.Get_right_child ->
       let node = lookup s env in
       let ptr_ptr = L.build_struct_gep node 2 "result" builder in
       let ptr = L.build_load ptr_ptr (s ^ "_ptr") builder in
       L.build_load ptr (s ^ "_rchild") builder, Some(ptr)
     | A.Dref ->
       let node = lookup s env in
       let ptr_ptr = L.build_struct_gep node 0 "result" builder in
       let ptr = L.build_load ptr_ptr (s ^ "_ptr") builder in
       L.build_load ptr (s ^ "_val") builder, Some(ptr))
   | SNoexpr     -> L.const_int i32_t 0, None
 in
  (* LLVM insists each basic block end with exactly one "terminator"
```

```
    instruction that transfers control.  This function runs "instr builder"
    if the current block does not already have a terminator.  Used,
    e.g., to handle the "fall off the end of the function" case. *)
let add_terminal builder instr =
  match L.block_terminator (L.insertion_block builder) with
  | Some _ -> ()
  | None -> ignore (instr builder) in

(* First allocate all local variables *)
let create_env builder m s =
  match s with
  | SVariable(sv) -> add_local m sv builder
  | _ -> m in

(* Remove SVariables *)
let parse_vars s = match s with
  | SVariable(sv) -> (match sv.sv_node_val with
    | true -> (
      match sv.sv_type with
      | A.Node(t) -> SExpr (SNode_assign(sv.sv_name, sv.sv_val, t))
      | _ -> raise (Failure ("internal error: node assignment check failed"))
      )
    | false -> (
      match sv.sv_val with
      | SNoexpr -> SExpr(SNoexpr)
      | expr -> SExpr(SAssign(sv.sv_name, expr, sv.sv_type))
    )
  )
  | stmt -> stmt in
(* Build the code for the given statement; return the builder for
    the statement's successor (i.e., the next instruction will be built
    after the one generated by this call) *)
 let rec stmt merge_block next_block env builder = function
  | SBlock sl -> let parsed_sl = List.map parse_vars sl in
    let new_env = {
      vars = List.fold_left (create_env builder) StringMap.empty sl;
      parent = Some(env)
    } in List.fold_left (stmt merge_block next_block new_env) builder
parsed_sl
  | SExpr (e) -> ignore(expr env builder e); builder
  | SReturn (e, _) -> ignore(match fdecl.srtype with
```

```
      (* Special "return nothing" instr *)
      A.Void -> L.build_ret_void builder
      (* Build return statement *)
      | _ -> let (e', _) = expr env builder e in L.build_ret e' builder);
    builder
  | SIf (predicate, then_stmt, else_stmt) ->
    let (bool_val, _) = expr env builder predicate in
    let merge_bb = L.append_block context "merge" the_function in
    let build_br_merge = L.build_br merge_bb in (* partial function *)

    let then_bb = L.append_block context "then" the_function in
    add_terminal (stmt merge_block next_block env (L.builder_at_end context
then_bb) then_stmt)
    build_br_merge;

    let else_bb = L.append_block context "else" the_function in
    add_terminal (stmt merge_block next_block env (L.builder_at_end context
else_bb) else_stmt)
    build_br_merge;

    ignore(L.build_cond_br bool_val then_bb else_bb builder);
    L.builder_at_end context merge_bb

  | SWhile (predicate, body) ->
    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

    let pred_builder = L.builder_at_end context pred_bb in
    let (bool_val, _) = expr env pred_builder predicate in
    let merge_bb = L.append_block context "merge" the_function in

    add_terminal (stmt (Some merge_bb) (Some pred_bb) env (L.builder_at_end
context body_bb) body)
      (L.build_br pred_bb);

    ignore(L.build_cond_br bool_val body_bb merge_bb pred_builder);
    L.builder_at_end context merge_bb

  | SFor (e1, predicate, e3, body) ->
```

```
    ignore(expr env builder e1);

    let pred_bb = L.append_block context "for" the_function in
    ignore(L.build_br pred_bb builder);

    let body_bb = L.append_block context "for_body" the_function in

    let next_bb = L.append_block context "for_next" the_function in

    let pred_builder = L.builder_at_end context pred_bb in
    let (bool_val, _) = expr env pred_builder predicate in
    let merge_bb = L.append_block context "merge" the_function in

    let next_builder = L.builder_at_end context next_bb in
    ignore(expr env next_builder e3);

    add_terminal (stmt (Some merge_bb) (Some next_bb) env (L.builder_at_end
context body_bb) body)
      (L.build_br next_bb);

    add_terminal next_builder (L.build_br pred_bb);
    ignore(L.build_cond_br bool_val body_bb merge_bb pred_builder);
    L.builder_at_end context merge_bb
  (* | SFor (e1, e2, e3, body) -> stmt merge_block next_block env builder
    ( SBlock [SExpr e1 ; SWhile (e2, SBlock [body ; SExpr e3]) ] ) *)
  | SNode_child(s, nodeop, e, _) -> let child_ptr = (match nodeop with
    | A.Set_left_child -> L.build_struct_gep (lookup s env) 1 (s ^ "_lchild")
builder
    | A.Set_right_child -> L.build_struct_gep (lookup s env) 2 (s ^
"_rchild") builder) in
    let (_, target) = expr env builder e in (match target with
      | Some(ptr) -> ignore(L.build_store ptr child_ptr builder); builder
      | None -> raise (Failure "no node detected")
      )
  | SBreak -> (match merge_block with
    | Some(bb) -> ignore(L.build_br bb builder); builder
    | None -> raise (Failure "not in loop--internal error, should have been
checked in semant"))
  | SContinue -> (match next_block with
    | Some(bb) -> ignore(L.build_br bb builder); builder
```

```
    | None -> raise (Failure "not in loop--internal error, should have been
checked in semant"))
  | SVariable(_) -> raise (Failure "internal codegen error--variable not
parsed out")
 in

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

 let builder = stmt None None local_env builder (SBlock fdecl.sbody) in
 (* Add a return if the last block falls off the end *)
 add_terminal builder (match fdecl.srtype with
    A.Void -> L.build_ret_void
  | t -> L.build_ret (default_vals t)
 )
 in

 List.iter (build_function_body global_env) functions;
 the_module
```

11.6 Tests

We have copy-pasted each test in our test suite from the tests folder, along with their expected output below.

test-add1.arbol

```
function int add(int x, int y) {
  return x + y;
}

function void main(){
  print_int((add(2, 2)));
}
```

4

test-add2.arbol

```
function int add(int x, int y) {
  return x + y;
}

function void main(){
  print_int((add(-1, -1)));
}
```

-2

test-arith1.arbol

```
function void main(){
  print_int(2 + 2);
}
```

4

test-arith2.arbol

```
function void main(){
  print_int(2 + 2 * 10 + 2);
}
```

24

test-arith3.arbol

```
function void main(){
  int x;
  x = 10;
  int y;
  y = 5;
  print_int(x + y);
}
```

15

test-break1.arbol

```
function void main(){
```

```
        int i;
        for(i = 0; i < 5; i = i + 1) {
            if (i == 3) {
                break;
            }
            print_int(i);
        }
    }
```

0 1 2

test-continue1.arbol

```
    function void main(){
        int i;
        for(i = 0; i < 5; i = i + 1) {
            if (i == 3) {
                continue;
            }
            print_int(i);
        }
    }
```

0 1 2 4

test-fibonacci.arbol

```
    function int fib(int x) {
        if (x < 2) {
            return 1;
        }
        return fib(x-1) + fib(x-2);
    }

    function void main(){
        print_int(fib(5));
    }
```

8

test-float1.arbol

```
function void main(){
  float a;
  a = 3.14159267;
  print_float(a);
}
```

3.141593

test-float2.arbol

```
function void main(){
  float a;
  float b;
  float c;
  a = 3.14159267;
  b = -2.71828;
  c = a + b;
  print_float(c);
}
```

0.423313

test-float3.arbol

```
function void testfloat(float a, float b) {
  print_float(a + b);
  print_float(a - b);
  print_float(a * b);
  print_float(a / b);
}

function void main(){
  float c;
  float d;

  c = 42.0;
  d = 3.14159;

  testfloat(c, d);
}
```

45.141590 38.858410 131.946780 13.369027

test-for1.arbol

```
function void main(){
    int i;
    for(i = 0; i < 5; i = i + 1){
        print_int(i);
    }
    print_int(42);
}
```

0 1 2 3 4 42

test-func1.arbol

```
function int add(int x, int y) {
  return x + y;
}

function void main(){
  int a = add(2, 2);
  print_int(a);
}
```

4

test-func2.arbol

```
function int fun(int x, int y)
{
  return 0;
}

function void main()
{
  int i;
  i = 1;

  fun(i = 2, i = i + 1);

  print_int(i);
}
```

2

test-func3.arbol

```
function void printall(int a, int b, int c, int d) {
  print_int(a);
  print_int(b);
  print_int(c);
  print_int(d);
}

function void main() {
  printall(42,17,192,8);
}
```

42 17 192 8

test-func4.arbol

```
function void main() {
  return;
}
```

null

test-func5.arbol

```
function int bar(int a, boolean b, int c) {
  return a + c;
}

function void main() {
  print_int(bar(17,false,25));
}
```

42

test-func6.arbol

```
int a;

function void foo(int c) {
  a = c + 42;
}
```

```
function void main() {
  foo(73);
  print_int(a);
}
```

115

test-gcd1.arbol

```
function int gcd(int a, int b) {
  while (a != b) {
    if (a > b) a = a - b;
    else b = b - a;
  }
  return a;
}

function void main() {
  print_int(gcd(2,14));
  print_int(gcd(3,15));
  print_int(gcd(99,121));
}
```

2 3 11

test-gcd2.arbol

```
function int gcd(int a, int b) {
  while (a != b)
    if (a > b) a = a - b;
    else b = b - a;
  return a;
}

function void main() {
  print_int(gcd(14,21));
  print_int(gcd(8,36));
  print_int(gcd(99,121));
}
```

7 4 11

test-height-printlevel.arbol

```
function int get_height(int @root) {
    int lheight = 0;
    if ([root != null) {
        lheight = get_height([root);
    }

    int rheight = 0;
    if (]root != null) {
        rheight = get_height(]root);
    }

    if (lheight > rheight) {
        return lheight + 1;
    } else {
        return rheight + 1;
    }
}

function void print_level(int @root, int level) {
    if (level == 0) {
        print_int(~root);
        return;
    }

    if ([root != null) {
        print_level([root, level - 1);
    }

    if (]root != null) {
        print_level(]root, level - 1);
    }
}

function void main() {
    int @a => 1;
    int @b => 2;
    int @c => 3;
    int @d => 4;
    int @e => 5;
```

```
        int @f => 6;
        int @g => 7;

        a <-- b;
        a --> c;
        b <-- d;
        b --> e;
        e --> f;
        f --> g;

        int height = get_height(a);
        int i;

        for(i = 0; i < height; i = i + 1) {
            print_level(a, i);
        }
    }
```

1 2 3 4 5 6 7

test-height1.arbol

```
    function int get_height(int @root) {
        int lheight = 0;
        if ([root != null) {
            lheight = get_height([root);
        }

        int rheight = 0;
        if (]root != null) {
            rheight = get_height(]root);
        }

        if (lheight > rheight) {
            return lheight + 1;
        } else {
            return rheight + 1;
        }
    }

    function void main() {
```

```
        int @a => 1;
        int @b => 2;
        int @c => 3;
        int @d => 4;
        int @e => 5;

        a <-- b;
        a --> c;
        b <-- d;
        b --> e;

        print_int(get_height(a));
    }
```

3

test-height2.arbol

```
    function int get_height(int @root) {
        int lheight = 0;
        if ([root != null) {
            lheight = get_height([root);
        }

        int rheight = 0;
        if (]root != null) {
            rheight = get_height(]root);
        }

        if (lheight > rheight) {
            return lheight + 1;
        } else {
            return rheight + 1;
        }
    }

    function void main() {
        int @a => 1;
        int @b => 2;
```

```
        int @c => 3;
        int @d => 4;
        int @e => 5;
        int @f => 6;
        int @g => 7;

        a <-- b;
        a --> c;
        b <-- d;
        b --> e;
        e --> f;
        f --> g;

        print_int(get_height(a));
    }
```

5

test-hello1.arbol

```
    function void main(){
      print("hello world!");
    }
```

hello world!

test-inorder1.arbol

```
    function void inorder_int(int @root) {
        # traverse the left subtree
        if ([root != null) {
            inorder_int([root);
        }

        # visit the root
        print_int(~root);

        if (]root != null) {
            inorder_int(]root);
        }
    }
```

```
function void main() {
    int @a => 1;
    int @b => 2;
    int @c => 3;
    int @d => 4;
    int @e => 5;

    a <-- b;
    a --> c;
    b <-- d;
    b --> e;

    inorder_int(a);
}
```

4 2 5 1 3

test-mod1.arbol

```
function void main() {
    int x = 0;
    int y = 3;
    int z = 6;

    int a = y%z;
    print_int(a);
}
```

3

test-node1.arbol

```
function void main(){
  int @a => 5;
  int b;
  b = ~a;
  print_int(b);
}
```

5

test-node2.arbol

```
function void main() {
    int x = 5;
    int @a => 5;
    test(a);
}

function void test(int@ a) {
    print_int(~a);
}
```

5

test-node3.arbol

```
function void main() {
    string @a => "a";
    string @b => "b";
    string @c => "c";
    a <-- c;
    a --> b;
    print(get_leftchild(a));
}

function string get_leftchild(string @t) {
    string @x = [t;
    return ~x;
}
```

c

test-node4.arbol

```
function void main() {
    string @a => "a";
    string @b => "b";
    string @c => "c";
    a <-- c;
    a --> b;
    print(get_rightchild(a));
}

function string get_rightchild(string @t) {
```

```
        string @x = ]t;
        return ~x;
    }
```

b

test-node5.arbol

```
    function void main() {
        int x = 5;
        int @b => 6;
        int @a => ~b;
        print_int(~a);
    }
```

6

test-node6.arbol

```
    function void main() {
        float @a;
        float@ x;
        float @y;
        a => 3.42;
        print_float(~a);
    }
```

3.42

test-node7.arbol

```
    function void main() {
        string @a => "a";
        string @b => "b";
        string @c => "c";
        string @d => "d";
        string @e => "e";

        a <-- c;
        a --> b;
        b <-- d;
        b --> e;
```

```
        print(get_leftchild(a));
        print(get_rightchild(a));
        print(get_leftchild(b));
        print(get_rightchild(b));

        string @right_child_of_a = ]a;


        print(get_rightchild(right_child_of_a));


        print(get_leftchild(right_child_of_a));
    }

    function string get_rightchild(string @t) {
        string @x = ]t;
        return ~x;
    }

    function string get_leftchild(string @t) {
        string @x = [t;
        return ~x;
    }
```

c b d e e d

test-null.arbol

```
    function void main() {
        int @a => 5;

        if ([a == null) {
            print("Node a is null");
        }
    }
```

Node a is null

test-postorder1.arbol

```
    function void postorder_int(int @root) {
```

```
        if ([root != null) {
            postorder_int([root);
        }

        if (]root != null) {
            postorder_int(]root);
        }

        # visit the root
        print_int(~root);
    }

    function void main() {
        int @a => 1;
        int @b => 2;
        int @c => 3;
        int @d => 4;
        int @e => 5;

        a <-- b;
        a --> c;
        b <-- d;
        b --> e;

        postorder_int(a);
    }
```

4 5 2 3 1

test-preoder.arbol

```
    function void preorder_string(string @root) {
        # visit the root
        print(~root);

        # traverse the left subtree
        if ([root != null) {
            preorder_string([root);
        }

        if (]root != null) {
```

```
            preorder_string(]root);
        }
    }

    function void main() {
        string @a => "a";
        string @b => "b";
        string @c => "c";
        a <-- c;
        a --> b;
        preorder_string(a);
    }
```

a c b

test-preorder2.arbol

```
    function void preorder_int(int @root) {
        # visit the root
        print_int(~root);

        # traverse the left subtree
        if ([root != null) {
            preorder_int([root);
        }

        if (]root != null) {
            preorder_int(]root);
        }
    }

    function void main() {
        int @a => 1;
        int @b => 2;
        int @c => 3;
        int @d => 4;
        int @e => 5;

        a <-- b;
        a --> c;
        b <-- d;
```

```
        b --> e;

        preorder_int(a);
    }
```

1 2 4 5 3

test-printlevel.arbol

```
    function void print_level(int @root, int level) {
        if (level == 0) {
            print_int(~root);
            return;
        }

        if ([root != null) {
            print_level([root, level - 1);
        }

        if (]root != null) {
            print_level(]root, level - 1);
        }
    }

    function void main() {
        int @a => 1;
        int @b => 2;
        int @c => 3;
        int @d => 4;
        int @e => 5;

        a <-- b;
        a --> c;
        b <-- d;
        b --> e;

        int i;
        int height = 3;

        for(i = 0; i < height; i = i + 1){
            print_level(a, i);
```

```
        }
    }
```

1 2 3 4 5

test-subtract1.arbol

```
    function int subtract(int x, int y) {
      return x - y;
    }

    function void main(){
      print_int((subtract(10, 8)));
    }
```

2

test-subtract2.arbol

```
    function int subtract(int x, int y) {
      return x - y;
    }

    function void main(){
      print_int((subtract(5, 10)));
    }
```

-5

test-while1.arbol

```
    function void main(){
      int i;
      i = 5;
      while (i > 0) {
        print_int(i);
        i = i - 1;
      }
      print_int(42);
    }
```

5 4 3 2 1 42

test-while2.arbol

```
function int foo(int a) {
  int j;
  j = 0;
  while (a > 0) {
    j = j + 2;
    a = a - 1;
  }
  return j;
}

function void main(){
  print_int(foo(7));
}
```

14

11.7 Git Logs

This project shows a history of 129 commits from February 23rd and ending on April 26th. Although the commit history may appear partially skewed, it is important to note that the majority of our implementation strategy involved “Live Share” through Visual Studio code. This allowed one team member to display the code on his screen, while the rest of the team members discussed and implemented changes to our code instantaneously. This resulted in additional commits by one team member more than the others; however, the entirety of our team collaborated in order to produce such commits. Hence, this facilitated a more intimate and comprehensive collaboration framework.

```
commit 4b6ab422628f2703463d22542ceb8924399f6776 (HEAD -> master,
origin/master, origin/HEAD)
Author: anthonypalmeira <anthonypalmeira@gmail.com>
Date:   Mon Apr 26 22:17:15 2021 -0300

    git logs txt file

commit e325f76a067e1cde8ea5b32021107a8f947a22f6 (HEAD -> master,
origin/master, origin/HEAD)
```

```
Author: derekhuizhang <derekhuizhang@gmail.com>
Date:   Mon Apr 26 19:58:04 2021 -0400

    delete tests_suite

commit 863236f0de3f1180a068ced92e431519331d7c84
Author: derekhuizhang <derekhuizhang@gmail.com>
Date:   Mon Apr 26 14:31:07 2021 -0400

    add run_arbol

commit 670fc7019a95daef5a96ef2f46cac773d14d9716
Author: anthonypalmeira <anthonypalmeira@gmail.com>
Date:   Mon Apr 26 14:54:31 2021 -0300

    test/continue

commit ea5b2637ad3f66abd10b590edce34d4ff8ef35ba
Merge: f560105 212d59e
Author: derekhuizhang <derekhuizhang@gmail.com>
Date:   Mon Apr 26 03:47:27 2021 -0400

    Merge pull request #21 from amm2497/continue-break

    Continue break

commit 212d59e065292b8eb371cfa1c6463ddaf2b493a1
Author: derekhuizhang <derekhuizhang@gmail.com>
Date:   Mon Apr 26 03:45:50 2021 -0400

    add break and continue

commit ac11df9bc14cfc105ff293f6e39711768cda2416
Author: derekhuizhang <derekhuizhang@gmail.com>
Date:   Mon Apr 26 02:10:26 2021 -0400

    add tests

commit f56010589b80f26cbde2b3c628f94226836907d0
Author: anthonypalmeira <anthonypalmeira@gmail.com>
Date:   Sun Apr 25 18:38:48 2021 -0300

    HEIGHT, PRINT LEVEL and MIX OF BOTH tests

commit bb684547d4ea2b63b46a4f7e1a9b1f8070851f26
Author: anthonypalmeira <anthonypalmeira@gmail.com>
Date:   Sun Apr 25 18:11:10 2021 -0300
```

```
        PRINT LEVEL traversal

    commit 3eaca30d67e9af9a3ba13ac24de6a90bc18a6a29
    Author: anthonypalmeira <anthonypalmeira@gmail.com>
    Date:   Sun Apr 25 15:25:41 2021 -0300

        Tests for INORDER/POSTORDER traversal

    commit bc2433a8247396208028c45a2d4f3677cbfc57f1
    Author: anthonypalmeira <anthonypalmeira@gmail.com>
    Date:   Sun Apr 25 15:18:55 2021 -0300

        Pre-order traversal tests

    commit 4b38f8a196e9d49e931bcb808a289aee92cbf3e7
    Author: anthonypalmeira <anthonypalmeira@gmail.com>
    Date:   Sun Apr 25 15:04:04 2021 -0300

        null test

    commit f0457da7c5ada880b136fcd13d59e86e6335fc6e
    Author: anthonypalmeira <anthonypalmeira@gmail.com>
    Date:   Sun Apr 25 14:48:54 2021 -0300

        Removed some tests

    commit f12eff5e3483c7f8a79c080f6cf3321e74b8633c
    Author: anthonypalmeira <anthonypalmeira@gmail.com>
    Date:   Sun Apr 25 14:37:53 2021 -0300

        test-node7 which multiple children. This could be four fav
    program for presentaion

    commit 26be84938c320d5ad854dda6c49417c47f3cae4b
    Author: anthonypalmeira <anthonypalmeira@gmail.com>
    Date:   Sun Apr 25 14:07:06 2021 -0300

        test node with float

    commit fea9dedc914f171b547e720b92b7f3de8a261a6a
    Author: Anthony Palmeira
    <66444090+anthonypalmeira@users.noreply.github.com>
    Date:   Sun Apr 25 14:05:49 2021 -0300

        Codegen null (#20)

        * add null type
```

```
        * cleaned up some stuff

        Co-authored-by: derekhuizhang <derekhuizhang@gmail.com>

    commit f989d80333578af27b22b662066e3de398223fc6
    (origin/codegen-null)
    Merge: 371a59d 50c3d58
    Author: Anthony Palmeira
    <66444090+anthonypalmeira@users.noreply.github.com>
    Date:   Sun Apr 25 14:05:27 2021 -0300

        Merge branch 'master' into codegen-null

    commit 50c3d587d9105e1397126d5db1f8686a45b9ac56
    Author: anthonypalmeira <anthonypalmeira@gmail.com>
    Date:   Sun Apr 25 14:00:39 2021 -0300

        new node test

    commit acf88d803c3947ecde612307d397c79a7924dd6d
    Author: anthonypalmeira <anthonypalmeira@gmail.com>
    Date:   Sun Apr 25 13:35:11 2021 -0300

        CLI tool instructions to README

    commit 371a59dd28548c1f0d26ec075b40922fb9d162fa
    Author: derekhuizhang <derekhuizhang@gmail.com>
    Date:   Sun Apr 25 09:27:39 2021 -0400

        cleaned up some stuff

    commit 3675542bb8db7ab3ebef602bad875673527a7cbf
    Author: derekhuizhang <derekhuizhang@gmail.com>
    Date:   Sun Apr 25 09:24:07 2021 -0400

        add null type

    commit e554864edda002fc63e42be9aac589b5fb1db7c2
    Author: Anthony Palmeira
    <66444090+anthonypalmeira@users.noreply.github.com>
    Date:   Sun Apr 25 02:42:23 2021 -0300

        Anthony/more stuff (#19)

        * Modified CLI tool. Now gives us output error or compile
    error

        * Updates test
```

```
    * updated README

    * Now able to run a single test. by passing -t flag

commit 4636aba8227b7deb1272f3107d344c25435dbde6
(origin/anthony/more-stuff, anthony/more-stuff)
Author: anthonypalmeira <anthonypalmeira@gmail.com>
Date:   Sun Apr 25 02:41:30 2021 -0300

    Now able to run a single test. by passing -t flag

commit 0211ece24ebd269103233ea78357d19d58670db0
Author: anthonypalmeira <anthonypalmeira@gmail.com>
Date:   Sun Apr 25 02:33:15 2021 -0300

    updated README

commit 247de3657ef9005e29317eab028097624d9841b5
Author: anthonypalmeira <anthonypalmeira@gmail.com>
Date:   Sun Apr 25 02:29:14 2021 -0300

    Updates test

commit 8e359ba77eb943c1f48a00a401e75e3e2fddea2c
Author: anthonypalmeira <anthonypalmeira@gmail.com>
Date:   Sun Apr 25 02:21:19 2021 -0300

    Modified CLI tool. Now gives us output error or compile error

commit f7bb1a987b608bc2e0e0a8cec71b553582c3d30d
Author: amm2497 <amccormickk08@gmail.com>
Date:   Sun Apr 25 00:52:50 2021 -0400

    Working through built-in function implementations for
preorder - still debugging

commit 2fc9ab4c4059f9cb993f0632172f78578eb214f9
Author: amm2497 <amccormickk08@gmail.com>
Date:   Sat Apr 24 23:59:34 2021 -0400

    code gen preorder

commit db325aeec74d730b9eeb4435042c382c2f44acae
Author: anthonypalmeira <anthonypalmeira@gmail.com>
Date:   Sun Apr 25 00:57:53 2021 -0300

    node 4 tests right child
```

```
commit 8f5a258865ccb14e204f931267196a5b2c60a0d6
Author: anthonypalmeira <anthonypalmeira@gmail.com>
Date:   Sun Apr 25 00:42:27 2021 -0300

    node test 3

commit 774e36feed267141bbebc841eafd9f02797470ca
Author: anthonypalmeira <anthonypalmeira@gmail.com>
Date:   Sun Apr 25 00:38:52 2021 -0300

    node2 test

commit 8ba0fe5cd26b91632b5463632874490b009c80a4 (refs/stash)
Merge: e5713b5 702ca8d
Author: anthonypalmeira <anthonypalmeira@gmail.com>
Date:   Sun Apr 25 00:32:17 2021 -0300

    WIP on master: e5713b5 Merge pull request #18 from
amm2497/codegen-node

commit 702ca8de9d056c504e012fd99261a9edb89d9ded
Author: anthonypalmeira <anthonypalmeira@gmail.com>
Date:   Sun Apr 25 00:32:17 2021 -0300

    index on master: e5713b5 Merge pull request #18 from
amm2497/codegen-node

commit 176c633f86a97ef82acc36918a14e77cf9a0db2f
Author: amm2497 <amccormickk08@gmail.com>
Date:   Sat Apr 24 23:30:41 2021 -0400

    anthony/ fixed node1 test

commit c569a94209b4c1a3ad2cf82e88774a97edbaf8a5
Author: amm2497 <amccormickk08@gmail.com>
Date:   Sat Apr 24 23:28:45 2021 -0400

    anthony/ fixed tests and codegen

commit e0996247e48e27dd7858debbdd41ae2e2ea8a3fb
Author: amm2497 <amccormickk08@gmail.com>
Date:   Sat Apr 24 22:43:58 2021 -0400

    Edited double_type to implement print_float successfully in
codegen

commit 28ba2d33681f1b7491142a9844a9b5977956c7ff
```

```
Author: amm2497 <amccormickk08@gmail.com>
Date:   Sat Apr 24 22:31:35 2021 -0400

    Working through print_float tests and debugging

commit 2080809486c01548b394df0c869b676eeeb44552
(origin/tests-built-ins)
Author: amm2497 <amccormickk08@gmail.com>
Date:   Sat Apr 24 22:15:19 2021 -0400

    Implemented print_float successfully in codegen

commit e5713b555e0ef9c1e73843e52b71a8356addd22a
Merge: ac23c88 c7af5f5
Author: derekhuizhang <derekhuizhang@gmail.com>
Date:   Sat Apr 24 22:12:53 2021 -0400

    Merge pull request #18 from amm2497/codegen-node

    Codegen node

commit c7af5f59c646f7a12e00de73306cf0b9f86e96eb
(origin/codegen-node)
Author: derekhuizhang <derekhuizhang@gmail.com>
Date:   Sat Apr 24 22:12:44 2021 -0400

    update syntax

commit b2150773677a29b94e32979cfc137ba922dad5b8
Merge: 4e82bab ac23c88
Author: derekhuizhang <derekhuizhang@gmail.com>
Date:   Sat Apr 24 22:09:20 2021 -0400

    Merge branch 'master' into codegen-node

commit 4e82bab178260469a4f9bf84065902f5cf1b2afa
Author: derekhuizhang <derekhuizhang@gmail.com>
Date:   Sat Apr 24 21:49:47 2021 -0400

    add tests

commit 674d91512295a6d2f477a8580cf194a4e015b27c
Author: derekhuizhang <derekhuizhang@gmail.com>
Date:   Sat Apr 24 21:30:39 2021 -0400

    fixed l/r child

commit ac23c8879c4746e92bcf306d4d369dc92c784a37
```

```
Author: anthonypalmeira <anthonypalmeira@gmail.com>
Date:   Sat Apr 24 22:05:00 2021 -0300

    remove try except

commit ff7e0606b54917d0d430cf3bab48654df4daa7a0
Author: anthonypalmeira <anthonypalmeira@gmail.com>
Date:   Sat Apr 24 21:59:01 2021 -0300

    produced files deleted by make

commit 9dd6fccbc2d023002dbd43519ccf2e5b693a20fb
Author: amm2497 <amccormickk08@gmail.com>
Date:   Sat Apr 24 19:59:55 2021 -0400

    Trying to implement print_float in code gen for arbol tests
involving floats

commit 2f2d1f03eb9354d5fa6b32e9ad9ba04ef6ea20be
(anthony/file-cleansing-update)
Author: amm2497 <amccormickk08@gmail.com>
Date:   Sat Apr 24 19:23:07 2021 -0400

    Debugging CLI tool for arbol file tests and implemented
print_int in codegen

commit 004d0d74f013d673743c37bbc0cd4a1f40fb64cb
Author: Anthony Palmeira
<66444090+anthonypalmeira@users.noreply.github.com>
Date:   Sat Apr 24 19:25:35 2021 -0300

    More Tests (#17)

    * node tests

    * Test

    * charminus, modulus, hello tests

    * nl

    * .

    * Changes to CLI. Compile Error/Output Error

    * subprocess

    * c files
```

```
commit 5560c80050fa99d12db2b3f0a07cc2d4d8d724b0
(origin/anthony/more-tests, anthony/more-tests)
Author: anthonypalmeira <anthonypalmeira@gmail.com>
Date:   Sat Apr 24 19:25:01 2021 -0300

    c files

commit 94bd693297007e4f2a589feac01d0bba37c70cde
Author: amm2497 <amccormickk08@gmail.com>
Date:   Sat Apr 24 17:11:40 2021 -0400

    Deleted LBRACK and RBRACK tokens from scanner/parser bc
arrays not supported

commit 10b7bfff16ab62db6f8a2b372ac5e0329f064b6b
Author: anthonypalmeira <anthonypalmeira@gmail.com>
Date:   Sat Apr 24 17:56:45 2021 -0300

    subprocess

commit 4c655d4d7397709d24d8a4a4b269497279e8bab7
Author: anthonypalmeira <anthonypalmeira@gmail.com>
Date:   Sat Apr 24 16:52:28 2021 -0300

    Changes to CLI. Compile Error/Output Error

commit 3ac16fcf50ec10abf5ff953fc906675bfb9da8aa
Author: anthonypalmeira <anthonypalmeira@gmail.com>
Date:   Sat Apr 24 16:35:54 2021 -0300

    .

commit 57028d680b9b613924a4d9a87abae6cac467306e
Author: anthonypalmeira <anthonypalmeira@gmail.com>
Date:   Sat Apr 24 16:34:56 2021 -0300

    nl

commit a6456b4a58f5bf8c9edbf02c1dbc93b5e54167e7
Author: anthonypalmeira <anthonypalmeira@gmail.com>
Date:   Sat Apr 24 16:32:48 2021 -0300

    charminus, modulus, hello tests

commit 4d27535b088224a25b972b262764016dc3efef69
Author: anthonypalmeira <anthonypalmeira@gmail.com>
Date:   Sat Apr 24 16:17:37 2021 -0300
```

```
     Test

commit 0e5fc717dfd5e999b2177570276f6858589cf66f
Author: derekhuizhang <derekhuizhang@gmail.com>
Date:   Sat Apr 24 13:26:40 2021 -0400

    fix

commit 2b4164e22cf8c51810d8f0975238a54cebd4fed5
Author: derekhuizhang <derekhuizhang@gmail.com>
Date:   Sat Apr 24 13:06:04 2021 -0400

    lchild/rchild

commit ecc017b7bada15a365e61d0ee40cd7189323080d
Author: anthonypalmeira <anthonypalmeira@gmail.com>
Date:   Sat Apr 24 13:36:49 2021 -0300

    node tests

commit ab2eb2fceeb26e835c6e439ea104666952c3c5e0
Author: derekhuizhang <derekhuizhang@gmail.com>
Date:   Sat Apr 24 03:40:50 2021 -0400

    add node sntrax

commit 4652db06682a3e199e6397d2aef9b4d6b4695a00
Author: derekhuizhang <derekhuizhang@gmail.com>
Date:   Sat Apr 24 01:27:29 2021 -0400

    getting segfault

commit c8bab3a790fc765909b7829ebfdc533643f8d562
Author: Anthony Palmeira
<66444090+anthonypalmeira@users.noreply.github.com>
Date:   Fri Apr 23 19:58:26 2021 -0300

    CLI Tool and Tests (#14)

    * CLI Tool For Running Tests

    * readme

    * update

    * Add/Subtract tests
```

```
        * line

        * Arithmetics

        * .

        * fib test

        * fib

        * fixtures

        * Asserting output files

        * nl

        * More tests

        * nl

    commit 491759f75a368a5c40a450a1a04ee834bb72c9d3
    Merge: e3c7d7a 63c2346
    Author: derekhuizhang <derekhuizhang@gmail.com>
    Date:   Fri Apr 23 18:26:16 2021 -0400

        merge

    commit 63c2346209ef71aabeaa32119bfb594addc27adc
    Author: derekhuizhang <derekhuizhang@gmail.com>
    Date:   Fri Apr 23 18:25:28 2021 -0400

        update gitignore

    commit 53c0e1e126825e3c67a57833651b6e7bedecdcf3
    (origin/anthony/test)
    Author: anthonypalmeira <anthonypalmeira@gmail.com>
    Date:   Fri Apr 23 19:24:28 2021 -0300

        nl

    commit 7f6ef792ae445f449dfce4290deb7da4548a0a73
    Author: derekhuizhang <derekhuizhang@gmail.com>
    Date:   Fri Apr 23 18:24:10 2021 -0400

        update gitignore

    commit e3c7d7ac55c9cbb774db0f0629b3dcd0b1d6e27d
    Merge: 80c38ce 5ba34ff
```

```
Author: derekhuizhang <derekhuizhang@gmail.com>
Date:   Fri Apr 23 18:22:54 2021 -0400

    fix conflicts

commit 5ba34ffa750451e72c439e2c5677dbbd2cad23b6
Author: derekhuizhang <derekhuizhang@gmail.com>
Date:   Fri Apr 23 18:14:38 2021 -0400

    update everything

commit 689723348f4dd05e19b8186374cb1c282d8db4b5
Merge: 17a5002 50b0f48
Author: derekhuizhang <derekhuizhang@gmail.com>
Date:   Fri Apr 23 16:55:50 2021 -0400

    Merge pull request #16 from amm2497/andrea-refactor

    Andrea refactor

commit 50b0f48af4a611bb294ff28940e664a0d60b798d
(origin/andrea-refactor)
Merge: 378d7d4 17a5002
Author: derekhuizhang <derekhuizhang@gmail.com>
Date:   Fri Apr 23 16:55:25 2021 -0400

    fix conflicts

commit 80c38ceb0aaf9cc13ebb4d8f3b94c597a72dbf9c
Author: derekhuizhang <derekhuizhang@gmail.com>
Date:   Fri Apr 23 16:46:22 2021 -0400

    add node skeleton

commit a9c1305dd95746f35ed51520d32f1755001d862f
Author: anthonypalmeira <anthonypalmeira@gmail.com>
Date:   Fri Apr 23 16:32:22 2021 -0300

    More tests

commit 0f79490ce0ed4dc6b8e06138a8213cc79288de17
Merge: 5aa89a1 17a5002
Author: anthonypalmeira <anthonypalmeira@gmail.com>
Date:   Fri Apr 23 15:31:29 2021 -0300

    Merge branch 'master' into anthony/test

commit 378d7d46e96e5f953f40034931182516fb1e7745
```

```
Author: amm2497 <amccormickk08@gmail.com>
Date:   Fri Apr 23 13:23:07 2021 -0400

    Added additional if statement tests with binop expr

commit f62131d1552e2498d1fe74bc2a010ee1da8e0a4d
Author: amm2497 <amccormickk08@gmail.com>
Date:   Fri Apr 23 13:05:10 2021 -0400

    Added pass/fail tests for all assignments, binops, unops

commit 17a5002e7de1ae9ab8fe1a96f8fc756e6c8e9029
Merge: 6f92fcb bcca51c
Author: derekhuizhang <derekhuizhang@gmail.com>
Date:   Thu Apr 22 21:56:45 2021 -0400

    Merge pull request #15 from amm2497/derek-refactor

    Derek refactor

commit bcca51cbc7a8737fb1bb265e08553070e281769a
Author: derekhuizhang <derekhuizhang@gmail.com>
Date:   Thu Apr 22 21:56:00 2021 -0400

    add basics

commit 6474e1bb619324316b0faff7d4f6b098fbc0b69b
Author: amm2497 <amccormickk08@gmail.com>
Date:   Thu Apr 22 16:32:10 2021 -0400

    Inserted Derek's additions and trying to debug testing errors
with LLVM package

commit 8bb57c1b82162ed1b5bd009a2115c8df84bb46ee
Author: derekhuizhang <derekhuizhang@gmail.com>
Date:   Wed Apr 21 02:38:42 2021 -0400

    add more tests

commit 7230b5dfc7d47d5250e77ce317cd21fc193a5e45
Author: derekhuizhang <derekhuizhang@gmail.com>
Date:   Wed Apr 21 02:31:38 2021 -0400

    add codegen

commit 2d4f408dde5fe53dc7b9a3c7c729e21ffbaa283d
Author: derekhuizhang <derekhuizhang@gmail.com>
Date:   Tue Apr 20 00:09:07 2021 -0400
```

```
    add more statements

commit 313495a2cd0db9cf92b2734240ecaa15fd90acf4
Author: amm2497 <amccormickk08@gmail.com>
Date:   Mon Apr 19 22:51:22 2021 -0400

    Code Gen additions and extra test cases

commit a83ac1ea749c90ddb0eadb2f955f7e1ea15361e6
Author: derekhuizhang <derekhuizhang@gmail.com>
Date:   Mon Apr 19 22:42:40 2021 -0400

    add node test case

commit 400cbb89cf79165448819c6432ad8800b40350e6
Author: derekhuizhang <derekhuizhang@gmail.com>
Date:   Mon Apr 19 22:42:27 2021 -0400

    add node test case

commit f7380d18faf591cdc52011b0370e33882e15651a
Author: derekhuizhang <derekhuizhang@gmail.com>
Date:   Mon Apr 19 22:14:46 2021 -0400

    fixed boolean error

commit 6b29f72e3253ecb31031d2422c7cb503ded12a60
Author: derekhuizhang <derekhuizhang@gmail.com>
Date:   Mon Apr 19 22:06:58 2021 -0400

    fixed some parsing bugs

commit 477cb027da9d8ecb6214fc54c4d744688b8800b1
Merge: 9d79057 4479976
Author: derekhuizhang <derekhuizhang@gmail.com>
Date:   Mon Apr 19 21:49:07 2021 -0400

    wqMerge branch 'derek-refactor' of
https://github.com/amm2497/arbol_tree_language into
derek-refactor

commit 9d79057dd287e823fd724cd300e670d4c774ed00
Author: derekhuizhang <derekhuizhang@gmail.com>
Date:   Mon Apr 19 21:48:34 2021 -0400

    add loops and more expressions
```

```
commit 44799760852c5d7c323be602f8df2a3d52aac5a4
Author: amm2497 <amccormickk08@gmail.com>
Date:   Mon Apr 19 21:34:33 2021 -0400

    Added some test cases

commit 0a803c330a1cf17e93b6247f6ca44c012d7d3e68
Author: derekhuizhang <derekhuizhang@gmail.com>
Date:   Mon Apr 19 16:03:00 2021 -0400

    add almost everything

commit 35fc723354fe726422ae61511d42230e4e39ddb7
Author: derekhuizhang <derekhuizhang@gmail.com>
Date:   Mon Apr 19 14:12:42 2021 -0400

    add some more env stuff

commit e53f311db6cf58d0f30922b481b48c834ae08ad7
Author: derekhuizhang <derekhuizhang@gmail.com>
Date:   Mon Apr 19 04:59:49 2021 -0400

    add environment

commit 56fd566c079bfa2d18dfcf05247718dbedaa469e
Author: derekhuizhang <derekhuizhang@gmail.com>
Date:   Mon Apr 19 02:47:37 2021 -0400

    add expr to inside func

commit 6f92fcb6fe1a37bd671e7cedb5833d383ab7d56f
Author: amm2497 <amccormickk08@gmail.com>
Date:   Sun Apr 11 16:45:49 2021 -0400

    Minor error fix in sast

commit 7b242871816e8b636484812fcd7f9c9c18b75864
Author: derekhuizhang <derekhuizhang@gmail.com>
Date:   Sun Apr 11 02:12:54 2021 -0400

    added several more statements

commit 294073ed9e4316ea96c2e70eff63b1ee65a94ca2
Author: derekhuizhang <derekhuizhang@gmail.com>
Date:   Sun Apr 11 02:11:02 2021 -0400

    Added statements
```

```
commit 1d11f4ae0495623bea157a9c688a579cfefc4b2d
Author: derekhuizhang <derekhuizhang@gmail.com>
Date:   Sun Apr 11 01:12:11 2021 -0400

    add sexpr to sast

commit 3545706a6a8bf57d035b464729b38ed54bcd757d
Author: derekhuizhang <derekhuizhang@gmail.com>
Date:   Sat Apr 10 23:31:13 2021 -0400

    add node_val

commit 08fb4ce6325a9966d8a918bfbad11005b011bbb5
Author: derekhuizhang <derekhuizhang@gmail.com>
Date:   Sat Apr 10 23:26:17 2021 -0400

    minor type change

commit 0cafb061940678413f86995a9f2e39917d80a467
Author: derekhuizhang <derekhuizhang@gmail.com>
Date:   Sat Apr 10 22:30:42 2021 -0400

    fix types

commit 5aa89a1d7065c89112826aa9b7a7910cf42de0f1
Author: anthonypalmeira <anthonypalmeira@gmail.com>
Date:   Sat Apr 10 20:45:27 2021 -0300

    nl

commit 3fd33045d8508526640bba0331c0bf4d9d05d28b
Author: anthonypalmeira <anthonypalmeira@gmail.com>
Date:   Sat Apr 10 20:44:49 2021 -0300

    Asserting output files

commit 188b158e427910c4d985f37abbf319f204c88b20
Author: anthonypalmeira <anthonypalmeira@gmail.com>
Date:   Sat Apr 10 20:20:42 2021 -0300

    fixtures

commit 6a6ba0c2d59f0cea01dd4a540ae1922db4bcfe38
Author: anthonypalmeira <anthonypalmeira@gmail.com>
Date:   Sat Apr 10 18:33:52 2021 -0300

    fib
```

```
commit ffcb5cc2835c73ad7bcb02b455dbf30e73c8c66c
Author: anthonypalmeira <anthonypalmeira@gmail.com>
Date:   Sat Apr 10 18:33:38 2021 -0300

    fib test

commit 85594ab300f76359fc9596468fb96197a2f47504
Author: anthonypalmeira <anthonypalmeira@gmail.com>
Date:   Sat Apr 10 18:30:55 2021 -0300

    .

commit 1cbceff72c37e68d1f3557b3ef9219dbfb470e33
Author: anthonypalmeira <anthonypalmeira@gmail.com>
Date:   Sat Apr 10 18:29:23 2021 -0300

    Arithmetics

commit 371537462552185237 4664caff5f6fd5d19310e7
Author: anthonypalmeira <anthonypalmeira@gmail.com>
Date:   Sat Apr 10 18:22:57 2021 -0300

    line

commit b5cd5b843b677538635f38f5a2425788021ca793
Author: anthonypalmeira <anthonypalmeira@gmail.com>
Date:   Sat Apr 10 18:20:20 2021 -0300

    Add/Subtract tests

commit 39131b85afd1efbce6095e6d3e5e157ea6554c30
Author: anthonypalmeira <anthonypalmeira@gmail.com>
Date:   Sat Apr 10 17:38:50 2021 -0300

    update

commit 70a6d73bb482141c92fe88ecc8a30c99f3f6c607
Author: anthonypalmeira <anthonypalmeira@gmail.com>
Date:   Sat Apr 10 17:34:28 2021 -0300

    readme

commit 9a533bf661c5c1317744e335d5aa07bcff929d5c
Author: anthonypalmeira <anthonypalmeira@gmail.com>
Date:   Sat Apr 10 17:29:37 2021 -0300

    CLI Tool For Running Tests
```

```
commit 043223b498b1d7107dbfd89b072a9d8a73c38181
Author: derekhuizhang <derekhuizhang@gmail.com>
Date:   Wed Apr 7 15:13:35 2021 -0400

    changes

commit 08b0e9a1c1b95a634646291aec2ef049ba0870ed
Merge: bcde98d c9c24ae
Author: derekhuizhang <derekhuizhang@gmail.com>
Date:   Wed Apr 7 13:53:34 2021 -0400

    Merge pull request #13 from amm2497/andrea-debug

    Hello world milestone

commit c9c24aed32c8ed602ccc69baea63e3c339267b80
Author: derekhuizhang <derekhuizhang@gmail.com>
Date:   Wed Mar 24 22:42:19 2021 -0400

    modify makefile

commit b5176d7dc2647b4c5654fa496748a67ce4bf5592
Author: derekhuizhang <derekhuizhang@gmail.com>
Date:   Wed Mar 24 22:41:01 2021 -0400

    remove unnecessary files

commit 4fd810a5401230f4e40b38ef1a480b2d3e5a1dc3
Author: derekhuizhang <derekhuizhang@gmail.com>
Date:   Wed Mar 24 22:37:42 2021 -0400

    update readme

commit 1940b3a5eb02282b1cbba87ee0ecfa47ff9772e4
(origin/andrea-debug, andrea-debug)
Author: derekhuizhang <derekhuizhang@gmail.com>
Date:   Wed Mar 24 22:29:23 2021 -0400

    add codegen

commit 75efe51bf27c2a6b04ff113b0087a1c4b2943a3a
Author: derekhuizhang <derekhuizhang@gmail.com>
Date:   Wed Mar 24 19:19:41 2021 -0400

    fix mll

commit f34699d78af5fcc8c3ab6725629d77f3166e7a96
Author: amm2497 <amccormickk08@gmail.com>
```

```
Date:   Wed Mar 24 14:56:32 2021 -0400

    Working through debugging code generator to output hello
world

commit a09e960910a6ecb84e7a4f1efe3125a69fdaf7d9
Author: derekhuizhang <derekhuizhang@gmail.com>
Date:   Wed Mar 24 05:21:31 2021 -0400

    add semantic checking

commit f68cf5908704a050067c008e303951ff847da8d1
Author: anthonypalmeira <anthonypalmeira@gmail.com>
Date:   Wed Mar 24 02:34:41 2021 -0300

    codegen

commit e29ef362b7ca29781425cfb8c6116954ddb509dc
Author: anthonypalmeira <anthonypalmeira@gmail.com>
Date:   Wed Mar 24 02:33:20 2021 -0300

    updates

commit 11b6db3917570b2f47bfd815e98beb5716eaee8d
Author: anthonypalmeira <anthonypalmeira@gmail.com>
Date:   Wed Mar 24 02:24:06 2021 -0300

    updated SAST

commit 5197f1e5694d9bad0d1d5a7eaa73203974647f3b
Author: anthonypalmeira <anthonypalmeira@gmail.com>
Date:   Wed Mar 24 01:54:32 2021 -0300

    new ast

commit 44fa31220924c6e5b371a6aa829fe2b40cb9107e
Author: anthonypalmeira <anthonypalmeira@gmail.com>
Date:   Wed Mar 24 00:07:34 2021 -0300

    Updated SAST

commit 80cd7a6363261fec2805c76f903b95a0cd7ceccd
Author: amm2497 <amccormickk08@gmail.com>
Date:   Tue Mar 23 22:51:05 2021 -0400

    Committing updated AST
```

```
commit 47b2278e0efc6ccb6c9a4fad1ac5e0ba98ad9044
(origin/hello_world, hello_world)
Author: amm2497 <57160538+amm2497@users.noreply.github.com>
Date:   Tue Mar 23 22:41:10 2021 -0400

    Andrea debug (#12)

    * Debugged node vs. string error in AST

    * Started Code Generator for "Hello World"

    Very rough coding of code generator - based almost entirely
off of MicroC

    * Added test case for hello world

    * Debugging parser

    * Debugging scanner, parser, ast, sast

    * Ast work s successfully

commit 77ff583e6b411aa107f873b0d92c9c02e67db030
(origin/revert-10-andrea-debug)
Author: Anthony Palmeira
<66444090+anthonypalmeira@users.noreply.github.com>
Date:   Tue Mar 23 23:40:08 2021 -0300

    Revert "Andrea debug"

commit 965d9a416ae1d1733a07aa5feb31f811d5c7725b
Author: amm2497 <amccormickk08@gmail.com>
Date:   Tue Mar 23 22:39:09 2021 -0400

    Ast work s successfully

commit ef6588be53c38298fe5b98e59ea968eb5cbb552a
Author: amm2497 <amccormickk08@gmail.com>
Date:   Tue Mar 23 22:32:03 2021 -0400

    Debugging scanner, parser, ast, sast

commit e858b0f7062581cf1f93108824ed5c2665d71245
Author: amm2497 <amccormickk08@gmail.com>
Date:   Tue Mar 23 21:44:00 2021 -0400

    Debugging parser
```

```
commit ec1c97de0daebe1aea0e8253faeff966e5af090f
Author: amm2497 <amccormickk08@gmail.com>
Date:   Tue Mar 23 16:05:48 2021 -0400

    Added test case for hello world

commit 1ccc59c5c82008432703b8df41307b27b1614925
Author: amm2497 <amccormickk08@gmail.com>
Date:   Tue Mar 23 13:55:41 2021 -0400

    Started Code Generator for "Hello World"

    Very rough coding of code generator - based almost entirely
off of MicroC

commit 3b3454834c3a22d4c3c7bf7cb7bd0cc71e34f9a9
Author: amm2497 <amccormickk08@gmail.com>
Date:   Mon Mar 22 23:40:31 2021 -0400

    Debugged node vs. string error in AST

commit dfbc01649c3b1bf3cb85486169b37ec489491512
Merge: 50c7bb1 30ebe42
Author: amm2497 <57160538+amm2497@users.noreply.github.com>
Date:   Mon Mar 22 23:06:57 2021 -0400

    Merge pull request #10 from amm2497/andrea-debug

    Andrea debug

commit 30ebe42b4b2e6ca5742f198bb907167f28d30af3
Author: amm2497 <amccormickk08@gmail.com>
Date:   Mon Mar 22 23:04:21 2021 -0400

    Successfully debugged parser without any errors

commit 7065a4dbf05ec78c634e506a1b2555c319f7e032
Author: amm2497 <amccormickk08@gmail.com>
Date:   Mon Mar 22 22:54:19 2021 -0400

    Debugging node ops in parser

commit ea6de81d7d1e6cd63414950af5866159e79fcc1f
Author: amm2497 <amccormickk08@gmail.com>
Date:   Mon Mar 22 22:32:48 2021 -0400

    Parser debugged with zero shift/reduce errors
```

```
commit 50c7bb17337c5c2ba4ad2c24d9ed11779b113faf
Author: amm2497 <amccormickk08@gmail.com>
Date:   Mon Mar 22 21:52:03 2021 -0400

    Debugging shift/reduce errors for parser.mly

commit 6ebff146507f8e7dbc6e59acfede50498f19ad6d
(origin/anthony/parser-debug)
Author: anthonypalmeira <anthonypalmeira@gmail.com>
Date:   Mon Mar 22 22:43:07 2021 -0300

    39 shift/reduce, 5 reduce/reduce

commit 107fa4a3ea68df7981cf30991f2d8ca2d8e8a0a7
Author: amm2497 <amccormickk08@gmail.com>
Date:   Mon Mar 22 20:44:08 2021 -0400

    Debugging shift reduce errors in parser

commit fde1b9118473ec6fa685890aeb54d57c3540b0ba
Merge: 7492a41 1cf0931
Author: Anthony Palmeira
<66444090+anthonypalmeira@users.noreply.github.com>
Date:   Mon Mar 22 16:36:22 2021 -0300

    Merge pull request #8 from amm2497/anthony/builtin-funcs-sast

    Semantically Checked AST & C-code built-in functions

commit 1cf09317187779461ce9d227e6402bfb9bda8a43
(origin/anthony/builtin-funcs-sast)
Author: anthonypalmeira <anthonypalmeira@gmail.com>
Date:   Mon Mar 22 16:35:36 2021 -0300

    Updated SASt

commit 601dc348f79b6d0c3ae6c544c6fdd056fab57545
Merge: cd8174a 7492a41
Author: anthonypalmeira <anthonypalmeira@gmail.com>
Date:   Mon Mar 22 16:20:06 2021 -0300

    Merge branch 'hello_world' into anthony/builtin-funcs-sast

commit cd8174a6c9b6738dc7af62f5449f750fe666bce8
Author: anthonypalmeira <anthonypalmeira@gmail.com>
Date:   Mon Mar 22 16:16:03 2021 -0300

    SAST, C-code and stubs
```

```
commit 7492a418d537ee3b275fe7c61f7cdc54f9e4b901
Author: derekhuizhang <derekhuizhang@gmail.com>
Date:   Mon Mar 22 14:28:52 2021 -0400

    add to ast

commit 76f7edc3541e6f96eb9defcd401e1d329af32d96
Author: derekhuizhang <derekhuizhang@gmail.com>
Date:   Mon Mar 22 06:29:14 2021 -0400

    added makefile

commit 50815e51ca7982079728bb9c4fd36410a6c32ed7
Author: derekhuizhang <derekhuizhang@gmail.com>
Date:   Mon Mar 22 06:29:04 2021 -0400

    fixed some parser errors

commit e306540e07c5ad2f34dc51164e1d5f2e4bed3bea
Author: amm2497 <amccormickk08@gmail.com>
Date:   Sun Mar 21 23:28:35 2021 -0400

    Main additions to AST and simplifying node ops in
parser/scanner

    Based off of MicroC for most part except for node op

commit 995a2e507ed885732b5c16975b9846946a8e2222
Author: amm2497 <amccormickk08@gmail.com>
Date:   Sun Mar 21 17:35:44 2021 -0400

    Updated scanner and parser (and some of AST)

    Wrote out statements and expressions to ensure they made
sense & simplified some expressions. Still need to work on AST
more

commit 5bd41a8b0b69f0f1d1c8507b8d8a784a9f28d89d
Author: amm2497 <amccormickk08@gmail.com>
Date:   Sun Mar 21 14:50:54 2021 -0400

    Revert "Delete arbol_scanner.mll"

    This reverts commit 0216612f7c6b84c6018dff7b594f1ea590dcb3b1.

commit 16a2b28777293858c8fac077dd7bcbd0d41a9817
Author: amm2497 <57160538+amm2497@users.noreply.github.com>
```

```
Date:   Tue Mar 16 21:57:10 2021 -0500

    Update README.md

commit 25eb0a148614a332cf94de20d9a54fc5ccc4b72a
Author: amm2497 <57160538+amm2497@users.noreply.github.com>
Date:   Tue Mar 16 21:56:38 2021 -0500

    Update README.md

commit 47e75551efe01c9b6df3c4ebb940320cc16c2059
Author: amm2497 <57160538+amm2497@users.noreply.github.com>
Date:   Tue Mar 16 21:56:15 2021 -0500

    Create README.md

    Description of language and instructions on how to
run/compile each file

commit 8b08ea0919834af772f8b79a6b5f8f1018be903f
Author: amm2497 <57160538+amm2497@users.noreply.github.com>
Date:   Tue Mar 16 21:48:25 2021 -0500

    Delete arbol_parser.ml

commit d3fcf5449ab599bf31e5aa648046fb89d78a8f6c
Author: amm2497 <57160538+amm2497@users.noreply.github.com>
Date:   Tue Mar 16 21:48:16 2021 -0500

    Delete arbol_scanner.ml

commit 5823d1ecddef406b34d3c453b769464c00715506
Author: amm2497 <57160538+amm2497@users.noreply.github.com>
Date:   Tue Mar 16 21:47:58 2021 -0500

    Delete arbol_parser.mly

commit 0216612f7c6b84c6018dff7b594f1ea590dcb3b1
Author: amm2497 <57160538+amm2497@users.noreply.github.com>
Date:   Tue Mar 16 21:47:11 2021 -0500

    Delete arbol_scanner.mll

commit 6800e7ca5a4656ec72722699a4548cff617c7908
Author: amm2497 <amccormickk08@gmail.com>
Date:   Tue Mar 16 21:45:27 2021 -0500

    Array Implementation & Began AST
```

```
    Parser (and Scanner) now allow array implementation: array
assignment & access.

    Implemented all ops/expr/stmt/function dec/var dec in AST.

commit 496447ae46130830d96e6617349945cf806aad14
Author: amm2497 <amccormickk08@gmail.com>
Date:   Tue Mar 16 21:37:31 2021 -0500

    Array implementations

    Edits on Parser & Scanner (added array implementation)

commit bcde98d431360b8732856827cffc6b0bcfb76d77
Author: amm2497 <amccormickk08@gmail.com>
Date:   Mon Mar 15 10:54:48 2021 -0500

    Create Arbol_LRM copy.pdf

    LRM that we submitted for Arbol

commit e2ee0c6ef9663852d9d40caf78e0bdceaa67ce3f
Author: amm2497 <amccormickk08@gmail.com>
Date:   Wed Feb 24 23:55:28 2021 -0500

    LRM submission - fixing minor syntax details

    No shift errors - submission files for LRM (2/24)

commit 102dee616b0ea1e9c022fedb8f2206c8307c55cd
Author: amm2497 <amccormickk08@gmail.com>
Date:   Wed Feb 24 17:20:54 2021 -0500

    Added all tree syntax features to Parser

    No shift errors - all tree syntax implemented

commit 40c06b1ae4e8b2102801a7f49636946e356ac94b
Author: amm2497 <amccormickk08@gmail.com>
Date:   Wed Feb 24 16:25:57 2021 -0500

    Testing and debugging shift errors

    Got no shift errors with all (non-tree) syntax

    Have one shift error with tree syntax
```

```
commit 9aa6b3e515f0853a5afe7ee7b7fb7081519c131a
Author: amm2497 <amccormickk08@gmail.com>
Date:   Tue Feb 23 22:48:45 2021 -0500

    Node Ops added to Parser

    Added Node Ops to Parser & refined expressions and statements

commit 15c6c1bd749f04195977efaa8636f8ef15714f29
Author: amm2497 <amccormickk08@gmail.com>
Date:   Tue Feb 23 17:32:11 2021 -0500

    Scanner & Parser without specialized tree syntax

    ARBOL without special tree syntax
    and without ocamlyacc testing.
```