ARG! Programming Language

Ryan Eagan, Mike Goldin, River Keefer, Shivangi Saxena

1. Introduction

ARG! is a language to be used to make programming a less frustrating experience. It is similar to C in its syntax, but is dynamically typed and fully type inferenced. The language focuses on making coding simpler for users, and is especially good for scientific computing.

2. Tutorial

To start using ARG!, type `make` in the root folder. The following sample code demonstrates some of the features of ARG!. To run a program, first type `./argc <filename>`, then run it through gcc using `gcc <filename>.c`. The output can then be seen by running `./a.exe`.

The syntax for ARG! is very similar to C, with the exception of keywords, that are always in CAPS.

Sample 1:

```
a = "Hello World!";
PRINT ("%x", a);
```

The output is:

```
Hello World!
```

Sample 2:

Unlike C, ARG! can handle elements of different types in the same array:

```
len = 5;
arr[len] = {0, "hello", "yo", 678, 8};
i = 0;
WHILE(i < len) {
        PRINT("%x\n", arr[i]);
        i = i + 1;
}
```

The output is:

```
0
hello
yo
678
8
```

Sample 3:

The PRINT function allows printing of any kind of datatype with the same representation. This can be seen here:

```
FUNCTION printall(a,b,c,d) {
        PRINT("%x\n", a);
        PRINT("%x\n", b);
        PRINT("%x\n", c);
        PRINT("%x\n", d);
}

printall("hello", "world", 1, 2);
printall(2, "arg string", 1, 2);
```

The output is:

```
hello
world
1
2
2
arg string
1
2
```

Sample 4:

Control Flow:

```
a = true;
b = false;

IF(a == true) {
        PRINT("%x\n", "a is true");
}
ELSE {
        PRINT("%x\n", "a is false");
}
IF(b == true) {
        PRINT("%x\n", "b is true");
}
ELSE {
        PRINT("%x\n", "b is false");
}
```

The output is:

```
a is true
b is false
```

3. Reference Manual

a. Lexical Elements

There are six types of lexemes in ARG - identifiers, keywords, operators, separators and whitespace.

i. Identifiers

Identifiers are tokens that refer to variable or function names. They always start with a letter and may have an underscore or digits in them. Uppercase and lowercase letters are distinguished. Identifiers must be unique for each variable within the same scope.

ii. Keywords

These keywords cannot be used as identifiers. They include:

if, else, return, while, FUNCTION

iii. Operators

Operators are used for performing arithmetic operations and establishing precedence.

-, +, *, /, (), unary -, ==, !=, <=, >=, [<Int>]

iv. Separators

Separators are used to delimit code blocks, identifiers and statements. ARG includes the following separators:

{ } , ;

Whitespace is also a separator, but not a token.

v. Whitespace

Whitespace is used to indicate separation between two tokens. It is ignored unless specified as part of a character array. It may or may not be used between operators and operands. Any extra whitespace is treated as one white space.

vi. Comments

/* indicates the start of a comment and * / indicates its end. Multi-line comments are supported.

b. Data Types

There are two types of data structures -

i. Primitive

a. Int

These are 32-bit structures that will only hold numeric values in the range −2,147,483,648 to 2,147,483,647.

b. Char

These are 8-bit structures which can contain any member of the ASCII character set, including escaped characters.

c. Double

Double values are 64 bits and include 1 sign bit, 11 exponent bits and 52 significand bits.

d. Boolean

Booleans are 8 bits and represent a logical true or false. All types evaluate to true except for the explicit boolean false.

ii. Derived

Arrays

Arrays are collections of objects of the same type, stored contiguously in memory. They are derived by appending the [<int>] operator to an identifier, where <int> is an integer specifying the array's size.

c. Literals

i. Int Literal

An int literal can be any decimal number 0 - 9 and may be preceded by the unary -.

ii. Char Literal

A character literal can be any ASCII character, delimited by single quotes. They correspond to Char arrays.

iii. String Literal

A string literal can be any set of ASCII characters, delimited by double quotes.

iv. Double Literal

A double literal represents numbers with fractional values. They may include either a decimal point with a fraction or an 'e' followed by a signed integer. It can have a maximum value of 1e+63. Floats may be preceded by the unary -

v. Boolean Literal

Structures of this type may have one of two values: *t rue* or *f alse* . They are not strings.

d. Operators

i. Unary

- -

  Negation for Ints and Doubles
- [<Int>]

  Appended to the right of an identifier, specifies that the identifier should be an array of size <Int>.

ii. Binary

- Addition (+)
- Subtraction (-)
- Multiplication (*)
- Division (/)

- Less Than/equal to (<=)
- Greater Than/equal to (>=)
- Equals (==)
- Not equals (!=)

iii. Assignment (=)

Operator Precedence

a. Unary operators
b. Multiplication, Division
c. Addition, subtraction
d. Relational operators (<=, >=, ==, !=)
e. Assignment operator
Operators at the same level are grouped left-to-right.

e. Program Structure

i. Declarations

Since ARG does automatic type-inferencing, the data type of a variable need not be specified. Declarations have the form:

```
identifier = value;
```

ii. Statements

- **Expression statements**

These are statements which evaluate to some result which can be assigned to a variable, tested for logical truth or thrown away. These generally include assignment statements or calls to functions. They are of the form:

```
expression;
```

- **Conditional statements**

Conditional statements evaluate the logical verity of an expression and branch execution on the basis of the evaluation. These statements may be of the form:

```
if (<boolean-expression>) statement;
if (<boolean-expression>) statement else statement;
```

- **Iteration statements**

Iterators are loops which run until a specified expression evaluates as true. Syntax –

```
while (<boolean-expression>) statement;
```

- **return statement**

Return is used to return values from one function to its caller. It can be used to return zero or more values. Syntax –

```
return;
return expression;
```

Compound statements in any block can be put inside curly braces, e.g. -

```
{
        expression1;
        expression2;
}
```

- **FUNCTION statement**

The FUNCTION keyword simply denotes the declaration of a new function. Minimally, it must be followed by an identifier, a left-parenthesis, a right-parenthesis and a curly-brace delimited code block. Optionally, arguments can be specified by placing comma-delimited identifiers within the parentheses.

iii. Scope

In ARG, curly-braces delimit code blocks and scope. The "global" scope is itself a code block implicitly wrapped in curly-braces by the compiler. Identifiers declared in outer code blocks are accessible by inner code blocks, but the opposite is not true.

4. Project Plan

I. Planning & Development process

The planning for the project was iterative, with different milestones being set that were in line with the deadlines set by Prof. Edwards. Further short-term goals were set up for each week, and specific tasks were distributed among the different team members. The team met every week on Mondays, to discuss the progress of the week. Also, meetings with our TA David Arthur were helpful in clearing out any problems we faced during the development.

Stages of development of the compiler were based on the compiler architecture. We started with building the Parser & Scanner of the compiler, and then moved to working on the semantics of the language and translation. It was decided that Arg would be based on C, and the final translation would be done by GCC. Once we were done with that, we moved onto working on the semantic checker and adding the dynamic type inferencing to the language, along with a symbol table for checking scopes.

II. Testing process

Once we had a working compiler, we designed a test suite that tested the various features of our language. We were also able to add succinct error messages to be shown when necessary. The testing suite worked as follows - for each program, we included both the ARG! version as well another version written in C. After that we simply checked to see if the outputs for both programs were the same or not.

III. Team Responsibilities

Although the team initially started out with fixed roles, as specified by Prof Edwards (*Manager, Language Guru, System Architect, Tester*), work was rather distributed over various roles for each team member.

Team Member	Responsibilities
Michael Goldin	Compiler Architecture, Code Generation
Ryan Eagan	Testing
River Keefer	Compiler Architecture, Code Generation
Shivangi Saxena	Semantics, Documentation

IV. Style Guide

Since ARG! is based on C, its style conventions were inspired from the C style guide -

- Code lines with more than 80 characters are broken into multiple lines.
- Variable names are separated using underscores instead of using CamelCase.
- Indentation was done using four spaces.

V. Software Development Environment

- Programming languages used: Ocaml 4.01.0 for building the scanner, parser and architecture of the language, C for testing
- GCC - for translating output of the compiler into C
- Development Environment: Vim, Atom
- Github git Repository

VI. Project Timeline

September 21st	Language defined & roles assigned
September 30th	Language proposal submitted
October 19th	Creation of project repository, first commit
October 26th	Language Reference Manual submitted
November 14th	Basic scanner & parser
November 15th	Running Hello World program
December 8th	Completed ARG! 1.0
December 5th	Code generation complete

December 15th	Completed scanner & parser with all additional features
December 16th	Testing complete & demos added
December 16th	Final version of ARG! 1.2.1 released
December 18th	Project Presentation

VII. Project Log

```
commit 4723fb452b8b0f7940f02bd46a00196b6dec9fb6
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Thu Dec 17 08:52:19 2015 -0500

    Bug fix.

commit 3eb432720478300ba3410fe76eca3a57f559d4a9
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Thu Dec 17 08:43:15 2015 -0500

    Bug fix.

commit 8782cacc7b4f7a2f4f642f46d84dadc6b0839fc5
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Thu Dec 17 08:43:02 2015 -0500

    Bug fix

commit 3f8fca603799ec909484da0f90d6ff93dddbd60d
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Thu Dec 17 08:36:42 2015 -0500

    Bug fix.

commit a78754af6993f37a5b8c95f3aa9d2bf6cef52bdb
Author: Ryan Eagan <eaganry@gmail.com>
Date:   Thu Dec 17 00:49:19 2015 -0500

    added passarray and bubble test cases
    both failing

commit 0e216c0860cbced05f39de2655f0be5ba94b153a
Author: Ryan Eagan <eaganry@gmail.com>
Date:   Wed Dec 16 18:13:56 2015 -0500

    new tests
    else is failing
```

```
commit aaab81821df3780c265308061b9dd744d3fcdc38
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Wed Dec 16 17:26:17 2015 -0500

    Bug fixes

commit 74f99587ac2ef9d962df1d89d0606f4ff3d90b04
Author: Ryan Eagan <eaganry@gmail.com>
Date:   Wed Dec 16 16:25:27 2015 -0500

    demos added
    demo and demo2 display arrays / functions with ambiguous types
    demos 3 - 5 display error messages

commit 70d7f1e389220e9b4a77a60c432baff6daf527bb
Author: Ryan Eagan <eaganry@gmail.com>
Date:   Wed Dec 16 15:46:40 2015 -0500

    added bool test case, fails on first run

commit 8656f6851ce65e95dcd592fa85164c5d83374e14
Author: Ryan Eagan <eaganry@gmail.com>
Date:   Tue Dec 15 15:35:38 2015 -0500

    gcd removed single line ifs

commit 95dd1e8b4fa679e8157b49e37344f0af968f5aed
Author: Ryan Eagan <eaganry@gmail.com>
Date:   Tue Dec 15 15:34:33 2015 -0500

    return fixed, prints don't have extra strings

commit 8ab051ff74e1d4f738042d97cfb0a14d77ecf68d
Author: Ryan Eagan <eaganry@gmail.com>
Date:   Tue Dec 15 15:27:32 2015 -0500

    updated returns, added gcd test case

commit ccdb383a2a4b527f72efd5a599c70ef1c87d73a4
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Tue Dec 15 15:17:48 2015 -0500

    Bugs.

commit 75348522dff05d238bc539310aab096a6fcfcd4a
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Tue Dec 15 15:17:03 2015 -0500

    Bugs
```

```
commit db23082a175d39abd014d66be2081a260224e020
Merge: c88be3b 8038c30
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Tue Dec 15 15:15:36 2015 -0500

    Merge branch 'dev' of https://github.com/skmgoldin/arg into dev

commit c88be3beb952c77d74f2e102b101a0082463fc64
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Tue Dec 15 15:15:33 2015 -0500

    Support function returns.

commit 8038c309cea9858d3a3cd05338c253b8ea95f4f8
Merge: 200089c 1848e2e
Author: Ryan Eagan <eaganry@gmail.com>
Date:   Tue Dec 15 15:14:17 2015 -0500

    Merge branch 'dev' of https://github.com/skmgoldin/arg into dev

commit 200089cbdeda31f3a2d9a9b1e4dfbab4c0f01fe2
Author: Ryan Eagan <eaganry@gmail.com>
Date:   Tue Dec 15 15:13:45 2015 -0500

    multiargs test case added
    failed on first run

commit 1848e2ead3593b3b468175795170d2b5af4d0b5e
Merge: 036ce22 3675948
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Tue Dec 15 15:10:28 2015 -0500

    Merge

commit 036ce22738a88cc707c97a664b0f2aa8d96938ae
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Tue Dec 15 15:09:11 2015 -0500

    Support array indexing by expr.

commit d28b198ac3bba99ce8f770fcee45f80d84dfc95d
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Tue Dec 15 15:08:47 2015 -0500

    Bug fixes.

commit 607285a570f565091bb98bbb3066a08fe642e7e4
Author: Mike Goldin <mag2277@columbia.edu>
```

```
Date:   Tue Dec 15 14:52:26 2015 -0500

    Bug fix in function body layout and implement array index by expr

commit 3675948f8b2bb1c1bba0a1b984fa6fa2ffdc68b3
Author: Ryan Eagan <eaganry@gmail.com>
Date:   Tue Dec 15 14:50:04 2015 -0500

    funcreturn and fib test cases added
    both failing. not sure about return vs RETURN though

commit f3a0828b8518c9b6e353f0654bf53a102e4ad765
Author: Ryan Eagan <eaganry@gmail.com>
Date:   Tue Dec 15 14:32:27 2015 -0500

    orderops test case added
    failed

commit b20214cac4769a61b169f920a427900121f5905a
Author: Ryan Eagan <eaganry@gmail.com>
Date:   Tue Dec 15 14:29:29 2015 -0500

    if test case added line breaks

commit 4c27a024f2a21107c002eb7e270f03228f035df4
Author: Ryan Eagan <eaganry@gmail.com>
Date:   Tue Dec 15 14:24:45 2015 -0500

    Added if else test case
    changed while to WHILE in arrayops test

commit aacdf1946fee40ea783e76030e098c00ce02189a
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Tue Dec 15 14:20:44 2015 -0500

    Support assignment to array elements.

commit 337e61b1decdd114d5565b353a66df7ba4987e41
Merge: e22643b 620e12d
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Tue Dec 15 14:07:37 2015 -0500

    Merge branch 'dev' of https://github.com/skmgoldin/arg into dev

commit e22643bd108a8b8e2b457677a716bf44535fa22a
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Tue Dec 15 14:07:33 2015 -0500

    Notes.
```

```
commit 620e12d5749d5e18bef115a239f3c60519bf9c16
Author: Ryan Eagan <eaganry@gmail.com>
Date:   Tue Dec 15 14:03:55 2015 -0500

    added colors to PASS and FAIL

commit 85316b4cc443852361cc301569fd4b306f73a33b
Author: Ryan Eagan <eaganry@gmail.com>
Date:   Tue Dec 15 13:51:50 2015 -0500

    arrayops test added, fails on first look

commit 94d79b82d8134466a494e8f0031207ebf68193f8
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Tue Dec 15 13:42:37 2015 -0500

    Bug fix.

commit e64c4fe9a211d2230ef2f8d91c2067ada16a58b5
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Tue Dec 15 13:41:59 2015 -0500

    Build up function names persistently in symbol table.

commit 2b59c80adf1a3e50e2b22e40d3026a868c7287e8
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Tue Dec 15 13:37:27 2015 -0500

    Fix parser bug, properly order statements in WHILE scope.

commit 563c4462d32f152946b5b3c50a040329f2332a91
Author: Mike Goldin <skmgoldin@gmail.com>
Date:   Tue Dec 15 13:32:24 2015 -0500

    Parse function formals as variables in function scope.

commit 7270a2ad4ed5a7e9360870328af6bf5cbaf56e01
Author: Ryan Eagan <eaganry@gmail.com>
Date:   Tue Dec 15 18:29:09 2015 +0000

    test.py updated for debian

commit 77651b53a59c42807e907830d986c5db8e27aa11
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Tue Dec 15 13:19:34 2015 -0500

    Better error messages
```

```
commit 2dc3138ea54a9d8725dc9c4881f121e1f7ca894b
Author: Mike Goldin <skmgoldin@gmail.com>
Date:   Tue Dec 15 13:12:54 2015 -0500

    Bug fix.

commit d77da223a4dbca4e38d4bcc734b5efa9e2e9ca1d
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Tue Dec 15 13:10:01 2015 -0500

    Silence GCC warnings.

commit c4e1260a6a3accf5472603a0c0086a15845a1d02
Author: Ryan Eagan <eaganry@gmail.com>
Date:   Fri Dec 11 13:23:17 2015 -0500

    test script shows line by line errors, shows arg compile errors

commit c7f24c2e6ef3cc0d340f84b693c21ef44248ab76
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Fri Dec 11 11:26:43 2015 -0500

    Bug fix. Load array elements in proper order.

commit 2f3258a53e2ab8ad778d59ddceb1a96f369c53f6
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Fri Dec 11 11:24:33 2015 -0500

    Implement array indexing.

commit 4636c8dfc6303cc6252d3367e11c340cbf3af0c0
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Wed Dec 9 17:25:17 2015 -0500

    Initial commit.

commit 568a781844aacec3e83a382668b27cf9377a5a9e
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Wed Dec 9 17:10:39 2015 -0500

    Updated comments.

commit 401e1bacd8235870ae64eef4fc1690e07c27fd00
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Wed Dec 9 17:02:43 2015 -0500

    Bug fix.

commit 18026cc62749187d15743722417acb7dee299b05
```

```
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Wed Dec 9 16:53:48 2015 -0500

    Symbol checking, jump table construction and code generation now happen
in a single scan-through of the source code. Bug fixes.

commit f62b09cec059db48518413bf653250f131caea74
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Wed Dec 9 15:36:05 2015 -0500

    Implement if and if-else.

commit 957eae76d3f98940d56640975046284bfebc4118
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Wed Dec 9 15:25:12 2015 -0500

    Implement while loops.

commit 23213575df880a03e1663cbab71c6c334071bac3
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Wed Dec 9 15:09:14 2015 -0500

    Stub-in for jump table implementation through statement level.

commit 28f549214a41f2142c6a3f1f7df85601d7f210e8
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Wed Dec 9 12:49:32 2015 -0500

    Comments.

commit 19376f775a5c8d171b4d72fc75b05559d43f8a35
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Wed Dec 9 12:47:31 2015 -0500

    Comment.

commit 1e79a8bcbb2579979b9d45a634aafb489daa5ddc
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Wed Dec 9 12:43:29 2015 -0500

    Array test program.

commit b3b375c59b2a46713ae5a25ced9c5cc6f140a477
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Wed Dec 9 12:43:19 2015 -0500

    Model for array test.

commit b39131691139167f6bc355eedefeef3ca84a8e4d
```

```
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Tue Dec 8 17:40:20 2015 -0500

    Implement array freeing.

commit 89a54f3c5683587f1ff3b0e6598942d57c012ea9
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Tue Dec 8 17:21:44 2015 -0500

    Dissallow calling of variables as functions.

commit 32a1d7f2f9774855b574270af8402044ce3f940f
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Tue Dec 8 17:16:13 2015 -0500

    Disallow assignment to in-use function names.

commit ec0021a09696620231b3c2a17235c8edbafe4919
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Tue Dec 8 16:56:32 2015 -0500

    Dead code.

commit 0e18fe8e2939ca7c55325f562d95b6bedf0f0ab2
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Tue Dec 8 15:05:09 2015 -0500

    Comments

commit 9a877f0ec9e9aeb305a10ba5463b3a4b07a6bc9c
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Tue Dec 8 15:01:37 2015 -0500

    Check if functions exist if they are called.

commit 185b67ae9b03cb32d81e8cdae9cfeb9d1d824e26
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Tue Dec 8 14:59:59 2015 -0500

    Bug fix for funcs with no args

commit 54f4306194304bb8c67e5396e0215fdeb03ae5bc
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Tue Dec 8 14:57:32 2015 -0500

    Fix nasty parser bug improperly re-ordering statements and funcs.

commit 28ef6cc3f6ae593a98c4e6c4b51fedb005514ab0
Author: Mike Goldin <mag2277@columbia.edu>
```

```
Date:   Tue Dec 8 14:20:37 2015 -0500

    Bug fix for functions with no arguments.

commit bcc2d8245da6577c5d25a96ae5873c0c3cae02ec
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Tue Dec 8 14:13:14 2015 -0500

    Check if function names have been declared.

commit df19655b95561654febb3ed73ce24e1f1849ea15
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Tue Dec 8 13:53:51 2015 -0500

    Implement scope checking for body.

commit 335b8dbfe910d9ed9f426017924cefff62bb3f7d
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Tue Dec 8 13:17:09 2015 -0500

    Implement symtable collision checking for function arguments

commit c22bd9dc91bfbb286541c456efe8264dbe5d93d0
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Mon Dec 7 15:54:18 2015 -0500

    WIP symtable implementation.

commit 6ee034ded691e8a926b02cc3d717a0567f3da09b
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Mon Dec 7 14:51:18 2015 -0500

    Fix ARG syntax

commit f590566083004b257474ccee649d71e302bf0768
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Mon Dec 7 14:25:49 2015 -0500

    Remove Noexprs

commit 69fabdac600c406d13900a0d8a105a0ef47b9024
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Mon Dec 7 14:25:38 2015 -0500

    Fix SR conflict

commit c9af8fad408765bee877ebe07677e61b3ed8a13c
Merge: b0b23b0 5730d18
Author: Mike Goldin <mag2277@columbia.edu>
```

```
Date:   Sun Dec 6 16:40:43 2015 -0500

    Repair merge conflicts

commit 5730d18f0644ff408187704ae9f8f13378c6c4b2
Author: Ryan Eagan <eaganry@gmail.com>
Date:   Sun Dec 6 15:35:42 2015 -0500

    print string changed from %s to %x

commit eeb09aee85b34346d9c58f4d5369458b46b8efb4
Author: Ryan Eagan <eaganry@gmail.com>
Date:   Sun Dec 6 15:28:42 2015 -0500

    Test script finds test names based on files

commit 20f58f13fce1442663cc0a02b3d95d44b5c1d615
Author: Ryan Eagan <eaganry@gmail.com>
Date:   Sun Dec 6 15:01:45 2015 -0500

    added return to set_array_element, not necessarily correct...

commit e3e1f712b76d477134dce68692d7c2d1ff55425e
Author: Ryan Eagan <eaganry@gmail.com>
Date:   Sun Dec 6 14:59:54 2015 -0500

    Changed boolean print from %s to %d

commit b0b23b0101fded896a826d1a47123b42b9978456
Author: River Keefer <rdk2123@columbia.edu>
Date:   Sat Dec 5 16:13:41 2015 -0500

    added parentheses to parser -- noticed that there are shift reduce
conflicts

commit b9c51accf110da1d9bd341bcb3b57d2cc01de38a
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Sat Dec 5 15:50:20 2015 -0500

    Fix bug printing bools.

commit 0c73bdc749764ecd7017037e092502d16c329034
Author: River Keefer <rdk2123@columbia.edu>
Date:   Sat Dec 5 15:31:32 2015 -0500

    parser tokens for binops

commit 41b518fe5ece9ae5792b1eb581e6b96f387aa881
Merge: d2b6846 7c5a944
```

```
Author: River Keefer <rdk2123@columbia.edu>
Date:   Sat Dec 5 15:28:07 2015 -0500

    Merge branch 'dev' of https://github.com/skmgoldin/arg into dev

commit d2b6846da96545c6dc27ae4ede0cb999467e1f01
Author: River Keefer <rdk2123@columbia.edu>
Date:   Sat Dec 5 15:28:04 2015 -0500

    added binops to parser

commit 7c5a944dc81c0fb75ad836534bb1f3a880e0b459
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Sat Dec 5 15:27:46 2015 -0500

    Equality operators.

commit efc65999fff2211e28e888688cfcd01de3bbcd1b
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Sat Dec 5 15:24:21 2015 -0500

    Scan arithmetic operators.

commit 6beaf94c658cceb86b3a455ff93e0a11cf0761e5
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Sat Dec 5 15:21:51 2015 -0500

    PROPOSAL ON THE SYMTABLE

commit 148998e15f7d66b793eb904124e9984fc68ed0d5
Author: River Keefer <rdk2123@columbia.edu>
Date:   Sat Dec 5 15:21:13 2015 -0500

    test.py change for binops

commit a09e85fd1b6f4905628cd73ffa412caca5265d10
Author: River Keefer <rdk2123@columbia.edu>
Date:   Sat Dec 5 15:18:38 2015 -0500

    added binops to argc.ml

commit cf16b6089034c47546425500959e13f056da0d5c
Author: River Keefer <rdk2123@columbia.edu>
Date:   Sat Dec 5 15:07:04 2015 -0500

    super simple monotype binop testing

commit bfbb642ceb70864f2e6202cb5c6d2817db466c4b
Author: River Keefer <rdk2123@columbia.edu>
```

```
Date:   Sat Dec 5 15:03:59 2015 -0500

    added c library for monotype binops

commit 2a7553f34f159a02ae9a1d84d62afd2b306ed69b
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Sat Dec 5 13:39:16 2015 -0500

    Comments.

commit a00e08926ead2e8e7f5b469a941abc6e67fbff12
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Sat Dec 5 13:34:59 2015 -0500

    Comments

commit fe1fafd04d78d952c9cfedda588079f37bd72911
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Sat Dec 5 13:30:40 2015 -0500

    Update comment.

commit da4e32d80bdf9c682d80a1b29434b3f0b638a708
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Sat Dec 5 13:24:40 2015 -0500

    Test binops.

commit 816e3171332cf1eef6af22e196d1211331b55714
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Sat Dec 5 13:17:09 2015 -0500

    Parity with ARG correspondent.

commit 1c17857cf49bf2429607e19cf0558366e41ee71c
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Sat Dec 5 13:11:02 2015 -0500

    Bug fix. C arg lists need to include the argument type.

commit c6d7948b674d3809443efff2a96ebbe359f37e06
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Sat Dec 5 13:08:54 2015 -0500

    Bug fix. Append semis at a higher level.

commit 28922d273421cf374001c69e417c098ca1ff295c
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Sat Dec 5 13:05:23 2015 -0500
```

```
    Bug fix.

commit 4cdda66307a89e1cbf6ca5914e5835d552bb88be
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Sat Dec 5 13:04:56 2015 -0500

    Implement arg call to C call translation.

commit 4baa51c4b71a176ca4f584763db59d92046eabcd
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Sat Dec 5 12:57:03 2015 -0500

    Fix bug appending extra commas to argument lists.

commit d7e91c786ce0af449d12f57bf677909b4f7dfd2e
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Sat Dec 5 12:47:23 2015 -0500

    Update to bring in line with latest ARG spec.

commit b5e720d9c81a8b3cc4767a0be2396e189dd5ee2f
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Fri Dec 4 20:01:18 2015 -0500

    Implement compiling of PRINT statement, refactor function
new_monotype_of_expr to monotype_of_expr

commit 06aeb435d9d093eec2b19f02242d33d445f9de5e
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Fri Dec 4 20:00:39 2015 -0500

    Scan and parse the PRINT token.

commit 3265dd6b1543786614e5923fa1ec1fea9c93dd83
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Fri Dec 4 18:46:47 2015 -0500

    Reformat using four-space tabs.

commit 2d7c3880c7ee81207486dfe9fb111f6bae744109
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Fri Dec 4 18:44:36 2015 -0500

    Comment.

commit bdb7fa5f8c9bec7783a0ec6c90e07e4e0780e3d2
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Fri Dec 4 18:42:21 2015 -0500
```

```
    Remove death zone, add note.

commit be978980620ca11ddd7feed175f043839b56f7b0
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Fri Dec 4 18:40:21 2015 -0500

    Update test case for helloworld

commit 7aa2465deb219df2dff6f42e4f39c623885fad27
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Fri Dec 4 18:38:57 2015 -0500

    Move monotype into tests, but THIS IS BAD, see note in argc

commit 9fd448206ab18f9f93996eae091f4ea9673e09fe
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Fri Dec 4 18:38:25 2015 -0500

    Add note.

commit 25da3b01edc12c95987aed4b8b3168a36c76cf41
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Fri Dec 4 18:34:38 2015 -0500

    Use new compiler name, argc

commit 21cffeddc6cb1a2b9a0a43a944739c899df0ff71
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Fri Dec 4 18:31:45 2015 -0500

    Fix Makefile, rename arg to argc.

commit d61c1fa6ea95b0bcf7177e2ae5668c0b0e7b1ba0
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Fri Dec 4 18:23:58 2015 -0500

    Untrack binary.

commit a779643368ee1e24edafd33536c2394c62f6e36c
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Fri Dec 4 18:14:09 2015 -0500

    Fix shift-reduce conflicts.

commit 69edd7984d8e2ab7fd9575405fce18dee930ab0f
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Fri Dec 4 18:06:35 2015 -0500
```

```
    Remove accidentally integrated test from helloworld.arg

commit 805ef8c49dacf8bb4554dee086d842596395be37
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Fri Dec 4 18:02:16 2015 -0500

    Added some junk.

commit c7e5b1e62071ee2cfa24e93d209ffe4129bd8fb0
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Fri Dec 4 18:01:59 2015 -0500

    Added code to test array creation.

commit 386a9147039b32b6c33f38149909f851141aed93
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Fri Dec 4 18:01:00 2015 -0500

    Bug fix.

commit 953553dcdb6ba5f17b4200f10c3172055f1635d0
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Fri Dec 4 18:00:52 2015 -0500

    Implement creation of new monotypes with heap array storage.

commit f259a463103dc163b2641689e85aa3c675c80359
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Fri Dec 4 16:26:34 2015 -0500

    Get useful debugging output from OCamlyacc in the file called
parser.output.

commit 2c9de6e96879d38b6e93d17ec636b4cbc1dd3e3c
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Fri Dec 4 16:24:20 2015 -0500

    Updated for monotype floats.

commit ae3f07a6921ce03945fc2354d76eaa3e49a681ef
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Fri Dec 4 16:00:23 2015 -0500

    Generate monotypes for ints, bools and floats.

commit 268cc99924d4e14ce3377679f7cc6eba54aebf09
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Fri Dec 4 15:55:50 2015 -0500
```

```
    Use floats instead of double, since OCaml uses floats.

commit cf9e7fd5465b20977e0f8348d1837f53167468f7
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Fri Dec 4 15:51:58 2015 -0500

    String literal assignments.

commit 61d412e0b89351830f0b420d1b6dd82bb4e4c14a
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Fri Dec 4 15:51:42 2015 -0500

    Fix compile bugs.

commit 5ee2c133b65f119f6e3992ddb51f3f5e0b2bc1f0
Author: Ryan Eagan <eaganry@gmail.com>
Date:   Fri Dec 4 16:57:47 2015 -0500

    test script v1 working

commit 1bab0403c139a12d6ee3e824f7b78cdf25e97f21
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Fri Dec 4 15:22:15 2015 -0500

    Comments.

commit b1a3925c8883991c1c6a31871ffe52f509707df0
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Fri Dec 4 15:17:00 2015 -0500

    Formatting improvement for generated code.

commit 5d4092db6768eeae4e1fb7637b2f5afe32a71c72
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Fri Dec 4 15:16:16 2015 -0500

    Process the entire list of statements in the body.

commit a47948a7bf89547a43947e7f1d32ccaf3d5488c4
Merge: 68037df 9f75114
Author: River Keefer <rdk2123@columbia.edu>
Date:   Fri Dec 4 15:14:18 2015 -0500

    Merge branch 'dev' of https://github.com/skmgoldin/arg into dev

commit 68037dfd74ff6739708f1ca66d800d25a57e6e98
Author: River Keefer <rdk2123@columbia.edu>
Date:   Fri Dec 4 15:14:07 2015 -0500
```

```
    added more literals to scanner / parser / ast

commit 9f751145114cbf83937d2bd93b64fed39160ba7c
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Fri Dec 4 15:12:12 2015 -0500

    Malloc arrays in the monotype.

commit d2278e57e3c4425f7c228abe2799ad23db10c2c5
Author: River Keefer <rdk2123@columbia.edu>
Date:   Fri Dec 4 14:47:47 2015 -0500

    ast fix for arrays

commit 3cf01d3fa1cdc83c49a3ce5953c72853f5a9b2aa
Author: River Keefer <rdk2123@columbia.edu>
Date:   Fri Dec 4 14:39:37 2015 -0500

    ast changes for arrays in progress

commit a07b2e93a529188f6d872ce8373946d6a32b729a
Author: River Keefer <rdk2123@columbia.edu>
Date:   Fri Dec 4 14:17:07 2015 -0500

    more array stuff -- example of constructed c code in test.c

commit 414cdb5d55b38dfa972161a0c418fd5d9995bff3
Author: River Keefer <rdk2123@columbia.edu>
Date:   Fri Dec 4 13:42:36 2015 -0500

    added arrays (unfinished) to monotype

commit 3c415844d29ed06ade7115abe95dc3f221d7ad69
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Fri Dec 4 12:29:09 2015 -0500

    Support parsing, scanning and stubbed compiling of IF and IF ELSE
statements.

commit 7191ded5ffc154cd1d5889216d75b7919cfc0707
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Fri Dec 4 12:19:19 2015 -0500

    Support statement lists everywhere.

commit 011a6a4c301b9818c249bfdb5ee09218dbff3621
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Fri Dec 4 12:13:50 2015 -0500
```

```
    Scan and parser IF statements.

commit 36831ee650a1d945c604c0b8442989de2300b1ac
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Fri Dec 4 12:10:04 2015 -0500

    add while to scanner.

commit ba4ab63fa2c0c14e9c68b35b1bd920afbc86a9f2
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Thu Dec 3 17:44:15 2015 -0500

    Fixed BUF 0

commit f11340a4268e16b1453e28159d1b33c82fb40f22
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Thu Dec 3 17:20:49 2015 -0500

    Comment update

commit b73fd88cfd538f131101bfd8f0adc6d730429e3f
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Thu Dec 3 17:11:15 2015 -0500

    Update some comments.

commit 37b51c35ced8b5de44a94d0545bc3276e3c53364
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Thu Dec 3 17:01:20 2015 -0500

    Comment.

commit 558eaa46c64d7672d7f66006ea153ce42a9e27d5
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Thu Dec 3 16:58:51 2015 -0500

    Improve BUG 0 code.

commit 8d031b30b8e6e0b4cc3dab40614bcc7d3aa67da1
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Thu Dec 3 16:51:54 2015 -0500

    Improved string buildup for While pattern in BUG 0

commit d35255cb8c49900e0fd2612a233024b9b67e44c3
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Thu Dec 3 16:36:14 2015 -0500
```

```
    Bug fixes. HACKY stub-in of arg_expr_to_c_expr rolled in to
arg_stmt_to_c_stmt. See comments.

commit 66457619c89ca43881abdc71d356067459869bca
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Thu Dec 3 16:00:28 2015 -0500

    Stub in arg_expr_to_c_expr

commit 1690d87df4fa8c84332742ac235dcd29e48c2ab2
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Thu Dec 3 15:57:18 2015 -0500

    Implement arg_stmt_to_c_stmt

commit 82e28eeb4afe4e252b5b3323ed1963842de2279b
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Thu Dec 3 15:45:34 2015 -0500

    Merge in squashed commit.

commit e6a079da81c1c0167fc95f1cb680480f29d4f5fe
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Thu Dec 3 15:40:20 2015 -0500

    Complete arg_func_to_c_func.

commit 7836b03343b439327e72777597f4f8cd2248dc76
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Thu Dec 3 15:32:55 2015 -0500

    Partial implementation of func translator, stub for body translator,
expand death zone for clarity.

commit 93f43041cfd2969b8cc84178aa9f16cfef4cd69d
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Thu Dec 3 15:03:15 2015 -0500

    Refactor translate_program

commit 190e87b1aaf64b37c2cd74fa3722ecd3a224f5c8
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Thu Dec 3 14:59:51 2015 -0500

    Reorg, mark old code for death.

commit 97ba6c94a5c89be50dfbbc7fd05e00a6d51e6141
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Thu Dec 3 14:42:38 2015 -0500
```

```
    Add comments, delete dead code.

commit 5a11fad069ea50f963ecf398dc527299378dbe2b
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Thu Dec 3 14:36:59 2015 -0500

    Out with oCaml-case, in with underscores.

commit 10347bd6149aaeb2bcd51201cba357d967fad055
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Thu Dec 3 14:34:00 2015 -0500

    Uncomment that ish.

commit ae8b5d17be0cb37dc938c45ed446be846c90a5a0
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Thu Dec 3 13:36:33 2015 -0500

    Fix compile errors.

commit 7f2a91f44c48bd7448d4c3bc8aca94edfb74b25f
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Thu Dec 3 13:34:40 2015 -0500

    Define function action and fix compile bugs.

commit 47c57694e4edb52da3786624a160fb6427106b80
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Thu Dec 3 13:34:01 2015 -0500

    Fix compile error.

commit 46f4c23900cf5f1170599b015ef3eb11bfe54d88
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Thu Dec 3 13:24:14 2015 -0500

    We are not MicroC\!

commit be8467f9a51f50777e560d2a9c21a5e0d787aa97
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Thu Dec 3 13:22:25 2015 -0500

    Redefine funcs, add tokens for braces

commit e0bed22b2e864f5a6394bf99459f53ca21722c28
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Thu Dec 3 13:22:06 2015 -0500
```

```
    Scan braces

commit 6065a2ea93cc0d7fdf19275cbb2d609d4f84e0fc
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Thu Dec 3 13:14:42 2015 -0500

    Fix compile error.

commit eebb475219daedafa352b1a2fa70db4383436720
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Thu Dec 3 13:05:58 2015 -0500

    Define an ARG program as a list of functions and a list of statements.

commit 354cbb8307845c125cb6f72122492e6b8fe4751a
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Thu Dec 3 12:58:52 2015 -0500

    Define functions in MicroC style.

commit cbe703e6346016af6d4d1447a456618b60c101b1
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Thu Dec 3 12:56:14 2015 -0500

    Fix namespace collision on 'function'

commit ebdc7d6d27ea9d503a1e09b7a8c3413220deab82
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Thu Dec 3 12:52:16 2015 -0500

    Define WHILE token.

commit e5d5ee5ce959a4e506837282fbb72e37cf879797
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Thu Dec 3 12:50:05 2015 -0500

    Define pattern and action for while loops.

commit 888d1faf29ad5cfc3b3d6dea12cb3dba14eecc42
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Thu Dec 3 12:36:51 2015 -0500

    Define loops in line with MicroC style.

commit de0405ab83a90bdeea74ec9d894f09e990ddb892
Merge: 2c33b42 cd8278d
Author: River Keefer <rdk2123@columbia.edu>
Date:   Mon Nov 30 15:52:46 2015 -0500
```

```
    Merge branch 'dev' of https://github.com/skmgoldin/arg into dev

commit 2c33b42a82c342b43f1f9474117711d924b399d7
Author: River Keefer <rdk2123@columbia.edu>
Date:   Mon Nov 30 15:52:44 2015 -0500

    small parser / scanner changes

commit cd8278deb45f9713d6c4cb4a21777d9185d73fa1
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Mon Nov 30 15:51:39 2015 -0500

    Delete dead code.

commit 377419c1e653791abedbcf5ae18db4f5a6445b7c
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Mon Nov 30 15:49:53 2015 -0500

    Cleanup

commit b82d269ea1221aa2f6ccb870a27a0a17eee27756
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Mon Nov 30 15:49:40 2015 -0500

    WIP. Refactoring.

commit e8bbbe48ba4735ef1d0944e41282fcfef18207b8
Merge: 5e531d5 9537920
Author: River Keefer <rdk2123@columbia.edu>
Date:   Mon Nov 30 15:49:05 2015 -0500

    Merge branch 'dev' of https://github.com/skmgoldin/arg into dev

commit 5e531d55e59754c2030bb21d757c788c48bd1eba
Author: River Keefer <rdk2123@columbia.edu>
Date:   Mon Nov 30 15:48:54 2015 -0500

    mess of parser / scanner changes

commit 9537920eaebc51429ec92b30e87abc2b2cb15b10
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Mon Nov 30 14:44:06 2015 -0500

    Example generated C code

commit a1075e409fc2cc6ce3b247d96209efab9e170e17
Merge: 23be8b0 69c91a8
Author: River Keefer <rdk2123@columbia.edu>
Date:   Mon Nov 30 14:33:36 2015 -0500
```

```
    Merge branch 'dev' of https://github.com/skmgoldin/arg into dev

commit 23be8b02c5fe27a4c434394fd47c098b2c51e0c3
Author: River Keefer <rdk2123@columbia.edu>
Date:   Mon Nov 30 14:33:21 2015 -0500

    function name changes

commit 69c91a801988ac7fae76ff86bdfd8c15d384005f
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Mon Nov 30 14:33:13 2015 -0500

    Updated monotype code and example.

commit 34f700e49f63ff1c742a69a8d3747ece3eb83b2f
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Mon Nov 30 13:46:42 2015 -0500

    Example arg->C conversion.

commit 04f19b7f015e6783efd987355789358e30a43db8
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Mon Nov 30 13:46:31 2015 -0500

    WIP ADT of monotype

commit 859d0642314b6ac27eda4ecb78aec83e56a27903
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Mon Nov 30 12:23:29 2015 -0500

    Redo monotype constructor flags

commit 44c316988ff1c5d4579669e8fb502594cc51d517
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Sun Nov 29 18:40:25 2015 -0500

    More monotype C code.

commit 2821916e6ba0582d1a0cc968656b2ab99872873d
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Wed Nov 25 14:47:50 2015 -0500

    Decide on C declaration prefixes.

commit aa368d8f2415b873d1c18c425f4468cde971cd89
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Wed Nov 25 13:39:49 2015 -0500
```

```
    Code cleanup.

commit f6a34961b379603d224beac5e21d396e49456b90
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Wed Nov 25 13:37:24 2015 -0500

    Implement monotype for runtime type inferencing.

commit dce30e5dd3d717cc4f1c4fac1d0b4e61bd6160ef
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Sun Nov 22 18:46:54 2015 -0500

    Deconfliction.

commit 1dc60ce4bda67c13c2d9724fc2e7c8f7f5881644
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Sun Nov 22 18:39:03 2015 -0500

    Rename function

commit f2b6742e14805f2af107e36a6ffba5846589471c
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Sun Nov 22 18:02:18 2015 -0500

    Formatting, comment.

commit 698abbf2872fb2f0073ffbeb32db62c31938fbd9
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Sun Nov 22 17:48:58 2015 -0500

    More dramatic call out.

commit 60fdec57347704173f40b2dedcac089bda9e6982
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Sun Nov 22 17:45:45 2015 -0500

    WIP. Implementing symbol table. Doesn't compile.

commit 923bd23fbe7314118d327bd71615691925e55c7d
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Sun Nov 22 16:01:39 2015 -0500

    Alphabetize

commit b0d354235da4c1cc422a8321745463ebdb8e39e0
Author: River Keefer <rdk2123@columbia.edu>
Date:   Mon Nov 16 13:03:56 2015 -0500

    tiny cleanup
```

```
commit 9430f1db6833f6b7432bf11f0f034d23cdf218db
Author: River Keefer <rdk2123@columbia.edu>
Date:   Mon Nov 16 12:59:02 2015 -0500

    added simple python test system

commit 4fa97961085867e358faa8afe925d68837892522
Author: River Keefer <rdk2123@columbia.edu>
Date:   Mon Nov 16 12:44:22 2015 -0500

    fixed helloworld.arg, arg.ml outputs to file, makefile feeds c code into
GCC

commit c6214f8ed1f6bbd75d8ec8bfde4b0600741b8fb6
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Sat Nov 14 17:28:39 2015 -0500

    Formatting.

commit 89e2d5adf8894dfc847d337ec6db13dbc20e3754
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Sat Nov 14 16:38:38 2015 -0500

    Detect strings and set ID type accordingly.

commit 108088699c8ea0592a159ef4cb8edded603be239
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Sat Nov 14 16:38:22 2015 -0500

    Bug fix.

commit 50e0eb46bbd6c1cbb1bad35ab883ddc45a43af49
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Sat Nov 14 14:50:17 2015 -0500

    Genuine special case for print statements.

commit 9147811b25949598b3018bc9e7d5366a47f90402
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Sat Nov 14 12:03:36 2015 -0500

    Properly print function call arguments.

commit d0b8aa15b9dcece29e397e3a31f791be99ec0f41
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Sat Nov 14 10:20:34 2015 -0500

    Bug fix.
```

```
commit 0a09080680db6adf37ad245d7996acc9f7ec02e4
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Sat Nov 14 09:56:54 2015 -0500

    Formatting.

commit 5f69ab5469fdcf280a1b5dbbe35a6203426ecbe3
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Sat Nov 14 09:56:31 2015 -0500

    Wrong, but a step closer toward translating function calls.

commit 430d1c4a4c4787f1e16849a08caa9895d8fbfb5d
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Sat Nov 14 08:53:11 2015 -0500

    Formatting.

commit f9876cffd61ae2a0c1e1ef8f57a938d790a6732e
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Sat Nov 14 08:52:26 2015 -0500

    Order statements properly.

commit 8208f7471563c26ebe3f1684bf76b1f0e51d6c03
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Sat Nov 14 08:51:56 2015 -0500

    Formatting.

commit cb17fa48f5d7b87e4a83b29435ce632675aa6d4e
Author: River Keefer <rdk2123@columbia.edu>
Date:   Fri Nov 13 17:33:27 2015 -0500

    it compiles, but in reverse order, print doesnt work yet

commit 93cedce15c1df71d1991dd60b33e46b66278b5d6
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Fri Nov 13 17:28:41 2015 -0500

    Bug fix for strings.

commit 8b4575ea1f25bbcde7c80d581cf9733d25d22d57
Author: River Keefer <rdk2123@columbia.edu>
Date:   Fri Nov 13 17:16:30 2015 -0500

    translator that doenst translate compiles!
```

```
commit e3742b3a92577ee5fb030b093d7a589751991028
Merge: 8f6c29e 143f37d
Author: River Keefer <rdk2123@columbia.edu>
Date:   Fri Nov 13 17:00:03 2015 -0500

    merge fix

commit 8f6c29eed28fa52982157d1a1959861793039315
Author: River Keefer <rdk2123@columbia.edu>
Date:   Fri Nov 13 16:58:12 2015 -0500

    re assembly of arg code beginning

commit 6aff84e6cc91670ba142271b54e4d8430a38f65b
Author: River Keefer <rdk2123@columbia.edu>
Date:   Fri Nov 13 16:27:08 2015 -0500

    stuff

commit 3aa6c220ecaa08950677b43aa30fba5fceac62f4
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Fri Nov 13 16:36:28 2015 -0500

    Make ID's expressions.

commit 143f37d762c44168affdc9d5c6135170015439cb
Author: River Keefer <rdk2123@columbia.edu>
Date:   Fri Nov 13 16:27:08 2015 -0500

    stuff

commit d9c8a1201bce862d652558b2960a4dc797ed8884
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Fri Nov 13 16:24:22 2015 -0500

    Bug fix

commit bb2036b9f62d23d7eb1933ddcf97016dbe49dac0
Author: River Keefer <rdk2123@columbia.edu>
Date:   Fri Nov 13 16:21:58 2015 -0500

    fixed semi bug

commit 7dd5f7ab3352a305e220d5c23efae59e5e603a89
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Fri Nov 13 16:12:03 2015 -0500

    Require semis
```

```
commit c6fe617881245c10a4ac5dddf304f61ca8c10c8e
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Fri Nov 13 16:09:26 2015 -0500

    Bug fixes.

commit d47c006b4e54355b620e9eded6a8b9b9752a38b9
Merge: 927628a ace628e
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Fri Nov 13 16:06:13 2015 -0500

    Merge branch 'river-test' into argmin

commit ace628e21c9ee0caac609a07d3232b4072ebfd60
Author: River Keefer <rdk2123@columbia.edu>
Date:   Fri Nov 13 16:05:45 2015 -0500

    actuals_opt

commit 927628aa9f410efecb42fc16d39ae52420425f69
Merge: bb27f12 4e7fd27
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Fri Nov 13 16:01:27 2015 -0500

    Merge branch 'river-test' into argmin

    Conflicts:
      parser.mly
      scanner.mll

commit 4e7fd27f705fb583a7e94ba942dedefac4e32b67
Author: River Keefer <rdk2123@columbia.edu>
Date:   Fri Nov 13 15:55:34 2015 -0500

    heloooo

commit bb27f12d9e2f1076b2af9a7a24d83099d6ae4e85
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Fri Nov 13 14:57:23 2015 -0500

    Bug fix

commit 8fe2b02e65394170cf119bc072d241a3104fa954
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Fri Nov 13 14:54:06 2015 -0500

    Change var to id

commit d539d7896782f56be7b8e918fb10953c1ff12be3
```

```
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Fri Nov 13 12:39:02 2015 -0500

    Remove print and parens

commit 828b9580390d891716658d5744d15e5d5c460f78
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Fri Nov 13 12:36:59 2015 -0500

    Remove print.

commit 5b405a3f3f9ef9850cd363a437a46360e145cef4
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Fri Nov 13 12:35:47 2015 -0500

    Remove print stuff.

commit 51d17a6dbf11cf8007297996c302f96f255ee29b
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Fri Nov 13 12:33:34 2015 -0500

    Bug fix

commit 992957e1eac3097ea021b5d16746ea09786e1460
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Fri Nov 13 12:28:41 2015 -0500

    Formatting.

commit 4ab3b1ba0adbe3eb667ec39db50a3288825b4a6e
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Fri Nov 13 12:27:49 2015 -0500

    Barebones support for variable lists.

commit 09d7e99466d184a462a9313e8a5c21c9d17b1b5b
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Fri Nov 13 11:32:32 2015 -0500

    Remove unneccesary tokens

commit b8c8102ba669eb2223b536ddf73e6285e8c97f71
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Fri Nov 13 11:32:10 2015 -0500

    Simplify

commit c5170691486d7ee5b7306bb83f35b0d46fef642d
Author: Mike Goldin <mag2277@columbia.edu>
```

```
Date:   Thu Nov 5 17:50:44 2015 -0500

    Accept print statements

commit d121d8660c5c8abcbf29e83489e5a63da837305c
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Thu Nov 5 17:46:28 2015 -0500

    Strip out almost everything, accept only empty variable declarations.

commit 8f747b64cb8d6a0abb387c4b2423dea3bc43d6a2
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Thu Nov 5 16:52:56 2015 -0500

    Strip out all tokens not necessary for printing an integer.

commit 8bf253a241d239343709db93e39dc925b3947425
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Thu Nov 5 14:39:25 2015 -0500

    Remove FOR, add constructors for new tokens in scanner.

commit e7cd0acca69c88e18ca02d83ad440da810c8824b
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Thu Nov 5 14:29:54 2015 -0500

    Add new tokens specified in LRM, remove FOR.

commit 0a5e16e89452b9adaa9bc2dc1d5fdb2f18287e94
Author: Ryan Eagan <eaganry@gmail.com>
Date:   Mon Oct 26 17:22:12 2015 -0400

    Added casting
    removed for loop parsing

commit 224b3546e6deeaa02b8bb1adca9be0cf34f22ec3
Author: Ryan Eagan <eaganry@gmail.com>
Date:   Mon Oct 26 02:54:46 2015 -0400

    Added SEMIs to vdecl ASSIGNs

commit 8a3d1fccc49fc6ca83689df869ee6a2910b2310e
Author: Ryan Eagan <eaganry@gmail.com>
Date:   Mon Oct 26 02:19:08 2015 -0400

    Fixed formatting error, extraneous '|'s

commit a6f4b13a972b94ebbd06f2e7b509faab890eacfb
Author: Mike Goldin <mag2277@columbia.edu>
```

```
Date:   Fri Oct 23 20:52:17 2015 -0400

    Align actions

commit b3c4b046548225e384877489c44dd006f4c1150b
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Fri Oct 23 20:49:05 2015 -0400

    Fix bugs related to BoolLiterals

commit 4a7043f58f3aecfe5a06db79f76cecd0cc2a9444
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Fri Oct 23 18:55:52 2015 -0400

    Support bools and floats as expressions

commit df4a44eea1c3fe5a5a6947ccab03c85ac141561b
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Fri Oct 23 17:53:21 2015 -0400

    Float and bool support

commit 33bc3c6039f78fd69dae396e727a983f66a74d9b
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Fri Oct 23 17:46:49 2015 -0400

    Implemented (I think) scanning and parsing of STOP\!\!\! statements with
n exclamation points.

commit 7c453696c98fc93170486494a7c394848c5c6926
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Fri Oct 23 17:33:09 2015 -0400

    Reorg to better resemble microC

commit 7797852d3f65dc2f466ccf87e10b2b20947d05ad
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Fri Oct 23 17:32:40 2015 -0400

    Support STOP statement

commit c559e98a328d7d90b067137233d73689367580f9
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Fri Oct 23 17:25:25 2015 -0400

    Fixed shift conflict

commit d118464c5cc738a4468a79383dba31c2cbf485b3
Author: Mike Goldin <mag2277@columbia.edu>
```

```
Date:   Fri Oct 23 17:16:45 2015 -0400

    With one shift reduce conflict, it lexes and yaccs...

commit 53fdca90df9774193ba4c57dfe226766507dc75f
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Fri Oct 23 17:09:06 2015 -0400

    Wrong var name

commit 76b53f326a8fa6fc609eee4bc6d3b3c6d9bd78b3
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Fri Oct 23 17:08:34 2015 -0400

    More expr patterns

commit eb4c2c63709435c67da9a5d40d0f571acdea7bd5
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Fri Oct 23 17:00:01 2015 -0400

    More definitions

commit 6aa851b8a25e1e351c9d02b667284fe61c66f7bb
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Fri Oct 23 16:39:33 2015 -0400

    Get rid of stuff we don't need.

commit 32688e2ad619c7c41d6b27127bea6f7654cb121e
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Fri Oct 23 16:31:20 2015 -0400

    Support for string literals

commit f176d221f6a95a56aefde350c261d946187ffc12
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Fri Oct 23 15:05:29 2015 -0400

    Simplified and cleaned up.

commit dc0d051430999751f30413e64cecf87fa3083b0d
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Fri Oct 23 15:02:31 2015 -0400

    Updated

commit 3875ddd1ba3d92765eb309ec87caa70a334a4ab0
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Fri Oct 23 14:57:40 2015 -0400
```

```
    Bug fixes

commit 96c221cc742f98d1c641d0cb21bd22e99b62da92
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Fri Oct 23 14:46:24 2015 -0400

    Various extensions and improvements. What a shitty commit message.

commit 368ea228399b1e14157bfbd4a717650909603bc8
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Fri Oct 23 12:14:33 2015 -0400

    Added more operators.

commit 9bb9a69a51a2d3b138df4ae8a076b2683ad7538a
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Fri Oct 23 12:07:04 2015 -0400

    Bring in line with scanner.

commit 0d47688b14e06cb2e8a4351da2c02534ad55b9dd
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Fri Oct 23 12:06:48 2015 -0400

    Initial commit.

commit 650d1624e6ecf91999165afe9b31a787804a4326
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Fri Oct 23 11:56:07 2015 -0400

    Reorg, added new tokens.

commit 6a9d9bb0f520dcc1211db9afb3e2c106113c9057
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Mon Oct 19 15:30:51 2015 -0400

    More tokens

commit 7a9c75845a28893a015692ff4fc14d577f47c45d
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Mon Oct 19 14:38:07 2015 -0400

    Deprecate EQ in favor of ASSIGN

commit db4cbe981d524a1516ea2c61f713f233b6330232
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Mon Oct 19 14:32:41 2015 -0400
```

```
    Let Cmp use ocaml compare operator.

commit acf6344a8c28f9776b7bba45bc1efcb28551f50f
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Mon Oct 19 14:28:12 2015 -0400

    Removed unneccessary code.

commit edb74d3ca14991eab99d35a27e95917706fe4e0a
Author: River Keefer <rdk2123@columbia.edu>
Date:   Mon Oct 19 14:19:50 2015 -0400

    broken stuff

commit 1691792a47d0417af4080efa64a35afd2fb005fe
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Mon Oct 19 14:02:57 2015 -0400

    Support for strlits

commit 66f45ad63beb43c10ba57c58aa5498f193e96b42
Author: River Keefer <rdk2123@columbia.edu>
Date:   Mon Oct 19 13:47:04 2015 -0400

    fixed cmp

commit 8d92f9b954dc02de3f6c2eb807989ac4faac80f7
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Mon Oct 19 13:12:33 2015 -0400

    initial commit

commit b51d271ba1669d8d264cefbf2fba1cbff9b5116a
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Mon Oct 19 13:10:08 2015 -0400

    Added arg-specific lexemes.

commit e359bb232d9eec4c4ec071426bd4cce448d24f29
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Mon Oct 19 13:09:52 2015 -0400

    More stuff to scan.

commit 32cd66f8bd5930f441b4a066b80b814d177d9985
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Mon Oct 19 13:09:34 2015 -0400

    Intial commit.
```

```
commit ac049e2d6efb0a3ae2c91ef52b4b6cb2c36def50
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Mon Oct 19 12:12:20 2015 -0400

    Initial commit.

commit 5dcf204eb787a0691b6803b17b9a5b897a4fb466
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Mon Oct 19 12:10:13 2015 -0400

    Intial commit

commit f0929296fcfb61a17e0a07c56ff4f4b945fbb82e
Author: Mike Goldin <mag2277@columbia.edu>
Date:   Mon Oct 19 12:02:19 2015 -0400

    Initial commit.
```

5. Architectural Design

The fundamental concept on which ARG!'s dynamic typing is built is the notion of the "monotype", which is a data structure capable of storing all types, including multidimensional arrays of non-uniform type. The monotype includes boolean flags indicating its internal type. These flags are written at runtime during variable declaration and on assignment operations. The evaluated monotype of any expression on the righthand side of an assignment is type-checked in order to set the flags of the monotype on the left hand side. The flags of multiple monotypes are read to check for equality during binary operations. Flags are also read to produce C format strings for printing, since the ARG! programmer only needs to specify %x to format any monotype.

The monotype is implemented as a C struct.

```
struct monotype {
     int isint;
     int i;

     int ischar;
     char *s;

     int isbool;
     int b;

     int isfloat;
     float f;
```

```
        int isarray;
        struct monotype* a;
        int a_len;
};
```

A side effect of using monotypes is that the compiler symbol table does not need to store variable types, since type errors can only be detected at runtime. Arrays and strings are allocated on the heap so that they may be returned from functions. ARG! frees all in-use heap memory before program termination.

ARG!'s scoping rules are such that the program body comprises a single scope and function bodies each comprise their own scope. Neither scope type is at all aware of variables declared in other scopes. Both scopes are aware of function names: the body is aware of all function names, and function bodies have awareness of functions declared prior to themselves.

The compiler, *argc*, builds up a symbol table by first running through the program functions and cleaning out accumulated variables (but not function names) after building up a symbol table and performing scope-checking for each one. Once all the functions have been processed the symbol table will contain a number of function names, but no variable names. At this point the symbol table is sent into the body (which is a single scope) and builds up new context as it progresses through statements, declaring errors both when undeclared variables are used as well as when functions are used as variables and vice-versa. The symbol table is therefore aware not only of names, but whether those names refer to functions or variables. This allows for better feedback to the user on compile errors.

6. Test Plan & Scripts

Various small tests were implemented to check if the different features of our language work well. The tests are designed to verify different aspects of the language - from its many operators to for/while loops, as well as the major addition to the language, its dynamic type inferencing.

For each of these tests, we wrote code in ARG, as well as corresponding code in C. We then checked to see if the output for both came out to be the same. Some of the tests can be seen below:

Test 1: Checking array operations

~/Arg/arg/tests

```
File Edit Options Buffers Tools Help
a[5] = {1, 2, 3, 4, 5};

PRINT("%x\n", a[3]);
```

arrs.arg

~/Arg/arg/tests

```
File Edit Options Buffers Tools C Help
#include <stdio.h>
#include "monotype.c"


int main() {
struct monotype *aaaa = malloc(sizeof(struct monotype) * new_monotype(0, 5, 0, 0, 0, NULL, 0).i);
aaaa[0] = new_monotype(0, 1, 0, 0, 0, NULL, 0);
aaaa[1] = new_monotype(0, 2, 0, 0, 0, NULL, 0);
aaaa[2] = new_monotype(0, 3, 0, 0, 0, NULL, 0);
aaaa[3] = new_monotype(0, 4, 0, 0, 0, NULL, 0);
aaaa[4] = new_monotype(0, 5, 0, 0, 0, NULL, 0);
struct monotype a  = new_monotype(4, 0, NULL, 0, 0, aaaa, new_monotype(0, 5, 0, 0, 0, NULL, 0).i);

if(a.a[new_monotype(0, 3, 0, 0, 0, NULL, 0).i].isint) {
printf("%d\n", a.a[new_monotype(0, 3, 0, 0, 0, NULL, 0).i].i);
} else if(a.a[new_monotype(0, 3, 0, 0, 0, NULL, 0).i].ischar) {
printf("%s\n", a.a[new_monotype(0, 3, 0, 0, 0, NULL, 0).i].s);
} else if(a.a[new_monotype(0, 3, 0, 0, 0, NULL, 0).i].isbool) {
if(a.a[new_monotype(0, 3, 0, 0, 0, NULL, 0).i].b) {
printf("%s\n", "True");
} else {
printf("%s\n", "False");
}} else if(a.a[new_monotype(0, 3, 0, 0, 0, NULL, 0).i].isfloat) {
printf("%f\n", a.a[new_monotype(0, 3, 0, 0, 0, NULL, 0).i].f);
} else { printf("%s\n", "Error!"); }


if(a.isarray) { free(a.a); }
        return 0;
}
```

Generated arrs.c

~/Arg/arg/tests

```
File Edit Options Buffers Tools C Help
#include <stdio.h>

int main() {
    int a[5] = {1, 2, 3, 4, 5};

    printf("%d\n", a[3]);

    return 0;
}
```

Corresponding C code in arrs-model.c

```
Shivangi@Fido ~/Arg/arg/tests
$ gcc arrs.c

Shivangi@Fido ~/Arg/arg/tests
$ ./a.exe
4
```

```
Shivangi@Fido ~/Arg/arg/tests
$ gcc arrs-model.c

Shivangi@Fido ~/Arg/arg/tests
$ ./a.exe
4
```

Output

Test 2: Print nth number in the Fibonacci Sequence –

~/Arg/arg/tests

```
File Edit Options Buffers Tools Help
FUNCTION fib(n) {
        IF(n < 2) {
                RETURN 1;
        }
        RETURN fib(n-1) + fib(n-2);
}

PRINT("%x\n", fib(0));
PRINT("%x\n", fib(1));
PRINT("%x\n", fib(2));
PRINT("%x\n", fib(3));
PRINT("%x\n", fib(4));
PRINT("%x\n", fib(5));
```

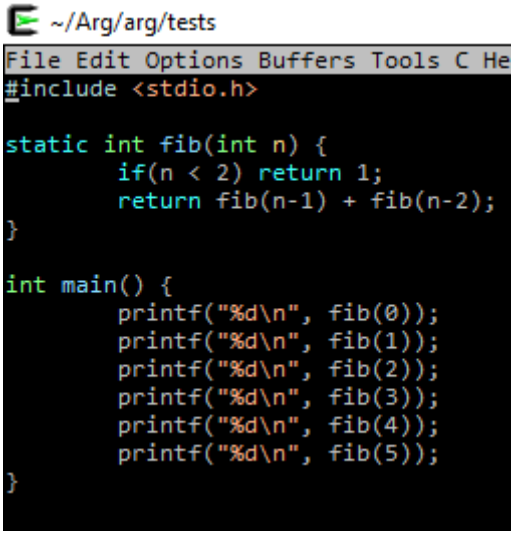
fib.arg

```
#include <stdio.h>
#include "monotype.c"
struct monotype fib(struct monotype n) {
if(monotype_less(n, new_monotype(0, 2, 0, 0, 0, NULL, 0)).b) {
return new_monotype(0, 1, 0, 0, 0, NULL, 0);
}return monotype_add(fib(monotype_sub(n, new_monotype(0, 1, 0, 0, 0, NULL,
0))), fib(monotype_sub(n, new_monotype(0, 2, 0, 0, 0, NULL, 0))));

}

int main() {
if(fib(new_monotype(0, 0, 0, 0, 0, NULL, 0)).isint) {
printf("%d\n", fib(new_monotype(0, 0, 0, 0, 0, NULL, 0)).i);
} else if(fib(new_monotype(0, 0, 0, 0, 0, NULL, 0)).ischar) {
printf("%s\n", fib(new_monotype(0, 0, 0, 0, 0, NULL, 0)).s);
} else if(fib(new_monotype(0, 0, 0, 0, 0, NULL, 0)).isbool) {
if(fib(new_monotype(0, 0, 0, 0, 0, NULL, 0)).b) {
printf("%s\n", "True");
} else {
printf("%s\n", "False");
}} else if(fib(new_monotype(0, 0, 0, 0, 0, NULL, 0)).isfloat) {
```

```
printf("%f\n", fib(new_monotype(0, 0, 0, 0, 0, NULL, 0)).f);
} else { printf("%s\n", "Error!"); }
if(fib(new_monotype(0, 1, 0, 0, 0, NULL, 0)).isint) {
printf("%d\n", fib(new_monotype(0, 1, 0, 0, 0, NULL, 0)).i);
} else if(fib(new_monotype(0, 1, 0, 0, 0, NULL, 0)).ischar) {
printf("%s\n", fib(new_monotype(0, 1, 0, 0, 0, NULL, 0)).s);
} else if(fib(new_monotype(0, 1, 0, 0, 0, NULL, 0)).isbool) {
if(fib(new_monotype(0, 1, 0, 0, 0, NULL, 0)).b) {
printf("%s\n", "True");
} else {
printf("%s\n", "False");
}} else if(fib(new_monotype(0, 1, 0, 0, 0, NULL, 0)).isfloat) {
printf("%f\n", fib(new_monotype(0, 1, 0, 0, 0, NULL, 0)).f);
} else { printf("%s\n", "Error!"); }
if(fib(new_monotype(0, 2, 0, 0, 0, NULL, 0)).isint) {
printf("%d\n", fib(new_monotype(0, 2, 0, 0, 0, NULL, 0)).i);
} else if(fib(new_monotype(0, 2, 0, 0, 0, NULL, 0)).ischar) {
printf("%s\n", fib(new_monotype(0, 2, 0, 0, 0, NULL, 0)).s);
} else if(fib(new_monotype(0, 2, 0, 0, 0, NULL, 0)).isbool) {
if(fib(new_monotype(0, 2, 0, 0, 0, NULL, 0)).b) {
printf("%s\n", "True");
} else {
printf("%s\n", "False");
}} else if(fib(new_monotype(0, 2, 0, 0, 0, NULL, 0)).isfloat) {
printf("%f\n", fib(new_monotype(0, 2, 0, 0, 0, NULL, 0)).f);
} else { printf("%s\n", "Error!"); }
if(fib(new_monotype(0, 3, 0, 0, 0, NULL, 0)).isint) {
printf("%d\n", fib(new_monotype(0, 3, 0, 0, 0, NULL, 0)).i);
} else if(fib(new_monotype(0, 3, 0, 0, 0, NULL, 0)).ischar) {
printf("%s\n", fib(new_monotype(0, 3, 0, 0, 0, NULL, 0)).s);
} else if(fib(new_monotype(0, 3, 0, 0, 0, NULL, 0)).isbool) {
if(fib(new_monotype(0, 3, 0, 0, 0, NULL, 0)).b) {
printf("%s\n", "True");
} else {
printf("%s\n", "False");
}} else if(fib(new_monotype(0, 3, 0, 0, 0, NULL, 0)).isfloat) {
printf("%f\n", fib(new_monotype(0, 3, 0, 0, 0, NULL, 0)).f);
} else { printf("%s\n", "Error!"); }
if(fib(new_monotype(0, 4, 0, 0, 0, NULL, 0)).isint) {
printf("%d\n", fib(new_monotype(0, 4, 0, 0, 0, NULL, 0)).i);
} else if(fib(new_monotype(0, 4, 0, 0, 0, NULL, 0)).ischar) {
printf("%s\n", fib(new_monotype(0, 4, 0, 0, 0, NULL, 0)).s);
} else if(fib(new_monotype(0, 4, 0, 0, 0, NULL, 0)).isbool) {
if(fib(new_monotype(0, 4, 0, 0, 0, NULL, 0)).b) {
printf("%s\n", "True");
} else {
printf("%s\n", "False");
}} else if(fib(new_monotype(0, 4, 0, 0, 0, NULL, 0)).isfloat) {
printf("%f\n", fib(new_monotype(0, 4, 0, 0, 0, NULL, 0)).f);
```

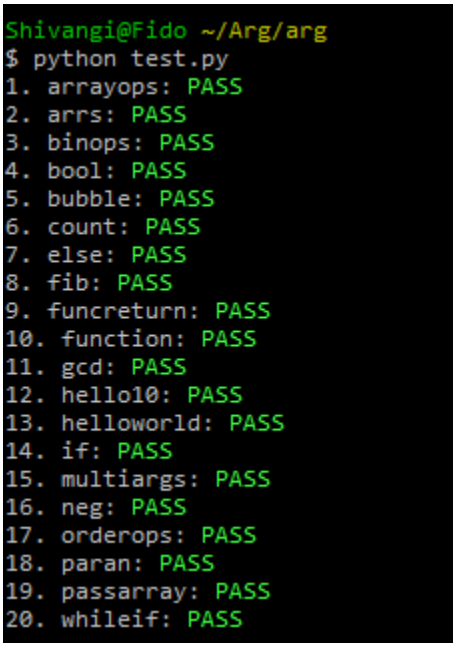
```
} else { printf("%s\n", "Error!"); }
if(fib(new_monotype(0, 5, 0, 0, 0, NULL, 0)).isint) {
printf("%d\n", fib(new_monotype(0, 5, 0, 0, 0, NULL, 0)).i);
} else if(fib(new_monotype(0, 5, 0, 0, 0, NULL, 0)).ischar) {
printf("%s\n", fib(new_monotype(0, 5, 0, 0, 0, NULL, 0)).s);
} else if(fib(new_monotype(0, 5, 0, 0, 0, NULL, 0)).isbool) {
if(fib(new_monotype(0, 5, 0, 0, 0, NULL, 0)).b) {
printf("%s\n", "True");
} else {
printf("%s\n", "False");
}} else if(fib(new_monotype(0, 5, 0, 0, 0, NULL, 0)).isfloat) {
printf("%f\n", fib(new_monotype(0, 5, 0, 0, 0, NULL, 0)).f);
} else { printf("%s\n", "Error!"); }


      return 0;
}
```

Generated fib.c



Corresponding C code in fib-model.c

Rest of the testing script can be found in section 8. As seen below, all 20 test cases passed:



7. Lessons Learned

a. Ryan Eagan

Time management was very important. I was thankful that other group members encouraged an early start to the project. Testing was very interesting, especially when I found that there were things we could write in ARG that I was unable to write a test model case for in C without significant extra data structures or extraneous code. Trying to figure out what mistakes others may have made in the compiler was quite challenging, but once a potential mistake was identified writing a test case for it was no problem.

b. Michael Goldin

Starting early was always going to be a winning strategy, and the end of the semester would have been a lot more painful than it was had we not. I came to really like programming in Ocaml by the end of the semester in spite of feeling really frustrated by it in the beginning--I think I'll do a lot more ML and functional programming in the future.

A good approach to building our compiler which David suggested was to go deep implementing a tiny tiny subset of features to completion and then expanding out from there with more features. Initially we were building the thing monolithically: implement the entire parser, the complete spec AST, and all of those at once in the compiler. That would have been a lot more difficult to do. Too much to wrap your brain around all at once. Deep > wide.

c. River Keefer

As Michael said, starting early was a huge advantage. Doing the whole project in the last couple weeks alone would have been hell.

Flexibility was also key. I was originally assigned to the tester role, but ended up spending more time on the compiler itself. Stubbornly trying to stick to our original roles would have meant some missed opportunity, I think, as I had a lot of fun writing OCaml and Ryan did a great job with the test suite.

Thanks to Michael, I learned some more complex git usage than I'm normally used to. This opened my eyes a bit to how useful it can be.

I'm also inspired to try to work on more functional projects in the future. I was thinking of starting a Haskell project in my spare time over break.

d. Shivangi Saxena

I learned how to work in big projects and collaborate with team members to fix deadlines and stick to them. Also, making sure you follow through the small things first and then get onto more complex features.

8. Full Code Listing

Argc.ml

```
Open Ast

let arg_file = Sys.argv.(1) ^ ".arg"
let c_file = Sys.argv.(1) ^ ".c"


type scope_entity = Function | Variable


(* Utility function, return a pair, second of which is true if the given list
   contains the given string and false otherwise. *)
let list_contains_string l s =
    List.fold_left (fun a b ->
        if ((snd a) || (String.compare (fst a) (fst b)) = 0) then (s, true) else
(s, false))
        (s, false) l


(* Utility function, return a pair, second of which is true if the given list
   contains the given string and that string is a Variable. false otherwise. *)
let element_is_variable l s =
    List.fold_left (fun a b ->
        if ((snd a) || ((String.compare (fst a) (fst b)) = 0) && (snd b =
Variable))
        then (s, true)
        else (s, false)
    )
    (s, false) l


(* Because everything in ARG must be represented by a C monotype, this function
   should return a pair, the first of which is a string of valid C code which
   will evaluate in C to a struct monotype. The second of the pair is the
   symbol table. *)
let rec monotype_of_expr expr st =
    match expr with
    | IntLiteral(i) -> ("new_monotype(0, " ^ string_of_int i ^
                    ", 0, 0, 0, NULL, 0)", st)
    | StrLiteral(str) -> ("new_monotype(1, 0, " ^ str ^ ", 0, 0, NULL, 0)", st)
    | BoolLiteral(b) -> if b then ("new_monotype(2, 0, 0, 1, 0, NULL, 0)", st)
                        else ("new_monotype(2, 0, 0, 0, 0, NULL, 0)", st)
    | FloatLiteral(f) -> ("new_monotype(3, 0, 0, 0, " ^ string_of_float f ^
```

```
                    ", NULL, 0)", st)
    | Assign(str, e) ->
        if (snd (list_contains_string st str) && snd (element_is_variable st str))
        then let (rhs, st) = monotype_of_expr e st in
            (str ^ " = " ^ rhs, st)
        else if snd (list_contains_string st str)
        then (print_string
             ("You are trying to assign to a function \"" ^ str ^ "\". Error.\n");
raise Exit)
        else let (rhs, st) = monotype_of_expr e st in
            ("struct monotype " ^ str ^ " = " ^ rhs, st @ [(str, Variable)])
    | Call(str, el) ->
        if (snd (list_contains_string st str) && not (snd (element_is_variable st
str)))
        then
            let arglist =
                List.fold_left (fun s e -> s ^ fst (monotype_of_expr e st) ^ ", ")
"" el in
            (* Arglist has an extra comma and space at its end. Remove them below.
*)
            let strlen = String.length arglist in
            let arglist = if String.length arglist != 0 then
            String.sub arglist 0 (strlen - 2) else "" in
            (str ^ "(" ^ arglist ^ ")", st)
        else if snd (list_contains_string st str)
        then (print_string
             ("You are trying to call a variable \"" ^ str ^ "\" as a function.
Error.\n"); raise Exit)
        else (print_string
             ("A function \"" ^ str ^ "\" is called which does not exist.
Error.\n"); raise Exit)
    | Id(str) ->
        if snd (list_contains_string st str)
        then (str, st)
        else (print_string
            ("An ID \"" ^ str ^ "\" is used which does not exist. Error.\n"); raise
Exit)
    | ArrId(str, i) ->
        if snd (list_contains_string st str)
        then (str ^ ".a[" ^ (fst (monotype_of_expr i st)) ^ ".i]", st)
        else (print_string
            ("An ID \"" ^ str ^ "\" is used which does not exist. Error.\n"); raise
Exit)
    | Binop(e1, op, e2) ->
        let arg_binop_to_c_binop e1 e2 = function
            | Add     -> ("monotype_add(" ^ fst (monotype_of_expr e1 st) ^ ", " ^
```

```
                        fst (monotype_of_expr e2 st) ^ ")", st)
            | Sub     -> ("monotype_sub(" ^ fst (monotype_of_expr e1 st) ^ ", " ^
                        fst (monotype_of_expr e2 st) ^ ")", st)
            | Mult     -> ("monotype_mult(" ^ fst (monotype_of_expr e1 st) ^ ", " ^
                         fst (monotype_of_expr e2 st) ^ ")", st)
            | Div     -> ("monotype_div(" ^ fst (monotype_of_expr e1 st) ^ ", " ^
                        fst (monotype_of_expr e2 st) ^ ")", st)
            | Equal     -> ("monotype_equal(" ^ fst (monotype_of_expr e1 st) ^ ", "
^
                          fst (monotype_of_expr e2 st) ^ ")", st)
            | Neq     -> ("monotype_neq(" ^ fst (monotype_of_expr e1 st) ^ ", " ^
                        fst (monotype_of_expr e2 st) ^ ")", st)
            | Less     -> ("monotype_less(" ^ fst (monotype_of_expr e1 st) ^ ", " ^
                         fst (monotype_of_expr e2 st) ^ ")", st)
            | Leq     -> ("monotype_leq(" ^ fst (monotype_of_expr e1 st) ^ ", " ^
                        fst (monotype_of_expr e2 st) ^ ")", st)
            | Greater     -> ("monotype_greater(" ^ fst (monotype_of_expr e1 st) ^
                           ", " ^ fst (monotype_of_expr e2 st) ^ ")", st)
            | Geq     -> ("monotype_geq(" ^ fst (monotype_of_expr e1 st) ^ ", " ^
                        fst (monotype_of_expr e2 st) ^ ")", st)
        in arg_binop_to_c_binop e1 e2 op


(* Generate a C string to create a new monotype with a persistent array of
   monotypes stored within it.
   name: the array's user-specified name.
   len: the array's user-specifie length.
   el: an expression list, the evaluations of which comprise the array's contents*)
let new_monotype_array name len el st =


    (* HACK. A user could in theory create an actual variable with this name. *)
    let tmpname = name ^ name ^ name ^ name in


    let len = monotype_of_expr len st in
    (* First malloc a monotype array to store persistently in the stack
       monotype *)
    "struct monotype *" ^ tmpname ^
    " = malloc(sizeof(struct monotype) * "^ (fst len) ^ ".i);\n" ^


    (* Load the evaluated results of the el into the malloc'd array in order. *)
    (fst
        (List.fold_left
            (fun p e ->
```

```
                (fst p) ^ tmpname ^ "[" ^ string_of_int (snd p) ^
                "] = " ^ fst (monotype_of_expr e st) ^ ";\n", succ (snd p)
            )
            ("", 0) (List.rev el)
        )
    ) ^


    (* Call the new_monotype constructor with the array flag and send in the
       malloc'd and loaded array. *)
    "struct monotype " ^ name ^ "  = " ^
    "new_monotype(4, 0, NULL, 0, 0, " ^ tmpname ^ ", " ^
    (fst len) ^ ".i);\n"


(* Generate an if/else block that directs a monotype to the appropriate printf
   format string.
   (* TODO: we should support PRINT statements with multiple arguments, so we'll
   need to build up the format strings in a smarter way, possibly by evaluating
   a list of expressions and programmatically building up fmt strings based on
   their types. *) *)
let arg_print_to_c_print fmt expr st =
    "if(" ^ fst (monotype_of_expr expr st) ^ ".isint) {\n" ^
    "printf(\"%d\\n\", " ^ fst (monotype_of_expr expr st) ^ ".i);\n" ^
    "} else if(" ^ fst (monotype_of_expr expr st) ^ ".ischar) {\n" ^
    "printf(\"%s\\n\", " ^ fst (monotype_of_expr expr st) ^ ".s);\n" ^
    "} else if(" ^ fst (monotype_of_expr expr st) ^ ".isbool) {\n" ^
    "if(" ^ fst (monotype_of_expr expr st) ^ ".b) {\n" ^
    "printf(\"%s\\n\", \"True\");\n" ^
    "} else {\n" ^
    "printf(\"%s\\n\", \"False\");\n}" ^
    "} else if(" ^ fst (monotype_of_expr expr st) ^ ".isfloat) {\n" ^
    "printf(\"%f\\n\", " ^ fst (monotype_of_expr expr st) ^ ".f);\n" ^
    "} else { printf(\"%s\\n\", \"Error!\"); }"


(* Add a new, unique label to the head of the jump table. Return the new jump
   table. *)
let add_label_to_jt jt =
    if List.length jt = 0 then ["j0"] else
    let labelnum = List.length jt in
    let label = "j" ^ string_of_int labelnum in
    label :: jt


(* Route an arg statement to its translator and return a triple with a valid C
```

```
   string, the jump table and the symbol table. *)
let rec arg_stmt_to_c_stmt stmt jt st =
    match stmt with
    | Expr(e) ->
        let (str, st) = monotype_of_expr e st in
        (str ^ ";\n", jt, st)
    | IfElse(e, s1, s2) ->
        let (if_body, jt, st) =
            List.fold_left
                (fun a b ->
                    let (sl, jt, st) = a in
                    let (stmt, jt, st) = arg_stmt_to_c_stmt b jt st in
                    (sl @ [stmt], jt, st)
                )
            ([], jt, st) s1
        in
        let (else_body, jt, st) =
            List.fold_left
                (fun a b ->
                    let (sl, jt, st) = a in
                    let (stmt, jt, st) = arg_stmt_to_c_stmt b jt st in
                    (sl @ [stmt], jt, st)
                )
            ([], jt, st) s2
        in
        ("if(" ^ fst (monotype_of_expr e st) ^ ".b) {\n" ^
        (List.fold_left (fun a b -> a ^ b) "" if_body)
        ^ "} else {\n" ^ (List.fold_left (fun a b -> a ^ b) "" else_body) ^ "}"
        , jt, st)


    | If(e, s) ->
        let (if_body, jt, st) =
            List.fold_left
                (fun a b ->
                    let (sl, jt, st) = a in
                    let (stmt, jt, st) = arg_stmt_to_c_stmt b jt st in
                    (sl @ [stmt], jt, st)
                )
            ([], jt, st) s
        in
        ("if(" ^ fst (monotype_of_expr e st) ^ ".b) {\n" ^
        (List.fold_left (fun a b -> a ^ b) "" if_body)
        ^ "}", jt, st)
    | While(e, s) ->
        let jt = add_label_to_jt jt in
```

```
        let label = List.hd jt in
        let (while_body, jt, st) =
            List.fold_left
                (fun a b ->
                    let (sl, jt, st) = a in
                    let (stmt, jt, st) = arg_stmt_to_c_stmt b jt st in
                    (sl @ [stmt], jt, st)
                )
            ([], jt, st) s
        in
        (
            label ^ ":;\nif(" ^ fst (monotype_of_expr e st) ^ ".b) {\n" ^
            (List.fold_left (fun a b -> a ^ b) "" while_body) ^
            "goto " ^ label ^ ";\n}",
            jt, st
        )
    | ArrayAssign(s, l, el) ->
        (new_monotype_array s l el st ^ "\n", jt, st @ [(s, Variable)])
    | ArrayElemAssign(s, i, e) ->
        (s ^ ".a[" ^ (fst (monotype_of_expr i st)) ^ ".i] = " ^ (fst
(monotype_of_expr e st) ^ ";\n"), jt, st)
    | Print(s, e) -> (arg_print_to_c_print s e st ^ "\n", jt, st)
    | Return(e) -> ("return " ^ (fst (monotype_of_expr e st)) ^ ";\n", jt, st)


(* Convert a list of arg statements to a list of valid C strings. Return that
   list, the jump table and the symbol table. *)
let arg_body_to_c_body arg_body jt st =
    List.fold_left
        (fun a b ->
            let (sl, jt, st) = a in
            let (stmt, jt, st) = arg_stmt_to_c_stmt b jt st in
            (sl @ [stmt], jt, st)
        )
    ([], jt, st) arg_body


(* Translate an arg function in the AST to a C function, returning a string of
   that translation along with the jump table and symbol table. *)
let arg_func_to_c_func arg_func jt st =
    let arglist =
        List.fold_left (fun a b -> a ^ "struct monotype " ^ b ^ ", ") ""
arg_func.formals in
    (* Arglist has an extra comma and space at its end. Remove them below. *)
    let strlen = String.length arglist in
    let arglist = if String.length arglist != 0 then
```

```
        String.sub arglist 0 (strlen - 2) else "" in


    let st = st @ (List.map (fun a -> (a, Variable))) arg_func.formals in
    let (func_body, jt, st) =
        List.fold_left
            (fun a b ->
                let (sl, jt, st) = a in
                let (stmt, jt, st) = arg_stmt_to_c_stmt b jt st in
                (sl @ [stmt], jt, st)
            )
        ([], jt, st) arg_func.body
    in
    (
        "struct monotype " ^ arg_func.fname ^ "(" ^ arglist ^ ") {\n" ^
        (List.fold_left (fun a b -> a ^ b) "" func_body) ^ "\n}",
        jt, st
    )


(* Utility function. Remove all scope_entities with the Variable type from the
   symbol table. Call this after translating functions. *)
let remove_vars_from_st st =
    List.fold_left
        (fun a b ->
            if snd b = Function
            then a @ [b]
            else a
        )
        [] st


(* Route the functions and body segments of the program pair to their respective
   handlers and return the result as a pair of strings. *)
let translate_program arg =
    (* Name working elements. *)
    let arg_funcs = fst arg in
    let arg_body = snd arg in


    (* Convert ARG to C. *)
    let (c_funcs, jt, st) =
        List.fold_left
        (fun a b ->
                let (sl, jt, st) = a in
                let st = st @ [(b.fname, Function)] in
```

```
                let (stmt, jt, st) = arg_func_to_c_func b jt
                    (remove_vars_from_st st) in
                (sl @ [stmt], jt, st)
        )
        ([], [], []) arg_funcs
    in
    let st = remove_vars_from_st st in
    let (c_body, jt, st) = arg_body_to_c_body arg_body jt st in
    (
        List.fold_left (fun a b -> a ^ b) "" c_funcs,
        List.fold_left (fun a b -> a ^ b) "" c_body,
        st
    )


let generate_free_block st =
    List.fold_left (fun a b ->
        if (snd b) = Variable
        then "if(" ^ (fst b) ^ ".isarray) { free(" ^ (fst b) ^ ".a); }\n"
        else "") "" st


(* Include necessary C libraries. Declare functions at top of file, then wrap
   body in a main function below.
   (* TODO: We shouldn't be including monotype.c, we should write its contents
      into the file. We obviously can't assume the user will have their own copy
      of monotype.c in their own directory! *)
*)
let wrap_program translated_program =
    let (functions,_,_) = translated_program in
    let (_,body,_) = translated_program in
    let (_,_,st) = translated_program in
    let free_block = generate_free_block st in
    "#include <stdio.h>\n#include \"monotype.c\"\n" ^ functions ^
    "\n\nint main() {\n" ^ body ^ "\n\n" ^ free_block ^ "\treturn 0;\n}\n"


let _ =
    let ic = open_in arg_file in
    let lexbuf = Lexing.from_channel ic in
    let arg = Parser.program Scanner.token lexbuf in
    let oc = open_out c_file in
    Printf.fprintf oc "%s\n" (wrap_program (translate_program arg));
    print_endline ("generated " ^ c_file);
    close_out oc;
    close_in ic;
```

Ast.mli

```
type op = Add | Sub | Mult | Div | Equal | Neq | Less | Leq | Greater | Geq

type expr =
  | Assign of string * expr
  | Call of string * expr list
  | Id of string
  | ArrId of string * expr
  | StrLiteral of string
  | IntLiteral of int
  | FloatLiteral of float
  | BoolLiteral of bool
  | Binop of expr * op * expr


type statement =
  | Expr of expr
  | If of expr * statement list
  | IfElse of expr * statement list * statement list
  | While of expr * statement list
  | ArrayAssign of string * expr * expr list
  | ArrayElemAssign of string * expr * expr
  | Print of string * expr
  | Return of expr


type func = {
  fname : string;
  formals : string list;
  body : statement list;
}


type program = func list * statement list
```

Parser.mly

```
%{ open Ast %}

%token ADD SUB MULT DIV EQUAL NEQ LESS LEQ GREATER GEQ
%token COMMA SEMI
%token LPAREN RPAREN
%token LBRACE RBRACE
%token LBRACK RBRACK
%token ASSIGN
%token FUNCTION WHILE IF ELSE PRINT RETURN
%token <string> STRLITERAL
%token <int>    INTLITERAL
%token <float>  FLOATLITERAL
%token <bool>   BOOLLITERAL
%token <string> ID
%token EOF


%right ASSIGN
%left EQUAL NEQ
%left LESS GREATER LEQ GEQ
%left ADD SUB
%left MULT DIV


%start program
%type <Ast.program> program


%%


program:
  code EOF                              { $1 }


code:
  | functions body                      { ($1, $2) }


body:
  | /* nothing */                       { [] }
  | body statement                      { $1 @ [$2] }
```

```
functions:
  | /* nothing */                    { [] }
  | functions func                   { $1 @ [$2] }


func:
  | FUNCTION ID LPAREN params_opt RPAREN LBRACE stmt_opt RBRACE
    {
      { fname = $2;
        formals = $4;
        body = (List.rev $7); }
    }


statement:
  | expr SEMI                        { Expr($1) }
  | IF LPAREN expr RPAREN LBRACE
    stmt_opt RBRACE ELSE LBRACE
    stmt_opt RBRACE                  { IfElse($3, (List.rev $6), (List.rev $10)) }
  | IF LPAREN expr RPAREN LBRACE
    stmt_opt RBRACE                  { If($3, (List.rev $6)) }
  | WHILE LPAREN expr RPAREN LBRACE stmt_opt RBRACE { While($3, (List.rev $6)) }
  | ID LBRACK expr RBRACK ASSIGN LBRACE actuals_opt RBRACE SEMI     {
ArrayAssign($1, $3, $7) }
  | ID LBRACK expr RBRACK ASSIGN expr SEMI     { ArrayElemAssign($1, $3, $6) }
  | PRINT LPAREN STRLITERAL COMMA expr RPAREN SEMI { Print($3, $5) }
  | RETURN expr SEMI                 { Return($2) }


expr:
  | ID ASSIGN expr                    { Assign($1, $3) }
  | ID LPAREN actuals_opt RPAREN      { Call($1, $3) }
  | STRLITERAL                        { StrLiteral($1) }
  | INTLITERAL                        { IntLiteral($1) }
  | FLOATLITERAL                      { FloatLiteral($1) }
  | BOOLLITERAL                       { BoolLiteral($1) }
  | ID                                { Id($1) }
  | ID LBRACK expr RBRACK             { ArrId($1, $3) }
  | expr ADD expr                     { Binop($1, Add, $3) }
  | expr SUB expr                     { Binop($1, Sub, $3) }
  | expr MULT expr                    { Binop($1, Mult, $3) }
  | expr DIV expr                     { Binop($1, Div, $3) }
  | expr EQUAL expr                   { Binop($1, Equal, $3) }
  | expr NEQ expr                     { Binop($1, Neq, $3) }
  | expr LESS expr                    { Binop($1, Less, $3) }
  | expr LEQ expr                     { Binop($1, Leq, $3) }
```

```
  | expr GREATER expr                { Binop($1, Greater, $3) }
  | expr GEQ expr                    { Binop($1, Geq, $3) }
  | LPAREN expr RPAREN               { $2 }


stmt_opt:
  | /* Nothing */                    { [] }
  | stmt_list                        { $1 }


stmt_list:
  | statement                        { [$1] }
  | stmt_list statement              { $2 :: $1 }


actuals_opt:
  | /* Nothing */                    { [] }
  | actuals_list                     { $1 }


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


params_opt:
  | /* nothing */                   { [] }
  | params_list                      { $1 }


params_list:
  | ID                               { [$1] }
  | params_list COMMA ID             { $3 :: $1 }
```

Scanner.mll

```
{ open Parser }
rule token = parse
| [' ' '\t' '\r' '\n'] { token lexbuf }
| "/*"                 { comment lexbuf }
| '('       { LPAREN }
| ')'       { RPAREN }
| ','       { COMMA }
| ';'       { SEMI }
| '='       { ASSIGN }
| '{'       { LBRACE }
| '}'       { RBRACE }
| '['       { LBRACK }
| ']'       { RBRACK }
| '+'       { ADD }
| '-'       { SUB }
| '*'       { MULT }
| '/'       { DIV }
| "=="      { EQUAL }
| "!="      { NEQ }
| "<"       { LESS }
| "<="      { LEQ }
| ">"       { GREATER }
| ">="      { GEQ }
| "IF"      { IF }
| "ELSE"    { ELSE }
| "WHILE"   { WHILE }
| "FUNCTION"{ FUNCTION }
| "RETURN"  { RETURN }
| "PRINT"   { PRINT }
| '"'['^''\n''"']*'"' as lxm { STRLITERAL(lxm) }
| ['0'-'9']+     as lxm { INTLITERAL(int_of_string lxm) }
| ['0'-'9']+'.'['0'-'9']    as lxm { FLOATLITERAL(float_of_string lxm)}
| "true"    { BOOLLITERAL(true) }
| "false"   { BOOLLITERAL(false) }
| ['A'-'Z' 'a'-'z']['A'-'Z' 'a'-'z' '0'-'9' '_']* as lxm { ID(lxm) }
| eof       { EOF }
| _ as char { raise (Failure("Illegal character " ^ Char.escaped char)) }

and comment = parse
| "*/"  { token lexbuf }
| _     { comment lexbuf }
```

Makefile:

```
OBJS = parser.cmo scanner.cmo argc.cmo
PROG = argc


all: $(PROG)


argc : $(OBJS)
	ocamlc str.cma -o argc $(OBJS)


scanner.ml : scanner.mll
	ocamllex scanner.mll


parser.ml parser.mli : parser.mly
	ocamlyacc -v parser.mly


%.cmo : %.ml
	ocamlc -c $<

%.cmi : %.mli
	ocamlc -c $<


.PHONY : clean
clean :
	rm -f parser.ml parser.mli scanner.ml *.cmo *.cmi $(PROG)

# Generated by ocamldep *.ml *.mli
arg.cmo: scanner.cmo parser.cmi ast.cmi
arg.cmx: scanner.cmx parser.cmx ast.cmi
parser.cmo: ast.cmi parser.cmi
parser.cmx: ast.cmi parser.cmi
scanner.cmo: parser.cmi
scanner.cmx: parser.cmx
parser.cmi: ast.cmi
```

Example.arg –

```
a = (int) argv[1];
b = "yo";
c = 420;
d = rando(a, b, c);
e = rando(a, b, c);

FUNCTION rando(a, b, c) {

  if(a == 5) {
    return b;
  } else {
    return c;
  }


}


PRINT("%x\n", d);
```

Test.py:

```
import subprocess
import sys
import os


GOOD = '\033[92m'
BAD = '\033[91m'
END = '\033[0m'

TESTS = []
for f in os.listdir("tests"):
       if ".arg" in f:
              TESTS.append(f[:-4])

#supp = " >/dev/null 2>&1";
supp = " > argerr"

if __name__ == "__main__":

              for i in range(0, len(TESTS)):
                     program = TESTS[i];

                     try:
                                   err = os.system("./argc tests/"+program+supp)
                                   if err != 0:
                                          print(str(i+1) + ". "+program +":
"+BAD+"FAIL"+END)
                                          print("\tFailed to compile")
                                          print "\t"+open("argerr", "r").read()
                                          continue
                                   os.system("gcc -w -o "+program+"
tests/"+program+".c")
                                   arg_output =
subprocess.check_output(["./"+program])
                                   os.system("rm " + program)
                                   os.system("rm tests/"+program+".c")
                     except:
                                   quit()
                     try:
                                   os.system("gcc -w -o model tests/"+program+"-
model.c")
                                   err = os.system("./model"+supp)
                                   model_output = open("argerr", "r").read()
                                   os.system("rm model")
```

```
                        except:
                                print("failed to run arg exec")
                                print(str(i+1) + ". "+program +": "
+BAD+"FAIL"+END)
                                print "\tModel failed to compile"
                                continue
                        if arg_output == model_output:
                                print(str(i+1) + ". "+program +":
"+GOOD+"PASS"+END)
                        else:
                                print(str(i+1) + ". "+program +":
"+BAD+"FAIL"+END)
                                arg_lines = arg_output.split("\n")
                                model_lines = model_output.split("\n")
                                for i in range(0,len(arg_lines)):
                                        if arg_lines[i] != model_lines[i]:
                                                print "\t"+str(i+1) + ". arg: " +
arg_lines[i] + "\t model: " + model_lines[i]
```

Demo programs:

Demo.arg –

```
len = 5;
arr[len] = {0, "hello", "yo", 678, 8};
i = 0;
WHILE(i < len) {
	PRINT("%x\n", arr[i]);
	i = i + 1;
}
```

Demo2.arg –

```
FUNCTION printall(a,b,c,d) {
	PRINT("%x\n", a);
	PRINT("%x\n", b);
	PRINT("%x\n", c);
	PRINT("%x\n", d);
}



printall("hello", "world", 1, 2);
printall(2, "arg string", 1, 2);
```

Demo3.arg –

```
FUNCTION rightname(a) {
	PRINT("%x\n", a);
}

rightname(1);
wrongname(1);
```

Demo4.arg –

```
variab = 5;
variab(10);
```

Demo5.arg –

```
FUNCTION f() {
	a = 5;
}

f = 6;
```

Testing Scripts: ~/tests

Arg files:

1. arrayops.arg –

```
FUNCTION printfunc(val) {
	PRINT("%x\n", val);
}


len = 10;
a[len] = {};


i = 0;
WHILE(i < len) {
	a[i] = 10 - i;
	i = i + 1;
}


i = i - 1;
WHILE(i >= 0) {
	printfunc(a[i]);
	i = i - 1;
}
```

2. arrs.arg –

```
a[5] = {1, 2, 3, 4, 5};

PRINT("%x\n", a[3])
```

3. binops.arg –

```
five = 5;
two = 2;


PRINT("%x\n", 400 + 20);
PRINT("%x\n", five + two);
```

4. bool.arg –

```
a = true;
b = false;



IF(a == true) {
	PRINT("%x\n", "a is true");
}
ELSE {
	PRINT("%x\n", "a is false");
}
IF(b == true) {
	PRINT("%x\n", "b is true");
}
ELSE {
	PRINT("%x\n", "b is false");
}
```

5. bubble.arg-

```
len = 10;
arr[len] = {1,5,10,3,67,3,2,88,43,8};



i = 0;
temp = 0;
j = 0;


WHILE(i < len - 1) {
	j = 0;
	WHILE(j < len - i - 1) {
		IF(arr[j] > arr[j+1]) {
			temp = arr[j];
			arr[j] = arr[j+1];
			arr[j+1] = temp;
		}
		j = j + 1;
	}
	i = i + 1;
}



i = 0;
WHILE(i < len) {
```

```
        PRINT("%x\n", arr[i]);
        i = i + 1;
}
```

6. count.arg –

```
i = 0;
WHILE(i < 100) {
        PRINT("%x\n", i);
        i = i + 1;
}
```

7. else.arg –

```
a = 1;
b = 0;



IF(a == 1) {
        PRINT("%x\n", "a is true");
        IF(b == 1) {
                PRINT("%x\n", "b is true");
        }
        ELSE {
                PRINT("%x\n", "b is false");
        }
}
ELSE {
        PRINT("%x\n", "a is false");
}
```

8. fib.arg –

```
FUNCTION fib(n) {
     IF(n < 2) {
                RETURN 1;
        }
        RETURN fib(n-1) + fib(n-2);
}



PRINT("%x\n", fib(0));
PRINT("%x\n", fib(1));
PRINT("%x\n", fib(2));
PRINT("%x\n", fib(3));
PRINT("%x\n", fib(4));
```

```
PRINT("%x\n", fib(5));
```

9. funcreturn.arg –

```
FUNCTION square(val) {
	RETURN val * val;
}



a = 5;
b = square(a);
PRINT("%x\n", b);
```

10. function.arg –

```
FUNCTION printfunc(str) {
	PRINT("%x\n", str);
}

printfunc("line1");
printfunc("line2");
```

11. gcd.arg –

```
FUNCTION gcd(a, b) {
	WHILE(a != b) {
		IF(a > b) {
			a = a - b;
		}
		ELSE {
			b = b - a;
		}
	}
	RETURN a;
}



PRINT("%x\n", gcd(2,10));
PRINT("%x\n", gcd(4,100));
PRINT("%x\n", gcd(3,15));
```

```
PRINT("%x\n", gcd(100,250));
```

12. hello10.arg –

```
i = 0;
msg = "Hello, world!";


WHILE(i < 10) {
    PRINT("%x\n", msg);
    i = i + 1;
}
```

13. helloworld.arg –

```
msg = "Hello, World!";

PRINT("%x\n", msg);
```

14. if.arg –

```
a = 1;
IF(a == 1) {
	PRINT("%x\n", "a is true");
}
ELSE {
	PRINT("%x\n", "a is false");
}

b = 0;

IF(b == 1) {
	PRINT("%x\n", "b is true");
}
ELSE {
	PRINT("%x\n", "b is false");
}
```

15. multiargs.arg –

```
FUNCTION multi(a,b,c,d,e,f) {
	PRINT("%x\n", a);
	PRINT("%x\n", b);
	PRINT("%x\n", c);
```

```
        PRINT("%x\n", d);
        PRINT("%x\n", e);
        PRINT("%x\n", f);
}


multi(1,2,3,4,5,6);
multi(2,2,2,2,2,2);
multi(77,3,2,62,2,2);
multi(23,21,22,32,2,92);
```

16. neg.arg –

```
a = 0-5;
b = 0-10;
c = a - b;
PRINT("%x\n", a);
PRINT("%x\n", b);
PRINT("%x\n", c);
```

17. orderops.arg –

```
a = 4 + 5 * 8;
b = (4 + 5) * 8;
PRINT("%x\n", a);
PRINT("%x\n", b);
```

18. paran.arg –

```
a = ((1-2)-3+5*9);
b = (((a - 2)));
c = ((0) - b);


PRINT("%x\n", a);
PRINT("%x\n", b);
PRINT("%x\n", c);
```

19. passarray.arg –

```
FUNCTION printarray(arr, len) {
        i = 0;
        WHILE(i < len) {
                PRINT("%x\n", arr[i]);
                i = i+1;
        }
```

```
}

a[5] = {1,2,3,4,5};
b[3] = {1, 10, 100};
printarray(a, 5);
printarray(b, 3);
```

20. whileif.arg –

```
a = 10;
b = 0;



WHILE(a != b) {
	IF(a > b) {
		a = a - 1;
	}
	IF(a < b) {
		a = a + 1;
	}
}
PRINT("%x\n", a);
```

C files –

1. arrayops-model.c –

```c
#include <stdio.h>
static void printfunc(int i) {
	printf("%d\n", i);
}

int main() {
	int len = 10;
	int a[len];

	int i = 0;
	while(i < len) {
		a[i] = 10-i;
		i++;
	}

    i--;
	while(i >= 0) {
		printfunc(a[i]);
		i--;
	}



}
```

2. arrs-model.c –

```c
#include <stdio.h>
int main() {
    int a[5] = {1, 2, 3, 4, 5};


    printf("%d\n", a[3]);


    return 0;
}
```

3. binops-model.c –

```c
#include <stdio.h>
int main() {
    int five = 5;
    int two = 2;



    printf("%d\n", 400 + 20);
    printf("%d\n", five + two);
}
```

4. bool-model.c –

```c
#include <stdio.h>
int main() {
	int a = 1;
	int b = 0;
	if(a) {
		printf("a is true\n");
	}
	else {
		printf("a is false\n");
	}
	if(b) {
		printf("b is true\n");
	}
	else {
		printf("b is false\n");
	}
}
```

5. bubble-model.c –

```c
#include <stdio.h>
int main() {
	int len = 10;
	int arr[10] = {1,5,10,3,67,3,2,88,43,8};
	int i = 0;
	int temp = 0;
	int j = 0;
	while(i < len - 1) {
		j = 0;
		while(j < len - i - 1) {
			if(arr[j] > arr[j+1]) {
				temp = arr[j];
				arr[j] = arr[j+1];
				arr[j+1] = temp;
```

```
                        }
                        j++;
                }
                i++;
        }

        i = 0;
        while(i < len) {
                printf("%d\n", arr[i]);
                i++;
        }


        return 0;
}
```

6. count-model.c –

```
#include <stdio.h>
int main() {
        int i;
        for(i=0;i<100;i++) {
                printf("%d\n", i);
        }


        return 0;
}
```

7. else-model.c –

```
#include <stdio.h>
int main() {
        int a = 1;
        int b = 0;


        if(a) {
                printf("a is true\n");
                if(b) {
                        printf("b is true\n");
                }
                else printf("b is false\n");
        }
        else printf("a is false\n");
```

```
        return 0;
}
```

8. fib-model.c –

```
#include <stdio.h>
static int fib(int n) {
        if(n < 2) return 1;
        return fib(n-1) + fib(n-2);
}


int main() {
        printf("%d\n", fib(0));
        printf("%d\n", fib(1));
        printf("%d\n", fib(2));
        printf("%d\n", fib(3));
        printf("%d\n", fib(4));
        printf("%d\n", fib(5));
}
```

9. funcreturn-model.c –

```
#include <stdio.h>
static int square(int val) {
        return val * val;
}


int main() {
        int a = 5;
        int b = square(a);
        printf("%d\n", b);
}
```

10. function-model.c –

```
#include <stdio.h>
static void test(char *str) {
        printf("%s\n", str);
}
```

```
int main() {
        char *s1 = "line1";
        char *s2 = "line2";
        test(s1);
        test(s2);
}
```

11. gcd-model.c –

```
#include <stdio.h>
static int gcd(int a, int b) {
        while(a != b) {
                if(a > b) a = a - b;
                else b = b - a;
        }
        return a;
}


int main() {
        printf("%d\n", gcd(2,10));
        printf("%d\n", gcd(4,100));
        printf("%d\n", gcd(3,15));
        printf("%d\n", gcd(100,250));
        return 0;
}
```

12. hello10-model.c –

```
#include <stdio.h>
int main() {
    int i = 0;
    char *msg = "Hello, world!";


    while(i < 10) {
        printf("%s\n", msg);
        i = i + 1;
    }
}
```

13. helloworld-model.c –

```
#include <stdio.h>
int main() {
        char *msg = "Hello, World!";
```

```
        printf("%s\n", msg);


        return 0;
}
```

14. if-model.c –

```
#include <stdio.h>
int main() {
        int a = 1;
        if(a) {
                printf("a is true\n");
        }
        else {
                printf("a is false\n");
        }


        int b = 0;
        if(b) {
                printf("b is true\n");
        }
        else {
                printf("b is false\n");
        }
}
```

15. multiargs-model.c –

```
#include <stdio.h>
static void multi(int a,int b,int c,int d,int e,int f) {
        printf("%d\n", a);
        printf("%d\n", b);
        printf("%d\n", c);
        printf("%d\n", d);
        printf("%d\n", e);
        printf("%d\n", f);
}


int main() {
        multi(1,2,3,4,5,6);
        multi(2,2,2,2,2,2);
        multi(77,3,2,62,2,2);
        multi(23,21,22,32,2,92);
```

```
        return 0;
}
```

16. negs-model.c –

```
#include <stdio.h>
int main() {
        int a = -5;
        int b = -10;
        int c = a -b;


        printf("%d\n", a);
        printf("%d\n", b);
        printf("%d\n", c);
        return 0;
}
```

17. orderops-model.c –

```
#include <stdio.h>
int main() {
        int a = 4 + 5 * 8;
        int b = (4 + 5) * 8;
        printf("%d\n", a);
        printf("%d\n", b);
}
```

18. paran-model.c –

```
#include <stdio.h>
int main() {
        int a = ((1-2)-3+5*9);
        int b = (((a - 2)));
        int c = ((0) - b);

        printf("%d\n", a);
        printf("%d\n", b);
        printf("%d\n", c);
}
```

19. passarray-model.c –

```
#include <stdio.h>
static int printarray(int * arr, int len) {
        int i = 0;
```

```
        while(i < len) {
                printf("%d\n", arr[i]);
                i++;
        }
}


int main() {
        int a[5] = {1,2,3,4,5};
        int b[3] = {1,10, 100};
        printarray(a, 5);
        printarray(b, 3);
        return 0;
}
```

20. whileif-model.c –

```
#include <stdio.h>
int main() {
        int a = 10;
        int b = 0;
        while(a != b) {
                if(a > b) {
                        a--;
                }
                if(a < b) {
                        a++;
                }
        }
        printf("%d\n", a);


        return 0;
}
```