Columbia's Awk Replacement Language (CARL)
Final Report

COMS 4115 Programming Languages and Translators (Spring 2017)

Darren Hakimi (dh2834)	System Architect
Keir Lauritzen (kcl2143)	Manager
Leon Song (ls3233)	Tester
Guy Yardeni (gy2241)	Language Guru

1. Introduction

Description of Language

Columbia's Awk Replacement Language (CARL) is a record-processing, string-parsing language with syntax based on GNU Awk (Gawk) with a subset of the language features.[1] GNU Awk is an extension of the original Awk language at Bell Labs by A. Aho, P. Weinberger, and B.Kernighan (hence, AWK). Awk is an interpreted language by CARL will be compiled.

CARL programs follows the same structure as Awk:

```
pattern1 {action1}
pattern2 {action2}
```

The language reads each record (nominally each line but defined in `RS`) from the input file, checks whether it matches the pattern and if it does it takes the listed action. The pattern matching is supported using regular expressions. All records are processed against each pattern in order (for example, all records will be processed against `pattern1` then all records will be processed against `pattern2`. This behavior matches Awk's behavior as an interpreted language.

Two special-purposed patterns are BEGIN and END. All actions in a BEGIN pattern are executed at the beginning of the program before any records are processed and they are executed only once. Actions with the END pattern are the same only executing at the end of the records. The actions specified inside { } between the BEGIN and END patterns will recur per field.

[1] This project is based on GNU Awk and the language documentation at https://www.gnu.org/software/gawk/manual/gawk.html was used extensively in preparation of this proposal.

The actions support the standard set of control statements, variable assignments, numerical operations, and user-defined functions. The syntax is similar to C but does not support pointers or dynamic memory (outside of the built in datatypes of string and associative array).

Intended Uses

CARL (like Awk) is intended to be used to parse and manipulate records in text files. Common uses for complex CARL programs are performing calculations on text based data sources, such as CSV files. With CARL it is possible to calculate statistics or perform bookkeeping on complex data tables. A simple CARL usage would be to parse a log file for a specific event.

Parts of the Language

CARL, like its predecessor AWK, follows a `pattern {action}` syntax.

Patterns:

`BEGIN`	`BEGIN` is a special keyword indicating that the action is to be run before any records are processed.
`END`	`END` is a special keyword indicating that the action is to be run after all records in the file have been processed.
Regular Expressions	Regular expressions can be used if they evaluate to non-zero or not null.

Regular Expressions:

CARL will use regular expressions heavily to parse through semi-structured files.

Regex Definition and Usage	Like awk, you can define a regex by enclosing it in slashes. e.g. `//abc//`, which will attempt to iterate through every record and perform some action if the string “`abc`” is found in that record.

Regex Operators:

`\`	Backslash is used to suppress any special meaning associated with the character e.g. “ `\”hello\”` “ will search for the string “`hello`”, inclusive of

	quotation marks
`^`	Forces the program to look for matches at the beginning of the string for example `/^foo/` will match “football” but not “afoot”
`$`	Similar to `^`, except the program looks for matches at the end of the string instead
`.`	This matches any **single** character. For example, `/.a/` matches “`ba`” and even the whitespace character “ `a`”, but not “`ab`”
`[ x ]`	Called the bracket expression. Matches any single character enclosed within the square brackets, for example `[0-9]` searches for any digits in the string
`[ ^x ]`	Complement of the bracket expression. Matches any single character **not** within the square brackets e.g. `[^0-9]` will match any character that is not a digit
`*`	Repeats the preceding regular expression as many times as necessary to find a match. For example, `\ab*\` matches “`ab`”, “`abbbbb`” and “a” Whereas `\(ab*)\` matches “`ab`”, “`abababab`”, **and the empty string**

Actions:

Awk is a Turing-complete language with support for different data types and structures, control flow, I/O, and user-defined functions. CARL will implement most of the language, but will exclude many of the built-in variables and some, less common, aspects of the language.

Data types and structures

Floating point	Supports a single type of floating-point number.
String	Standard string data type.
Associative arrays	One-dimensional associative array. There is language support for for-loops through the array and adding and deleting elements. Indices of arrays can be either numbers or strings.

Control flow

`If` statement	C-style if-then-else controls. `if (expr) { body } else {body}`
`For` statement	C-style for loops.
`While` statement	C-style while loops.
User-defined functions	The ability to implement functions

CARL enforces the need for {} around the body of control switches. Only for loops are supported. While and do-while are not supported.

Reserved variables:

`RS`	Variable that stores the literal for separating the input file into multiple records.
`FS`	Variable that stores the literal for separating a record into multiple fields.
`NPAT`	Number of records processed. Internal variable.
`__patterns`	Array of patterns. Used internally.
`__actions`	Array of actions. Used internally.

Builtin Functions

`Print_string`	Print the string provided.
`print_float`	Print the float provided.

Original Language Whitepaper

Description of Language

Columbia's Awk Replacement Language (CARL) is a record-processing, string-parsing language with syntax based on GNU Awk (Gawk) with a subset of the language features.[2] GNU Awk is an extension of the original Awk language at Bell Labs by A. Aho, P. Weinberger, and B.Kernighan (hence, AWK). Awk is an interpreted language by CARL will be compiled.

CARL programs follows the same structure as Awk:

```
pattern1 {action1}
pattern2 {action2}
```

The language reads each record (nominally each line but defined in `RS`) from the input file, checks whether it matches the pattern and if it does it takes the listed action. The pattern matching is supported using regular expressions. Each record is subsequently divided into fields and the actions can be applied to each field (nominally delimited by spaces but defined using `FS`). All records are processed against each pattern in order (for example, all records will be processed against `pattern1` then all records will be processed against `pattern2`. This behavior matches Awk's behavior as an interpreted language.

Two special-purposed patterns are BEGIN and END. All actions in a BEGIN pattern are executed at the beginning of the program before any records are processed and they are executed only once. Actions with the END pattern are the same only executing at the end of the records. The actions specified inside { } between the BEGIN and END patterns will recur per field.

The actions support the standard set of control statements, variable assignments, numerical operations, and user-defined functions. The syntax is similar to C but does not support pointers or dynamic memory (outside of the built in datatypes of string and associative array). The actions can operate on the entire record using the $0 field designator or on positional field indicators $1, $2,... Field designations are do not allow manipulation within the field, so CARL provides a set of builtin functions to manipulate records.

Intended Uses

CARL (like Awk) is intended to be used to parse and manipulate records in text files. Common uses for complex CARL programs are performing calculations on text based data sources, such

[2] This project is based on GNU Awk and the language documentation at https://www.gnu.org/software/gawk/manual/gawk.html was used extensively in preparation of this proposal.

as CSV files. With CARL it is possible to calculate statistics or perform bookkeeping on complex data tables. A simple CARL usage would be to parse a log file for a specific event.

We intend to extend CARL from Awk to provide additional capability for semi-structured data. Examples of semi-structured data include XML and HTML files, where there is computer-readable information but it varies in length and form. CARL has a wide array of uses in semi-structured files. CARL can parse through files and extract portions of the file data based on tags. CARL could be used for web scraping because as it can iterate through the HTML and identify patterns in the tags. The data extracted, such as email addresses and phone numbers, can be formatted into a table for later use. An example extension identified so far is the inclusion of an `MF` reserved variable, which stores the field that matches the regular expression.

Parts of the Language

CARL, like its predecessor AWK, follows a `pattern {action}` syntax.

Patterns:

`BEGIN`	`BEGIN` is a special keyword indicating that the action is to be run before any records are processed.
`END`	`END` is a special keyword indicating that the action is to be run after all records in the file have been processed.
Expressions	Expressions can be used if they evaluate to non-zero or not null.
Character classes	Character classes are shorthands to describe sets of characters that have a specific attribute, and can only be used inside bracket expressions. They are denoted by "[:" followed by the keyword for that character class, then ":] E.g. `[:alnum:]` matches all alphanumeric characters, a convenient way to avoid typing "`/[A-Za-z0-9]/`" Character classes and their keywords are defined by the POSIX standard

Regular Expressions:

CARL will use regular expressions heavily to parse through semi-structured files.

Regex Definition and Usage	Like awk, you can define a regex by enclosing it in slashes. e.g. `/abc/`, which will attempt to iterate through every record and

	perform some action if the string “abc” is found in that record.
Dynamic Regex	CARL will allow for any expression to be used as a regex. That is, given an expression, CARL converts it to a string then uses it as a regex
Ambiguity	In cases of ambiguity, CARL takes the leftmost longest match For example, \a*\ for the string “aaaabbaa” matches with “aaaa” instead of “aa”.

Regex Operators:

\	Backslash is used to suppress any special meaning associated with the character e.g. “ \”hello\” “ will search for the string “hello”, inclusive of quotation marks
^	Forces the program to look for matches at the beginning of the string for example /^foo/ will match “football” but not “afoot”
$	Similar to ^, except the program looks for matches at the end of the string instead
.	This matches any **single** character. For example, /.a/ matches “ba” and even the whitespace character “ a”, but not “ab”
[x]	Called the bracket expression. Matches any single character enclosed within the square brackets, for example [0-9] searches for any digits in the string
[^x]	Complement of the bracket expression. Matches any single character **not** within the square brackets e.g. [^0-9] will match any character that is not a digit
*	Repeats the preceding regular expression as many times as necessary to find a match. For example, \ab*\ matches “ab”, “abbbbb” and “a” Whereas \(ab*)\ matches “ab”, “abababab”, **and the empty string**

Actions:

Awk is a Turing-complete language with support for different data types and structures, control flow, I/O, and user-defined functions. CARL will implement most of the language, but will exclude many of the built-in variables and some, less common, aspects of the language.

Data types and structures

Integers	Supports a single type of integer.
Floating point	Supports a single type of floating-point number.
String	Standard string data type.
Associative arrays	One-dimensional associative array. There is language support for for-loops through the array and adding and deleting elements. Indices of arrays can be either numbers or strings.

There is automatic type conversion between integers and floating point operators, with escalation from integers to floating-point as needed (irreversible).

Control flow

`If` statement	C-style if-then-else controls. `if (expr) { body } else {body}`
`For` statement	C-style for loops. Python-style loops `(element in array)` for associative arrays.
`Break` statement	C-style breaks to exit loops.
`Return` statement	Return from user-defined functions.
User-defined functions	The ability to implement functions

CARL enforces the need for {} around the body of control switches. Only for loops are supported. While and do-while are not supported.

Reserved variables:

`RS`	Variable that stores the literal for separating the input file into multiple records.
`FS`	Variable that stores the literal for separating a record into multiple fields.
`NPAT`	Number of patterns. Used internally.
`__patterns`	Internal variable used with the library.
`__actions`	Internal variable used with the library.

Builtin Functions

`match(`*`string, regex`*`)`	Given a *string*, return the character index of the longest, leftmost substring matched by *regex*. Return 0 otherwise (note that we return 1 if the match occurs at the beginning of the string).
`substr(`*`string, start, length`*`)`	Looks at *string* and returns a *length* character-long substring beginning from *start*. For example, `substr("hello", 2, 3)` returns "`llo`".
`sub(`*`regex, new, target`*`)`	Searches the *target* string for the given *regex*, and replaces it with *new*. We always find the longest, leftmost substring and replace only once. For example, `str = "columbia college"` `sub("col", "example", str)` will set `str` to be "`exampleumbia college`"
`gsub(`*`regex, new, target`*`)`	Like `sub()`, except we replace every occurrence in *target*. Note that the *target* argument is optional, in which case we search the entire input record. Return the number of substitutions made.
`length(`*`string`*`)`	Returns the number of characters in *string*.

split(*string, array, separator)*	Splits the given *string* into pieces using *separator*, and stores each piece in *array*.
	
	For example, `str = "example@email@fake.com"`
	
	`split(str, array, "@")` sets the contents of *array* to be:
	
	`array[0] = "example"`
	
	`array[1] = "email"`
	
	`array[2] = "fake.com"`
	
	string remains unchanged. `split()` returns the number of elements created, so in this case `split() = 3`.

Example programs

Email parsing example:

```
carl '$0, /.*@.*\..*/ { print MF}'
replaces the longer
awk 'match($0, /.*@.*\..*/) { print substr( $0, RSTART, RLENGTH )}'
```

Phone number parsing example:

```
CARL example:
carl'$0, /\([0-9]{3}\)[[:space:]][0-9]{3}\-[0-9]{4}/ { print MF}' filename
```

replaces the longer awk version:

`awk 'match($0, /\([0-9]{3}\)[[:space:]][0-9]{3}\-[0-9]{4}/) { print substr( $0, RSTART, RLENGTH )}' filename.`

Text database parsing example:

Input file:

```
1.      (212)-123-1011 $120,000    Andrew          M
2.      (222)-298-8494 $50,000     Bob             M
3.      (114)-124-1234 $170,000    Katy            F
4.      (123)-222-1111 $90,000     Lisa            F
5.      (123)-132-4666 $80,000     Billy           M
6.      (222)-115-1515 $130,000    Alexa           F
7.      (718)-123-4556 $242,000    Mia             F
```

Code:

```
BEGIN  {
       total=0;
       highEarners=0;
       lowEarners=0;
       females=0;
       males=0;
}
{
       gsub(/\$|,/,"",$3);
       gsub(/\(|\)|-/,"",$2);
       personID=$1;
       phoneNum=$2;
       salary=$3;
       name=$4;
       gender=$5;
       total=total+salary;
       if (salary>=100000) {
              highEarners=highEarners+1;
       } else {
              lowEarners=lowEarners+1;
       }
       if (gender=="F") {
              females=females+1;
       } else if (gender=="M") {
              males=males+1;
       }
       areaCode = substr(phoneNum, 0, 3);
       if (dict[areaCode]) {
              dict[areaCode] = dict[areaCode]+1;
       } else {
              dict[areaCode] = 1;
       }
}
END    {
       print "Total Amount: $"total;{
       average=total/NR;
       print "Average Amount: $"average;
       print "High Earners: "highEarners;
       print "Low Earners: "lowEarners;
```

```
        print "Male to Female Ratio: "males":"females;
        for (i in dict) {
                print "Area code: "i" has "dict[i]" people.";
        }
}
```

Output (by running by "carl -f [code-file] [data-file]":

```
Total Amount: $882000
Average Amount: $126000
High Earners: 4
Low Earners: 3
Male to Female Ratio: 3:4
Area code: 222 has 2 people.
Area code: 114 has 1 people.
Area code: 212 has 1 people.
Area code: 718 has 1 people.
Area code: 123 has 2 people.
```

2. Language Tutorial

Hello World!

This is an example of a "Hello World!" program in CARL. It wraps the main code in a BEGIN block (explained later), initializes a string variable and prints it:

Code:

```
BEGIN  {
        string hello_variable = "Hello World!";
        print_string(hello_variable);
}
```

Using the Compiler

CARL is built in OCAML and uses a C++ engine, compiling a CARL program can be done in 3 simple steps. In the folder where you've unzipped the CARL source code:

1. ./buildcarl → This script compiles the LLVM compiler, C++ engine, and the standard library
2. ./buildcarlp <name of your .carl program> → This script looks for a .carl file located in the folder /tests and compiles a .test file into the /tmp folder. If you want to change this

functionality, just open the buildcarlp script and change the TEST directory to wherever you wish

3. ./tmp/<name of .test file> ./input_file → Every carl program needs to have an input file provided to it, regardless of whether your program actually parses through it. In such cases, a blank file will suffice.

Basics

Primitives

All primitives are declared like C-style variables, i.e. <type> <identifier>. The primitive types that CARL supports are:

1. String
2. float

Arrays

Arrays in CARL are associative arrays (a.k.a hashmaps). Arrays are declared just like other primitives: <type> <identifier>. In CARL, arrays can have two types: array_float or array_string.

Array_float type arrays can only store floats as values, and likewise array_string type arrays can only store strings as values. All arrays use strings as keys.

Array syntax is similar to most conventional array syntax.

- Initializing an empty array:

```
array_float myArray = [];
```

- Adding a <key, value> pair:

```
myArray[“example_key”] = “string literal”;
```

- Retrieving a value:

```
string temp = myArray[“example_key”];
```

Here is an extensive example of how to use CARL arrays:

Code:

```
BEGIN {
    float temp_float;
    array_float myArrayFloat1 = [];
    array_float myArrayFloat2 = [];
    array_string myArrayString1 = [];
    string str = "key1";
    float val = 1;
    myArrayString1["abc"] = "2";
    myArrayString1["def"] = "MEANING OF LIFE?";
    myArrayFloat1[str] = val;
    myArrayFloat2[str] = myArrayFloat1[str] * 42;
}

END {
    string temp_string;
    temp_string = myArrayString1["abc"];
    print_string(temp_string);
    temp_string = myArrayString1["def"];
    print_string(temp_string);

    temp_float = myArrayFloat2[str];
    print_float(temp_float);
}
```

Operators

CARL supports the following binary operators:

- Arithmetic (+ , - , * , /). Note: remember that all arithmetic operations are float operations.
- Comparison, equality, and logical operators (== , != , <, >, >=, <=, ||, &&). || and && work on both floats and strings. Comparison and equality operators only support floats.

CARL supports these unary operators:

- The logical negation operator '!'
- Regex pattern operators: *, ^ (regex negation), ?
- Arithmetic negation '-'

Control Flow

Control Flows behave similar to C-style control flow. CARL supports If, While and For loops. Blocks in control flows are executed from top to bottom.

Example while code:

```
while (float_val1) {
        print_float(float_val1);
        float_val1 = float_val1 - 1;
        float_val1 = slight_increase(float_val1);
}
```

Example if block:

```
if (string_val1) {
        print_string(string_val1);
        while (string_val1) {
                print_string(string_val1);
                string_val1 = "";
        }
  }
```

Pattern Actions

Pattern-Actions are the core of a CARL program. Every CARL program is essentially a list of pattern-actions.

A “pattern” is simply a regex pattern. When CARL is fed an input file, it parses the file into lines. For every pattern in the carl program, the engine performs a regex match on every line of the file. If there is a match on that line, then the CARL engine will perform the action.

Actions are broadly a list of statements. They can be as simple as incrementing a variable, adding to a dictionary, or calling several user-defined functions.

BEGIN and END are two special patterns in CARL. The engine *always* executed whatever code is in BEGIN at the start of the program, and END at the end of the program. Think of them as startup and cleanup operations.

User-Defined Methods

CARL supports user-defined methods. All methods must be defined above the BEGIN block, and are declared with the following syntax:

Function <return_type> <function name> (<arguments_list>)

Once declared and defined, methods can be called from any action. So when the CARL engine finds a matching pattern, it will call the method within that associated action.

3. Language Manual

Final Language Reference Manual

- Citations:
- "The GNU Awk User's Guide, https://www.gnu.org/software/gawk/manual/gawk.html
- Brian Kernigan and Dennis Ritchie, "The C Programming Language," Prentice Hall, 2nd Edition, 1988.

1. Introduction

This language reference manual describes Columbia's Awk Replacement Language (CARL), which, as made evident by the language name, is a revised and simplified implementation of the AWK language. CARL came to fruition as part of a group project for the Columbia course, Programming Languages & Translators taught by Prof. Stephen Edwards. Similar to AWK, CARL uses a `pattern {action}` model to manipulate file data. CARL can scan through files i.e. EXCEL files and use regular expressions to identify data. We will use OCaml to create the compiler for CARL with a target backend of LLVM.

2. Lexical Conventions

2.1 Comments

CARL opens a comment with /* and closes the comment section with */ allowing for multi line and single line comments.

2.2 Identifiers

Identifiers are a sequence of letters, digits, and underscores (_) that explicitly begin with a letter or underscore. There are 52 possible letters, being that they can be uppercase or lowercase and from the ASCII set.

2.3 Keywords

The following identifiers are keywords and are restricted from use otherwise:

`BEGIN, END, if, else, for, while, return, RS, FS, NPAT, function, float, string, array_string, array_float, void, __actions, __patterns, __action[digits]`

The following character sequences are also keywords:

`{, }, [, ], (, ), =, ==, !=, >, <, >=, <=, &&, ||, *, /, +, -, ^, ;, $, $0, $1, $2, …, ,, ‘, “, \, !, |, ?, //`

2.4 Float Constants

All numbers are floats. No decimal point is needed, but can be included. Negative floats are prefixed by a dash (-).

2.5 String Literals

A String consists of a sequence of zero or more chars and is surrounded by double quotes (“. . .”). Strings offer several escape sequences that are represented by chars within the string.

3. Meaning of Identifiers

Identifiers are names that refer to corresponding variables and functions. Refer to section 2.2 for a precise definition of identifiers.

Variable and function identifiers have explicit types.

3.1 Storage Scope

The identifier storage scope is split between local and global. Variables in functions have local scope. Variables in patterns have global scope. All global variables declared in any action are accessible in any other action. Floats are initialized to zero.

4. Conversions

4.1. Rules for Boolean operations

Any non-empty string will be treated as true. Any non-zero float will be treated as true.

4.2. Arithmetic Conversion

Only one numeric data type is supported: float.

5. Rule

Like Awk, CARL programs consists of a series of rules. A rule specifies one pattern to look for, and one action to perform when the pattern is found. Actions are enclosed in braces to separate it from the pattern definition. Rules are separated by newlines. Thus, a series of rules will look like this:

rules:
pattern { action }
pattern { action }
pattern { action }

CARL goes through each pattern and tries each line against the pattern. For each pattern that matches, the action is executed. If no pattern matches, then no action is taken. Once CARL attempts searching through each line, then the next rule is tried.

6. Patterns

Patterns control the execution of actions in a rule -- CARL performs an action on an input record every time it matches a pattern. CARL supports similar types of patterns to awk. These are:

1. Regex Patterns
2. Special BEGIN/END Patterns

6.1 Regex Patterns

A Regex pattern simply contains a regex constant in the pattern definition of a rule. Remember that a rule just looks like this: *Pattern { Action }*

1. Declaring a Regex: A regular expression is always introduced and terminated with 2 forward slashes (`//`).
2. A range pattern is defined by $[X_1\text{-}X_2]$, such that X_1 and X_2 are defined at either uppercase letters, lowercase letters, or digits. X_1 must have a lower value than X_2 on the ASCII table.

3. OR: The operator | will be placed between two patterns. The expression can match to one of the two patterns.
4. Kleene Star: The operator * implies 0 or more occurrences of the immediately preceding pattern.
5. The operator + implies 1 or more occurrences of the immediately preceding pattern.
6. The operator ? implies 0 or 1 occurrences of the immediately preceding pattern.
7. To capture a group of patterns, parenthesis are placed around the group.
8. Concatenation: When regexp follow each other, with no symbol between, they are concatenated regexp.
9. NOT: The symbol ^ is placed before a alphanumeric sequence to imply matching anything but the specified alphanumeric sequence.
10. The . symbol implies any character except line terminators.

So a rule with a regex pattern might look like this:

```
//[A-Z]// { print “wow!” }
```

This rule prints “wow!” for any record that contains a capital letter.

6.2 BEGIN/END

BEGIN and END are keywords reserved to define special patterns. The BEGIN rule is executed once before any input is read, and similarly the END rule is executed after every input is read. Think of them as the startup and cleanup actions for a CARL program.

7. Expressions

Expressions in CARL are very similar to C, being composed of identifiers, literals, grouping with (), function calls, and binary and unary operators. In addition, CARL expressions include array creating and indexing operations.

expression:

(*expression*)
string-literal
float-literal
identifier
expression + *expression*
expression - *expression*
expression * *expression*
expression / *expression*
expression == *expression*
expression != *expression*
expression < *expression*
expression <= *expression*

expression > *expression*
expression >= *expression*
expression && *expression*
expression || *expression*
-*expression*
!*expression*
identifier = *expression*
identifier ($actuals\text{-}list_{opt}$)
[]
identifier [*expression*]
identifier [*expression*]= *expression*

Identifiers are expressions and defined by the rules in Section 2.2. Identifiers are declared. Constants are primary expressions. They are defined in Section 2. Strings are also expression and are enclosed double quotes "`string`". Parenthesized expressions are expressions. Variables cannot change type.

Precedence Table

Grammar	Description	Associativity
(...)	Grouping, which causes the expression within the parentheses to be evaluated before surrounding expressions.	None
Function calls	Function calls are the same as C. Multiple assignment-expressions can be passed to functions. All expressions are fully evaluated (assignment-expressions are at the bottom of the context-free grammar) before passing to the function. No spaces may be between postfix-expression and left parentheses. Arguments are separated by commas. Arguments are fully evaluated and passed by value to the function call. Functions must be defined prior to being called. The types for function called will be explicitly stated. Functions only have the scope of the function code.	Left
!*expression*	Unary NOT negates the logical evaluation of boolean predicates in if, while, for statements. The string or float value evaluated to a boolean value prior to negation. CARL does not have Boolean values, any value greater than zero is considered true and exactly zero and	Right

	negative numbers considered false. Logical negation converts all values of 0 to 1 and all values > 0 to 0. Empty strings are considered false while all other strings are considered true.	
-*expression*	Unary negation returns the opposite signed value of a float. Negation is at the same precedence level as NOT.	Right
Multiplication and division operators (*, /)	Multiplicative operators associate left to right and include * and /. Both operands must be floating point numbers and always results in a floating-point number.	Left
Addition and subtraction operators (+,-)	Additive operators associate left to right and include + and -. They only operate on floating-point numbers and always result in a floating-point number.	Left
Comparion operators (<,<=,>, >=)	The standard comparison operators <, <=, =>, > are available in CARL. Comparisob operators are evaluated left to right. The relational operators apply only to floating point numbers. The operators return 1 if true and 0 if false. Comparison operators can only be used in `if`, `for`, and `while` predicate statements.	Left
Equality operators (==, !=)	The standard equality operators <, <=, =>, > are available in CARL. Equality operators are evaluated left to right. The equality operators apply only to floating point numbers. The operators return 1 if true and 0 if false. Comparison operators can only be used in `if`, `for`, and `while` predicate statements.	Left
Logical AND (&&)	The logical AND operator uses the symbol &&. Logical AND returns true, if and only if both expressions are true.	Left

	The left-side expression is evaluated first. There are no bitwise operators as in C. The operator is evaluated from left to right.	
Logical OR (||)	The logical OR operator uses ||. The logical OR returns true if either expression is true. There are no bitwise operators as in C.	Left
Assignment (=)	Stores the result of the right side expression into the left side expression (typically an identifier). Operations are evaluated from right to left, so a = b = 4 is equivalent to a=(b=4).	Right
Array assignment = (*identifier*[*expr*]=)	Stores the result of the right side expression into the left side array with the identifier key. Array references require a string key, but the key can be formed through another expression. The key is hashed as part of the associative array.	Right
Array references [“key”]	Returns the stored value identified by the key from the array. Array references require a string key, but the key can be formed through another expression. The key is hashed as part of the associative array.	Left
Array creation ([])	Creates a new array with the given identifier.	Left
Identifiers	Identifiers are primary expressions and defined by the rules in Section 2.2. Identifiers are declared.	N/A
String literals	String literals are defined in Section 2 and composed of strings surrounded by double quotes. For example “string_literal” is a valid string literal.	N/A
Float literals	Float literals are numeric data types. They can be negative and contain an optional decimal point (.). Examples include 5, -5, -5.0, 5.0.	N/A

8. Arrays

8.1. Declaration and Initialization

Like AWK, all arrays are actually hash tables or associative arrays. Arrays are declared with the type of value they store. String and float storage cannot be mixed. Arrays cannot store other arrays. We declare an array using the following syntax:

```
array_string myarray;

array_float myarray;
```

Arrays must be initialized using the array create expression ([]). Created arrays can be assigned to declared arrays during declaration or after declaration.

8.2. Storage

New entries are added to the array by specifying the array identifier followed by brackets containing the new or existing key, followed by assignment to the value:

```
myArray[“new-key”] = “value”;

myArray[“new-key”] = “new value”;
```

New assignments to the array result in the original value being overwritten.

8.3 Retrieval

Entries can be retrieved by referencing the key with the array identifier. The array value can then be used as any other value. Example syntax is as follows:

```
a = myarray[‘old-key”];
```

9. Declarations

A declaration introduces new entities into the program and identifies them. The type of the entity is explicitly typed when the identifier is first introduced.

9.1 Variables

Variables may only be declared at the beginning of actions and functions. Variables and statements may not be mixed. Variables declared within actions are globally scoped. Variables declared in functions are locally scoped. Variables may be initialized or not at declaration.

Floating point numbers:

A floating point type will be specified with “float” and contains a decimal within (-)[0-9]*.[0-9]* with the possibility of being negative.

```
float my_float;

float my_float = 4.123;
```

Strings:

A string type will be specified with “string” and has opening and closing of two quotation or two apostrophe characters.

```
string my_language;

string my_language = “CARL”;
```

Associative arrays:

These are defined in Section 8.

```
array_string myarray;
```

9.2 Functions:

Functions:

A function’s return type will be specified as one of the data types mentioned in Section 9.1. It is wrapped in parenthesis and curly brackets wrapping the function’s contents. A function is declared with “function”, its return type, function arguments, and function body. Function declarations must precede all rule declarations.

```
function int my_func(value1, value2){
        Float new_value = value1 + value2;
        return new_value;
}
My_variable = my_func(val1, val2);
```

10. Statements

Statements are executed in order from top to bottom, left to right.

10.1 Expression Statements:

Expression statements execute an expression, which often includes an assignment. Expression statements end in a semicolon.

```
expression;
```

10.2 Conditional Statements:

Conditional statements evaluate expressions and execute other statements based on if the expression is true or false.

```
if ( expression ) { statement }
if ( expression ) { statement } else { statement }
```

10.3 For Statements:

For statements execute the specified body statement based on a condition. “expression1” is used for initialization. “expression2” is used as a condition, “expression3” is used as an increment. The body statement will continue re-executing as long as the condition set in expression2 is true.

```
for ( expression1 ;  expression2 ; expression3 ) {statement}
```

10.4 While Statements:

For statements execute the specified body statement based on a condition. The body statement will continue re-executing as long as the condition set in expression is true.

```
while ( expression ) statement
```

10.5 Return Statements:

A return statement will return from a function. A return statement can either return nothing or return a value from an expression.

```
return;

return expression;
```

11. Scope

11.1 Variable Scope In Functions:

Global:

Identifiers assigned in actions can be used globally in functions. Functions that re-assign a variable will change it globally. Functions can declare local variables with the same identifier so they don't always have to change the variable globally. Example:

```
void function my_func() {

        i = 6;

        print "i func is equal to: "i;

}

int i = 5;

my_func();

print "i global is equal to: "i;
```

Results:

```
i func is equal to: 6

i global is equal to: 6
```

11.2 Variable Scope In Action:

All variables declared within actions are global. Float are initialized to zero while strings are initialized to null.

12. Grammar

program:

function-declaration-list$_{opt}$ *rule-list*$_{opt}$ *EOF*

function-declaration-list:
function-declaration-list function-declaration

function-declaration:
`function` *type identifier* (*formal-list*$_{opt}$) {*variable-declaration-list*$_{opt}$ *statement-list*$_{opt}$}

type: one of
`void string float array_float array_string`

formal-list:
type identifier
formal-list, type identifier

variable-declaration-list:
variable-declaration-list variable-declaration

variable-declaration:
type identifier assign$_{opt}$;

assign$_{opt}$*:*
= *expression*

rule-list:
rule-list rule

rule:
pattern action

pattern:
BEGIN
END
//*regex-sequence*//

action:
{*variable-declaration-list*$_{opt}$ *statement-list*$_{opt}$}

statement-list:
statement-list statement

statement:
expression;

`return` *expr*;
{*statement-list*}
`if` (*expression*) *statement*
`if` (*expression*) *statement* `else` *statement*
`for`(*expression*$_{opt}$; *expression*; *expression*$_{opt}$) *statement*
`while` (*expression*) *statement*

expression:
string-literal
float-literal
identifier
expression + *expression*
expression - *expression*
expression * *expression*
expression / *expression*
expression == *expression*
expression != *expression*
expression < *expression*
expression <= *expression*
expression > *expression*
expression >= *expression*
expression && *expression*
expression || *expression*
-*expression*
!*expression*
identifier = *expression*
identifier (*actuals-list*$_{opt}$)
(*expression*)
[]
identifier [*expression*]
identifier [*expression*]= *expression*

actuals-list:
expression
actuals_list, *expression*

REGEX Grammar:

regex-sequence:
regex
regex regex-sequence

regex:

regex-literal
.
(*regex-sequence*)
[*regex-set-sequence*]
regex?
regex+
*regex**
regex|

regex-set:

regex-literal
regex-literal - regex-literal
^ *regex-set*
{*regex-set-sequence*}

regex-set-sequence:

regex-set
regex-set regex-set-sequence

Original Language Reference Manual

Columbia's Awk Replacement Language (CARL)

Language Reference Manual

COMS 4115 Programming Languages and Translators (Spring 2017)

Darren Hakimi (dh2834)	System Architect
Keir Lauritzen (kcl2143)	Manager
Leon Song (ls3233)	Tester
Guy Yardeni (gy2241)	Language Guru

- Citations:
- "The GNU Awk User's Guide, https://www.gnu.org/software/gawk/manual/gawk.html
- Brian Kernigan and Dennis Ritchie, "The C Programming Language," Prentice Hall, 2nd Edition, 1988.

1. Introduction

This language reference manual describes Columbia's Awk Replacement Language (CARL), which, as made evident by the language name, is an implementation of the AWK language. CARL came to fruition as part of a group project for the Columbia course, Programming

Languages & Translators taught by Stephen Edwards. The initial purpose behind the creation of CARL is for it to be used as a web scraper. Similar to AWK, CARL uses a `pattern {action}` model to manipulate file data. CARL can scan through files i.e. HTML files and use regular expressions to identify data. We will use OCaml to create the compiler for CARL with a target backend of LLVM.

2. Lexical Conventions

2.1 Comments

CARL only consists of single line comments. Comments are introduced by `#`, with no intervening blanks, and is terminated by the end of the line. Therefore, CARL ignores the remainder of the liner after the comment symbol.

2.2 Identifiers

Identifiers are a sequence of letters, digits, and underscores (_) that explicitly begin with a letter or underscore. There are 52 possible letters, being that they can be uppercase or lowercase and from the ASCII set.

2.3 Keywords

The following identifiers are keywords and are restricted from use otherwise:

`BEGIN, END, if, elseif, else, for, for..in, break, continue, return, RS, FS, NR, NF, RSTART, RLENGTH, MF, in, function`

The following character sequences are also keywords:

`{, }, [, ], (, ), =, ==, != +=, -=, *=, %=, ^=, **==, >, <, >=, <=, &&, ||, *, /, %, +, -, ^, **, ;, $, $0, $1, $2, …, ,, ‘, “, \, ~, !~, !`

2.4 Integer Constants

Integers consist of a sequence of digits, 0-9, that do not begin with 0. Integers will all be base 10. There will not be any built-in conversions to any other bases, such as octal or hex. Negative integers are prefixed by a dash (-).

2.5 Float Constants

Floats consist of an integer component followed by a decimal point (.) and a fraction component. The integer and fraction component both consist of a set of digits, 0-9. Either the integer component or the fraction component could be omitted. If the integer component is omitted, the mandatory fraction component must be preceded by the decimal point.

2.6 String Literals

A String consists of a sequence of zero or more chars and is surrounded by double quotes (“. . .”). Strings offer several escape sequences that are represented by chars within the string:

Newline (`\n`), tab (`\t`), double quote (`\”`), single quote (`\’`), backslash (`\`)

2.7 Regexp Constants

A regexp consists of expressions that are used for pattern matching. A regular expression is always introduced and terminated with a forward slash (`/`). A range pattern is defined by $[X_1\text{-}X_2]$, such that X_1 and X_2 are defined at either uppercase letters, lowercase letters, or digits. X_1 must have a lower value than X_2 on the ASCII table. Regular Expressions can be compared using the `~` and `!~` operators to respectively determine if and if not a regexp matches another.

3. Syntax Notation

The syntax notation of this reference manual for all code, including words, characters, regular expressions and keywords, will be typed in the `Inconsolata` font. Code formatting and notation will be represented with *dark grey and italicized Times New Roman* font.

4. Meaning of Identifiers

Identifiers are names that refer to corresponding variables, functions, or actions. Refer to section 2.2 for a precise definition of identifiers.

Variable and function identifiers have inferred types, which implies that identifiers will not have type declarations and that the compiler will determine the type.

4.1 Storage Scope

The identifier storage scope is split between local and global. Local scope identifiers are declared inside of code blocks. Local identifiers can only be accessed within the block they are declared. Global scope identifiers are initialized specifically within a BEGIN or END block. Global identifiers can be accessed anywhere in the code that is below the declaration of the identifier. Functions will normally be global scope, but if the function parameter contains the variable with

The identifier storage scope is entirely global. All identifiers are initialized specifically within a BEGIN or END block. Global identifiers can be accessed anywhere in the code that is below the declaration of the identifier.

5. Objects and Lvalues

An Object is a named storage location and a lvalue is a reference to an object. In CARL, the lvalue is on the left hand side of the declaration and represented by a variable or array element. The objects types in CARL are integer, float, string, associative array, and functions.

6. Conversions

6.1. String to Integer Conversion

CARL will support string-to-integer conversion, **but not integer-to-string** conversions.

CARL converts strings to numbers by interpreting any numeric prefix of the string as numerals e.g. str = “8371this is an example” will convert str to the integer 8371. CARL will also support scientific notation conversions as well e.g. the string “1e3” converts to the integer 1000.

6.2. Arithmetic Conversion

Only two numeric data types are supported: int and float. For arithmetic operations, if one operand is a float, the other operand will be converted to a float as well

7. Rule

Like Awk, CARL programs consists of a series of rules. A rule specifies one pattern to look for, and one action to perform when the pattern is found. Actions are enclosed in braces to separate it from the pattern definition. Rules are separated by newlines. Thus, a series of rules will look like this:

rule:
pattern { action }
pattern { action }
pattern { action }

CARL reads the input file one line at a time. For each line, CARL looks for the patterns in each rule, trying patterns in the order that they were written. If no pattern matches, then no action is taken. Once CARL attempts searching for every pattern, it moves on to the next line.

8. Patterns

Patterns control the execution of actions in a rule -- CARL performs an action on an input record every time it matches a pattern. CARL supports similar types of patterns to awk. These are:

1. Regex Patterns
2. Expression Patterns
3. Range Patterns
4. Special BEGIN/END Patterns
5. Empty Pattern

8.1 Regex Patterns

A Regex pattern simply contains a regex constant in the pattern definition of a rule. Remember that a rule just looks like this: *Pattern { Action }*

So a rule with a regex pattern might look like this:

```
/[A-Z]/ { print “wow!” }
```

This rule prints “wow!” for any record that contains a capital letter.

8.2 Expression Patterns

CARL expressions are valid as patterns. An expression is a match if it’s value evaluates to non-zero (if it is a number) or non-null (if it is a string). The expression is reevaluated every time CARL tests a new record. Expressions can use variables or fields from the input record (e.g. $3).

Comparison expressions are a common pattern. Consider the following:

```
$2 == /[A-Z]/ { print “wow! }
```

This rule only looks for capital letters in the second field of the record, and prints “wow!” if the pattern matches.

Boolean expressions such as this one are very useful as well:

```
/columbia/ && /.edu/ { print “student from columbia” }
```

This rule makes sure the record contains both the words “columbia” and “edu” before executing the action.

8.3 Range patterns

A range pattern is defined by two patterns, separated by a comma.

```
beginpat , endpat { Action}
```

We use range patterns when we want to match a range of consecutive records. When a record matches *beginpat*, the range pattern is turned on, and CARL performs the action for every consecutive record. CARL continues scanning records until it finds a record that matches *endpat*, then it turns the range pattern off.

Note that matching for range patterns is *inclusive*, meaning that the records that match *beginpat* and *endpat* will execute the action.

You can use range patterns alongside other rules as well. Consider the following:

```
/[A - Z]/ { print “capital letter found” }

/hello/ , /goodbye/ { print “range is ON” }

/[0-9]/ { print “numeric character found” }
```

This is a series of three rules. Two of the rules use simple regex patterns, while the middle rule uses a range pattern.

CARL will try to match these three patterns for every record. Once it finds a record with “hello”, it turns the range pattern on, and prints “range is ON” for every record until it comes across another record with “goodbye”. As CARL is doing this, it continues to match patterns from the other rules as well i.e. look for any capital letters or numbers in every subsequent record while still searching for “goodbye”. Note that only after “goodbye” is found does CARL try to search for “hello” again.

It is also possible for a range pattern to be turned on and off by the **same** record, in which case the action is executed for that record only.

8.4 BEGIN/END

BEGIN and END are keywords reserved to define special patterns. The BEGIN rule is executed once before any input is read, and similarly the END rule is executed after every input is read. Think of them as the startup and cleanup actions for a CARL program.

CARL allows for multiple BEGIN and END rules, running them in the order which they are written. Note that BEGIN and END cannot be used with any operators or expressions.

8.4.1 I/O issues when using BEGIN/END

Remember that BEGIN actions run before any input is read, so variables like $0 are undefined because there are no input records, and hence no fields yet.

8.5 The Empty Pattern

The empty pattern matches **every** input record. You can define it by declaring an action without a corresponding pattern e.g. simply do:

{ print $1 }

Which prints the first field in every record.

9. Expressions

9.1. Primary Expressions

Primary expression are the same as in C: identifiers, constants, strings, and (expressions).

primary-expressions:
identifier
constant
(expression)

Identifiers are primary expressions and defined by the rules in Section 2.2. Identifiers are not declared, rather they are created when initialized using the assignment operator. Constants are primary expressions. They are defined in Section 2. Strings are also primary expression and are enclosed double quotes "`string`". Parenthesized expressions are primary expressions.

Regex constants are also primary expressions. Regex

9.2. Prefix Field Reference

When CARL processes a file and record matches the pattern, they the record is further parsed into fields based on a field separator.

prefix-field-expression:
primary-expression
$prefix-field-expression

The field separator is nominally whitespace, but can be assigned at run time. The fields can then be addressed positionally using a field reference. Field references are ones-indexed and $0 is reserved for the entire record. Field reference must be numeric expressions.

9.3. Postfix Expressions

Postfix expressions are as follows and all operators are left associative:

postfix-expression:
prefix-field-expression
postfix-expression[expression]
postfix-expression(argument-expression-list)
postfix-expression++
postfix-expression--
argument-expression-list:
assignment-expression
argument-expression-list, assignment-expression

9.3.1. Array References

Array references can be any expression, but numeric expressions will be converted to strings prior to hashing in the associative array.

9.3.2. Function Calls

Function calls are the same as C. Multiple assignment-expressions can be passed to functions. All expressions are fully evaluated (assignment-expressions are at the bottom of the context-free grammar) before passing to the function. No spaces may be between postfix-expression and left parentheses. Arguments are separated by commas.

Arguments are fully evaluated and passed by value to the function call. Functions must be defined prior to being called. The types for function called will be inferred based on the operations of the function as defined in Section 10. Functions only have the scope of the function code.

9.3.3. Postfix Incrementation

Postfix incrementation is the same as C. With a postfix incrementor, the value is provided then incremented. This precedence differs slightly from GAWK.

9.4. Prefix Incrementation

Prefix incrementor operators function the same as C. The value is incremented by one and then returned.

prefix-inc-expression:
postfix-expression
++prefix-inc-expression
--prefix-inc-expression

9.5. Exponentiation Operators

CARL supports exponentiation using both ** and ^, although ^ is recommended. Exponentiation operators group right to left.

exponentiation-expression:
prefix-inc-expression
*prefix-inc-expression ** prefix-inc-expression*
prefix-inc-expression ^ prefix-inc-expression

Exponentiation requires both operands to be numerics.

9.6. Unary Operators

Unary operators are right associative and are as follows:

unary-expression:
exponentiation-expression
unary-operator unary- expression
unary-operator: one of
+ - !

Unary-expressions must be numeric for all operators.

9.6.1. Unary Plus Operator

This does nothing but it included for symmetry with minus operator.

9.6.2. Unary Minus Operator

Changes the sign of the unary-expression. Operand must be numeric.

9.6.3. Logical Negation Operator (!)

CARL does not have Boolean values, like C any value greater than zero is considered true and only exactly zero is considered false. Logical negation converts all values of 0 to 1 and all values > 0 to 0. Empty strings are considered false while all other strings are considered true.

9.7. Multiplicative Operators

Multiplicative operators associate left to right and include *, /, and %. Division results in integer to floating point conversion unless both operands are constants and the result is an integer. Modulus automatically results in an integer. Multiplication of two integers results in an integer, while multiplication of one or more floating points results in a floating point.

multiplicative-expression:
unary-expression
*multiplicative-expression * unary-expression*
multiplicative-expression / unary-expression
multiplicative-expression % unary-expression

Expressions must be numeric.

9.8. Additive Operators

Additive operators associate left or right and include + and -.

additive-expression:
multiplicative-expression
additive-expression + multiplicative-expression
additive-expression - multiplicative-expression

9.9. String Concatenation

String concatenation occurs when two strings are separated by a ‘ ‘. The string is formed from right to left.

string-concat-expression:
additive-expression
string-concat-expression string-concat-expression

Both expressions must be strings or numerics with implicit conversion to strings. Strings are concatenated into a single string.

9.10. Relational and Equality Operators

The standard relational operators <, <=, ==, =>, >, != are available in CARL. Relational operators are evaluated left to right.

relational-expression:
string-concat-expression
relational-expression relational-operator string-concat-expression
relational-operator: one of
<, <=, ==, =>, >, !=

The relational operators apply to both numeric values and string. The operators return 1 if true and 0 if false. Strings are compared in lexicographical order character by character. Comparison between floating-point numbers and integers results in the conversion of the integer to a floating-point number.

The greater-than symbol (>) is also used for redirection for print statements. The context is determined automatically.

9.11. Matching Operators

Matching operators are used to compare strings to regexp constants. If the regular expression matches the string then the expression returns a 1, otherwise it returns a 0.

matching-expression:
relational-expression
matching-expression ~ string-concat-expression
matching-expression !~ string-concat-expression

One of the operands must be a regexp constant while the other must be a string. Operators are left associative.

9.12. Array membership

Associative arrays member is tested using the keyword in: `key in array`, where key is a potential key and array is an associative array. The operator is evaluated left to right.

array-member-expression:
matching-expression
string-concat-expression in primary-expression

Array names cannot be built from strings and numerics for the purpose of membership tests. The array name must be explicitly defined.

9.13. Logical AND Operator (&&)

The logical AND operator uses the symbol &&. Logical AND returns true, if and only if both expressions are true.

logical-and-expression:

array-member-expression

logical-and-expression && array-member-expression

The left-side expression is evaluated first. If it is 0, then the right-side expression is never evaluated and the expression returns 0 (the operator is said to be short circuited.) If any side effect behavior from the left side would occur (for example, ++), then this side effect does not occur. If the left-side expression is true (numeric greater than 0, non-empty string), then the right-side expression is evaluated. The are no bitwise operators as in C. The operator is evaluated from left to right.

9.14. Logical OR Operator (||)

The logical OR operator uses ||. The logical OR returns true if either expression is true.

logical-or-expression:

logical-and-expression

logical-or-expression || logical-and-expression

Logical OR operators are short-circuited in the same was as logical AND operators, except that if the left-side operator is true, then it automatically returns 1. Operations are evaluated from left to right.

9.15. Conditional Operator

CARL supports conditional operators (short form if statements). These function the same as C.

conditional-expression:

logical-or-expression

logical-or-expression ? expression : conditional-expression

The logical-or-expression is evaluated and then either expression or conditional expression is chosen. Like GAWK, these group right to left.

9.16. Assignment Expressions

Assignments must be to prefix-inc-expressions. Operations are evaluated from right to left, so a = b = 4 is equivalent to a=(b=4).

expression:

conditional-expression

prefix-inc-expression assignment-operator expression

assignment-operator: one of

=, +=, -=, *=, %=, ^=, **=

Assignment operators complete the operation in the operator using the value of the left-side and the value of the right side then storing it back in the left hand operator. “/=” has been removed

to eliminate the ambiguity with regular expressions. Assignment operators return their assigned value.

9.17. Print Operator

The `print` and `printf` functions in CARL do not use parentheses, unlike C but like GAWK. This design decision was due to the common use of printing statements. They can print multiple expressions that are separated by commas. Expressions are evaluated left to right and printed in order.

print-expression:
expression
print print-expression-list
printf string-concat-expression, print-expression-list
print-expression-list:
expression
print-expression-list, expression

`Print` expressions contain only comma-separated expressions and each is printed in turn. The first argument after `printf` must be a string.

10. Declarations

A declaration introduces new entities into the program and identifies them. The type of the entity is inferred based on the value of the entity upon assignment.

The following entities are available for declaration:

10.1 Data types:

Integers:

An integer type will be inferred by having a value within (-)[0-9]+ with the possibility of being negative.

Example: my_int = 5

Floating points:

A floating point type will be inferred by containing a decimal within (-)[0-9]*.[0-9]* with the possibility of being negative.

Example: my_float = 4.123

Strings:

A string type will be inferred by the opening and closing of two quotation or two apostrophe characters.

Example: my_language = “CARL”

Associative arrays:

An associative array type is inferred through curly brackets. An associative array can either be one with integers for indices or with strings for indices.

Example: my_array = {1 : “C”, 2 : “A”, 3 : “R”, 4 : “L”}

10.2 Functions:

Functions:

A Function type is inferred by being wrapped in parenthesis and curly brackets wrapping the function's contents.

```
Example: function my_func(value1, value2)

{

new_value = value1 + value2;

return new_value;

}

My_variable = myfunc(arguments);
```

11. Statements

Statements are executed in order from top to bottom, left to right.

11.1 Assignment Statements:

Assignment statements specify an identifier and its equality, ending in semicolon.

```
Identifier = value;
```

11.2 Conditional Statements:

Conditional statements evaluate expressions and execute other statements based on if the expression is true or false.

```
if ( expression ) { statement }

if ( expression ) { statement } else {statement}

if ( expression ) { statement } elseif ( expression ) {statement}
else {statement}
```

11.3 For Statements:

For statements execute the specified body statement based on a condition. "expression1" is used for initialization. "expression2" is used as a condition, "expression3" is used as an

increment. The body statement will continue re-executing as long as the condition set in expression2 is true.

```
for ( expression1 ;  expression2 ; expression3 ) { statement }
```

11.4 Break Statements:

A Break statement will terminate the enclosing For statement. It is usually used in a For statement when a certain condition occurs.

```
break;
```

11.5 Continue Statements:

A continue statement will jump to the next cycle of a For statement, skipping the rest of the body of the For statement below it. It is usually used in a For statement when a certain condition occurs.

```
continue;
```

11.6 Return Statements:

A return statement will return from a function. A return statement can either return nothing or return a value from an expression.

```
return;

return {expression};
```

12. Scope

12.1 Variable Scope In Functions:

Global:

Variable scope for functions in CARL is global.Initialized variables used in block statements or functions and modified within will have their value changed on a global level.

Example:

```
function my_func() {
        i = 6;
        print "i func is equal to: "i;
}
i = 5;
my_func();
print "i global is equal to: "i;
```

Results:

```
i func is equal to: 6
i global is equal to: 6
```

Local:

In order to keep the value of a variable unchanged on the global level when it is used in a function, it can be passed as an argument to the function and prepended with with a pipe character.

Example:

```
function my_func(|i) {
        i = 6;
        print "i func is equal to: "i;
```

```
        }

        i = 5;

        my_func();

        print "i global is equal to: "i;
```

Results:

```
        i func is equal to: 6

        i global is equal to: 5
```

12.2 Variable Scope In Action:

Variables declared within the BEGIN and END sections are global, while variables declared in between them are local.

13. Grammar

Basic-unit:
- *Declaration*
- *Rule*

Declaration:
- *function-definition*
- *variable-definition*

Function-definition:
- *declaration*
- *declaration-list*
- *statement*
- *parameter-list*

Rule:
- *Pattern - { Action }*

Pattern:
- *BEGIN/END*
- *Empty Pattern*
- *Regex Pattern*
- *Expression Pattern*
- *Range Pattern*

Action:
- *Statement*
- *Function-call*

Statement:

print-expression

Assignment Statement

Conditional Statement

For statement

Return statements

Break Statements

Continue Statements

print-expression:

expression

print print-expression-list

printf string-concat-expression, print-expression-list

print-expression-list:

expression

print-expression-list, expression

expression:

conditional-expression

prefix-inc-expression assignment-operator expression

assignment-operator: one of

*=, +=, -=, *=, %=, ^=, **=*

conditional-expression:

logical-or-expression

logical-or-expression ? expression : conditional-expression

logical-or-expression:

logical-and-expression

logical-or-expression || logical-and-expression

Logical-and-expression:

array-member-expression

logical-and-expression && array-member-expression

array-member-expression:

matching-expression

string-concat-expression in primary-expression

matching-expression:

relational-expression

matching-expression ~ string-concat-expression

matching-expression !~ string-concat-expression

relational-expression:

string-concat-expression

relational-expression relational-operator string-concat-expression

relational-operator: one of

<, <=, ==, =>, >, !=

string-concat-expression:

additive-expression
string-concat-expression string-concat-expression

Additive-expression:
multiplicative-expression
additive-expression + multiplicative-expression
additive-expression - multiplicative-expression

Multiplicative-expression:
Unary-expression
*multiplicative-expression * unary-expression*
multiplicative-expression / unary-expression
multiplicative-expression % unary-expression

unary-expression:
exponentiation-expression
unary-operator unary- expression

unary-operator: one of
+ - !

prefix-inc-expression:
postfix-expression
++prefix-inc-expression
--prefix-inc-expression

postfix-expression:
prefix-field-expression
postfix-expression[expression]
postfix-expression(argument-expression-list)
postfix-expression++
postfix-expression--

primary-expressions:
identifier
(expression)

argument-expression-list:
expression
argument-expression-list, expression

Identifier:
Constant
Function-definition

Constant:
integer-constant
float-constant
Regexp-constant
string constant

14. Examples

15.1 Email parsing example:

```
carl '$0, /.*@.*\..*/ { print MF}'
```

replaces the longer

```
awk 'match($0, /.*@.*\..*/) { print substr( $0, RSTART, RLENGTH )}'
```

15.2 Phone number parsing example:

```
carl'$0, /\([0-9]{3}\)[[:space:]][0-9]{3}\-[0-9]{4}/ { print MF}'
filename
```

replaces the longer awk version:

```
awk 'match($0, /\([0-9]{3}\)[[:space:]][0-9]{3}\-[0-9]{4}/) { print
substr( $0, RSTART, RLENGTH )}' filename.
```

15.3 Text database parsing example:

Input file:

```
1.     (212)-123-1011      $120,000    Andrew        M
2.     (222)-298-8494      $50,000     Bob           M
3.     (114)-124-1234      $170,000    Katy          F
4.     (123)-222-1111      $90,000     Lisa          F
5.     (123)-132-4666      $80,000     Billy         M
6.     (222)-115-1515      $130,000    Alexa         F
7.     (718)-123-4556      $242,000    Mia           F
```

Code:

```
BEGIN  {
       total=0;
       highEarners=0;
       lowEarners=0;
       females=0;
       males=0;
}
{
       gsub(/\$|,/,"",$3);
       gsub(/\(|\)|-/,"",$2);
       personID=$1;
       phoneNum=$2;
       salary=$3;
       name=$4;
       gender=$5;
       total=total+salary;
       if (salary>=100000) {
              highEarners=highEarners+1;
       } else {
              lowEarners=lowEarners+1;
       }
       if (gender=="F") {
              females=females+1;
       } elseif (gender=="M") {
              males=males+1;
       }
       areaCode = substr(phoneNum, 0, 3);
       if (dict[areaCode]) {
              dict[areaCode] = dict[areaCode]+1;
       } else {
              dict[areaCode] = 1;
       }
}
END    {
```

```
        print "Total Amount: $"total;
        average=total/NR;
        print "Average Amount: $"average;
        print "High Earners: "highEarners;
        print "Low Earners: "lowEarners;
        print "Male to Female Ratio: "males":"females;
        for (i in dict) {
               print "Area code: "i" has "dict[i]" people.";
        }
}
```

Output (by running by "carl -f [code-file] [data-file]":

```
Total Amount: $882000
Average Amount: $126000
High Earners: 4
Low Earners: 3
Male to Female Ratio: 3:4
Area code: 222 has 2 people.
Area code: 114 has 1 people.
Area code: 212 has 1 people.
Area code: 718 has 1 people.
Area code: 123 has 2 people.
```

Notes:

Changes from GAWK:

1. Removing the /= arithmetic assignment operator, which is ambiguous with REGEX.
2. C precedence is used for incrementors, where postfix has a higher precedence than prefix. In GAWK they are equal precedence.
3. All numbers in GAWK are floating point numbers. CARL will have Integer type and floating point type and the type will be infered
4. No bitwise comparison operators | and |&
5. CARL will allow the usage of elseif in an if-else statement.
6. CARL will allow only For loop statements.
7. CARL will allow the construct
8. CARL allows variables to be local to functions without them being changed globally by prepending the variable argument with | whereas AWK would conventionally prepend the variable argument with an extra space.

4. Project Plan

Planning, Specification, Development and Testing

- Continuous discussion and planning using Slack.
- Interactive Google Docs for simultaneously editing LRM and Report.
- Weekly status meetings with TA.
- Frequent coding sessions in CS Lounge.
- Often use Test Driven Development approach when working on adding functionality by adding the tests first, and then writing the code to make them pass.

Programming Style Guide

- Strive for no tabs
- Indent with 4 spaces
- Number of columns per line (width of page) should not exceed an exorbitant amount. Approximately 100 columns.
- No binary files committed to git
- Use build scripts
- Write tests for new functionality and then add the code to make tests pass (TDD)
- Run testall script before committing new code in order to ensure that new code does not break existing functionality
- Avoid copy-pasting code
- Compile using buildcarl executable script
- Generate test executable using buildcarlp executable script

Timeline

- February - Proposal, Design, LRM
- March - Set up environment, MicroC, Scanner, Parser, S/R R/R Conflicts
- April - Tests, Test scripts, Build Scripts, Floats, Associative Arrays, Engine, Regex, Strings, Semantic check
- May - Functions, Control flow, Associative Arrays syntax,

Roles and Responsibilities

- Darren Hakimi (dh2834) - Tester
- Keir Lauritzen (kcl2143) - Manager
- Leon Song (ls3233) - System Architect
- Guy Yardeni (gy2241) - Language Guru

Software Development Environment (Tools and Languages)

- VirtualBox Ubuntu VM
- Git/Github
- Slack
- Sublime Text Editor
- MicroC Start
- OCaml
- Llvm
- C
- C++
- Bash

Project Log

```
commit 7dfd23ea247e587f116da1e57a4a9d7ba173012b
Author: Darren Hakimi <dh2834@columbia.edu>
Date:   Wed May 10 19:02:14 2017 -0400

    signed buildcarl, buildcarlc and testall files

commit 6be13093357a566071f11c1e3fc725411ece4b01
Author: Leon Song <ls3233@columbia.edu>
Date:   Wed May 10 10:28:38 2017 -0400

    Leon's signature

commit bc6f3293ca7bf5fced118c95853b6c3a348f2651
Author: keirl <keirl@users.noreply.github.com>
Date:   Wed May 10 10:13:20 2017 -0400

    Update parser.mly

commit 5275c5e0d6a8d8a16b6512b4e8934c755723a58a
Author: keirl <keirl@users.noreply.github.com>
Date:   Wed May 10 09:46:29 2017 -0400

    Update ast.ml
```

```
commit f221b54d5e2ba18c008957c7a7ca6193d8077de2
Author: Darren Hakimi <dh2834@columbia.edu>
Date:   Tue May 9 23:53:45 2017 -0400

    signed test .carl files

commit 45f52f0bc4f9594498268ce7c390514eda0a7234
Author: keirl <keirl@users.noreply.github.com>
Date:   Tue May 9 22:59:10 2017 -0400

    Update scanner.mll

commit f22670a4473d809de7b90b27f3409b6925b6cdad
Author: keirl <keirl@users.noreply.github.com>
Date:   Tue May 9 22:53:25 2017 -0400

    Update scanner.mll

commit 6100efc50d1e24ccdf8cc7917bbf6531aaa05cbd
Author: keirl <keirl@users.noreply.github.com>
Date:   Tue May 9 22:26:34 2017 -0400

    Update semant.ml

commit a2a4cd5fe6de79539b62582a02cac40fe8a2c295
Author: keirl <keirl@users.noreply.github.com>
Date:   Tue May 9 22:25:49 2017 -0400

    Update scanner.mll

commit e3a835c9b49a7a8688b527e458b0158d85ab786e
Author: keirl <keirl@users.noreply.github.com>
Date:   Tue May 9 22:25:11 2017 -0400

    Update sast.ml

commit 6dd34341422c08be46ce3bab814ebd72fea19149
Author: keirl <keirl@users.noreply.github.com>
Date:   Tue May 9 22:24:21 2017 -0400

    Update parser.mly

commit d2e59f6fd3250a3ab8a822e48c5dd17c957e68d3
Author: keirl <keirl@users.noreply.github.com>
Date:   Tue May 9 22:23:29 2017 -0400

    Update codegen.ml

commit 39e47270feb5dee1ac07f8d404e954cb5e004e3a
Author: keirl <keirl@users.noreply.github.com>
Date:   Tue May 9 22:21:50 2017 -0400

    Update check.ml
```

```
commit a369f6dbdeeee2be017db003364132c86a7fd807
Author: keirl <keirl@users.noreply.github.com>
Date:   Tue May 9 22:21:20 2017 -0400

    Update ast.ml

commit 156b74c8e4bcc758a7dba0fada8883805843f82e
Author: keirl <keirl@users.noreply.github.com>
Date:   Tue May 9 22:20:51 2017 -0400

    Update ast.ml

commit 9ee5dd062f59ad6153022b44a9bd586241cca3f1
Author: keirl <keirl@users.noreply.github.com>
Date:   Tue May 9 22:19:58 2017 -0400

    Update Makefile

commit 84b4802fc4a1a7a1eb45a7fab96e5572a3fdd824
Author: keirl <keirl@users.noreply.github.com>
Date:   Tue May 9 22:19:39 2017 -0400

    Update carl.ml

commit 223e7173aef1f0fe02aabd00e9e750dde5295f3f
Author: keirl <keirl@users.noreply.github.com>
Date:   Tue May 9 22:19:05 2017 -0400

    Update carl.ml

commit d089014a5ef3a3bf95491c6f1006d2a84ae12ad3
Author: keirl <keirl@users.noreply.github.com>
Date:   Tue May 9 22:18:47 2017 -0400

    Update Makefile

commit 9d3371c624fd76a75a6af69a830aabd0b055e09c
Author: keirl <keirl@users.noreply.github.com>
Date:   Tue May 9 22:18:04 2017 -0400

    Update ast.ml

commit 76a88573d116d8f3a788bfe2ae9f2e86225200fa
Author: keirl <keirl@users.noreply.github.com>
Date:   Tue May 9 22:17:07 2017 -0400

    Update carl_engine.h

commit bfa1b2862b8a18cde3b1ebbdf75020511616506d
Author: keirl <keirl@users.noreply.github.com>
Date:   Tue May 9 22:16:21 2017 -0400

    Update carl_engine.cpp
```

```
commit c40ecec15149a64f3bf3d91a146b9614a6c52b93
Author: Darren Hakimi <dh2834@columbia.edu>
Date:   Mon May 8 18:12:13 2017 -0400

    added more regex tests and fails

commit 3841f9cad790f9d804fe365d0007c1ab2e239fd8
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Mon May 8 14:36:37 2017 -0400

    prog2 initilziation

commit b5dbd8d9ef46fb4d1c879bd2cad8a4b39b87a06b
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Mon May 8 14:31:12 2017 -0400

    prog1 intiazliation

commit e34221b7f57f80467f364bbdab0389c8383f4a21
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Mon May 8 14:28:34 2017 -0400

    prog3

commit fcc84a7078387ab89d7c45933c4c5d3e317956b7
Author: Leon Song <ls3233@columbia.edu>
Date:   Mon May 8 13:29:59 2017 -0400

    hitchhiker

commit 07ef1fab14b319bcc0e09fb2a90bf425b1d6cf96
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Mon May 8 13:13:51 2017 -0400

    all tests are passing

commit ff82bfeeda5d7eba45fd0ec8e2a6dc27acc12e74
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Mon May 8 13:06:18 2017 -0400

    update out subtraction float whatever

commit 08204b5aefed3045d25c8247f22a1d27bf269c37
Merge: 447fc82 e280095
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Mon May 8 12:49:13 2017 -0400

    Merge branch 'master' of github.com:keirl/carl

commit 447fc82007f17b944e88cbc224b2661aadc7e1a1
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Mon May 8 12:49:02 2017 -0400
```

```
    Clean up from merge.

commit e280095aecea534a9a0b8cfc0ff833587bc711a1
Merge: ab2f6bd 25f4579
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Mon May 8 12:43:34 2017 -0400

    Merge branch 'master' of https://github.com/keirl/carl

commit ab2f6bdf72de05d7b329a65e14895cdf96dfb620
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Mon May 8 12:43:14 2017 -0400

    program2

commit 25f457940d5ac5f5a7e0ea935c8ad127c96db7f6
Merge: 20b552e e32a68b
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Mon May 8 12:31:49 2017 -0400

    Merged arrays and initialization.

commit 20b552ef03fc63397f26efb395f56b4500cdd7bc
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Mon May 8 12:15:08 2017 -0400

    Init code, partial check.

commit e32a68b4f870e5560168ec65fd33510f58df629f
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Mon May 8 11:56:54 2017 -0400

    update

commit 30b2264ce3b462c2e80166af43deb38a748b67cb
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Mon May 8 11:48:23 2017 -0400

    updaefixtests

commit 981a255b486433074f29f610a5c4297dadabd9c2
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Mon May 8 11:37:09 2017 -0400

    array syntax tests update

commit a2672189e8cda32b0112f843f9af8cdcb68c4561
Merge: 295391d 0caf311
Author: Leon <ls3233@columbia.edu>
Date:   Mon May 8 11:30:53 2017 -0400

    Merge pull request #10 from keirl/syntax3

    Syntax3
```

```
commit 0caf3117f3bf34e76eeef701266b1234fb6193a0
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Mon May 8 11:29:10 2017 -0400

    latest working

commit 644d72128991d51ed9da75d3a6b79a140fbffcdb
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Mon May 8 11:16:25 2017 -0400

    latest

commit 295391dd429e4ff27b6321f87f86ca3f682838be
Author: Darren Hakimi <dh2834@columbia.edu>
Date:   Mon May 8 11:09:38 2017 -0400

    updated testall to show fail cases as PASS on proper fail

commit d7433247202fdc269a17b0a5c4e7918cc59dcfe1
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Mon May 8 10:55:14 2017 -0400

    updated

commit 33352fad200cb6f54c21f65f22ae02474be3007d
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Mon May 8 10:54:46 2017 -0400

    Put in initializing variables.

commit 3dfcb2a5dc6e0d10a9a4f8190322605b6f7ed04e
Author: Leon Song <ls3233@columbia.edu>
Date:   Mon May 8 10:43:04 2017 -0400

    parse errors pretty syntax

commit badee721006dc236654cbd0c57a4352dbe51ffa1
Merge: cff06b0 364729b
Author: Leon Song <ls3233@columbia.edu>
Date:   Mon May 8 10:10:39 2017 -0400

    merged with master, FAIL tests still failing but array tests passing

commit 364729b89e2e3472e1f6ae94a8c533eb5d661095
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Sun May 7 23:41:25 2017 -0400

    slight fix

commit f7044ffd047eeaaa490e80aa9fd8a7b00206cb33
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Sun May 7 23:37:48 2017 -0400
```

```
    slightly kewler program1

commit ec07ca4dbc8c52e8dc5c69d2de39cf8bfb92b364
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Sun May 7 23:17:20 2017 -0400

    potential program 1

commit cff06b0ee147b6a027f406601129341bc5dc5994
Author: Leon Song <ls3233@columbia.edu>
Date:   Sun May 7 19:13:00 2017 -0400

    array tests passing again after modifying check.ml

commit 321e65dc5333903cdf8189091dbdb7ee2ed76c58
Author: Darren Hakimi <dh2834@columbia.edu>
Date:   Sun May 7 18:16:43 2017 -0400

    updated testall to track fail tests

commit 8f1f0b1727da6cf008696b6dde9f1c5e730a7596
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Sun May 7 18:01:01 2017 -0400

    bit more

commit 51c3ac5b2b412d63b4bb04651740ae3a0a78bdaf
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Sun May 7 17:57:44 2017 -0400

    more stuff

commit 1422b1b1b4ea739363ddf6dbe71a44f50c887375
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Sun May 7 17:47:47 2017 -0400

    minor update

commit 89dc8876b7b84731902bda9d5f2b0dea294ca765
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Sun May 7 17:41:25 2017 -0400

    string var pred for control flow seems to work

commit e2c70fd321f1eba4408732b9f5037a5db530f740
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Sun May 7 17:22:00 2017 -0400

    slight update in strlen call no difference

commit d137529a18adc7fc129e3388dc680e8e9b43d33d
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Sun May 7 17:09:46 2017 -0400
```

```
    Added if_string_var test.

commit aad6342ad6924607946b2bcd70ff3022b1ceb42d
Author: Leon Song <ls3233@columbia.edu>
Date:   Sun May 7 16:04:27 2017 -0400

    failing match statements

commit 6193eb9ff903b0998252488ada65f7f4d9c91067
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Sun May 7 15:34:40 2017 -0400

    modify check for string var cont

commit c69e4b2e7138aa82eecfbb559110a91a2af244fb
Author: Leon Song <ls3233@columbia.edu>
Date:   Sun May 7 14:07:06 2017 -0400

    syntax test

commit 6e75751f9e4e5ff01fa21db47a07f50cbc0afa96
Author: Leon Song <ls3233@columbia.edu>
Date:   Sun May 7 14:05:38 2017 -0400

    added builtins

commit 89e902f632b0e85d1829b4d02184318f146a9b5c
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Sun May 7 13:39:04 2017 -0400

    minor check update

commit 5c0920bb8a50bdb8db69ad6dd47725116c6b4bdc
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Sun May 7 13:30:59 2017 -0400

    add function

commit ea73c12c901f738cba2b8d4eff99f4071c9091e1
Merge: 41676f7 86766f1
Author: Leon Song <ls3233@columbia.edu>
Date:   Sun May 7 13:13:58 2017 -0400

    Merge branch 'master' of https://github.com/keirl/carl into HEAD

commit 41676f7982948e29b5ca01a98eabf5250cdfa296
Author: Leon Song <ls3233@columbia.edu>
Date:   Sun May 7 13:13:35 2017 -0400

    before merging with master, float access steps pass

commit 86766f1b8c2ccbfa7f64c2d7b5e5fe0729356357
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Sun May 7 13:11:16 2017 -0400
```

```
    more string variable control stuff

commit 76cc21ec69a914a4cc084d92a57aaea423dc929c
Author: Leon Song <ls3233@columbia.edu>
Date:   Sun May 7 11:08:11 2017 -0400

    can use pretty syntax for just float arrays

commit df67c70dbbed5eda1636d28883b6cb2912a8f57e
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Sun May 7 11:05:04 2017 -0400

    latest on checker3

commit 3e989df0e053c211ce62152eb5def12ce25db751
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Sat May 6 17:33:01 2017 -0400

    float variable seems to work

commit 0fcc9f5f5344c73d8eb0d6b615bfec853eee9603
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Sat May 6 01:05:57 2017 -0400

    potential variable control flow progress

commit 5661bf90482f6dcb7e8c737b2b831cbc63349611
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Fri May 5 11:52:25 2017 -0400

    print float in check

commit 64ebdeeb2413e40f148f3434f62c9b0f25b80645
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Fri May 5 11:36:38 2017 -0400

    some prog

commit 11a0b95a1eb4a64be86d03cab645a00b0bff1d79
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Fri May 5 10:41:02 2017 -0400

    Starting checker.

commit 4ae4ad20dfaf40ba9555c0fb288188903d60ad10
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Fri May 5 09:51:33 2017 -0400

    Cleaned up git repo.

commit c92ce14d4f20209b68e8e14c40331dbe9fc9bc3e
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Thu May 4 20:45:45 2017 -0400
```

```
    Functions fixed by Keir

commit 18d073e08eba5ec5bd9ed2105d386d5337941d66
Author: Leon Song <ls3233@columbia.edu>
Date:   Mon May 1 07:31:06 2017 -0400

    fixed buildcarlp again, changed .out files for array tests

commit f4812b519e2be03e225bc6bf10df2531f4c4dd7b
Author: Darren Hakimi <dh2834@columbia.edu>
Date:   Mon May 1 01:44:06 2017 -0400

    fixed some syntax bugs in function tests

commit 1fe652c41978238291c75ccfb2845ca4bd11bcd1
Author: Darren Hakimi <dh2834@columbia.edu>
Date:   Mon May 1 01:31:53 2017 -0400

    fixed regex tests

commit 80fd5e555e870a869d79e506e51813ea270f45c9
Author: Darren Hakimi <dh2834@columbia.edu>
Date:   Mon May 1 01:17:16 2017 -0400

    more test updates

commit bbac865cab21076020256f068f49d6920ec76fac
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Sun Apr 30 23:58:09 2017 -0400

    Corrected some of the tests.

commit 6b56c9d0ed4a85173a9e62f8f6bdb603ba710787
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Sun Apr 30 23:33:05 2017 -0400

    Fixed broken tests.

commit edbc4e79d074e31a4603c7d48cf6954b11136c4b
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Sun Apr 30 23:18:31 2017 -0400

    Added -f to eliminate clean up errors.

commit b6bf99b452aa59b22266ca15ee3866df3d9acf8b
Author: Darren Hakimi <dh2834@columbia.edu>
Date:   Sat Apr 29 03:33:00 2017 -0400

    more testing, struggling with regex

commit eff373dbe31e7e942b3c099b14bb5678e6ec70e5
Author: Leon Song <ls3233@columbia.edu>
Date:   Fri Apr 28 15:18:11 2017 -0400
```

```
    hashmap_remove wrapper

commit 1036e0c3b867d8f450db4e09f6261bdefb14e056
Author: Leon Song <ls3233@columbia.edu>
Date:   Fri Apr 28 15:12:46 2017 -0400

    destroy all works with arrays! Carl engine calls this on every function. Valgrind
verified no mem leaks

commit 5af18c5b72d6f821092c598311e801939b9258de
Author: Leon Song <ls3233@columbia.edu>
Date:   Fri Apr 28 13:51:39 2017 -0400

    destroy all function works. No mem leaks

commit bbdda4ec2dd20dd76dc69611b0602e8b8bbc173c
Author: Darren Hakimi <dh2834@columbia.edu>
Date:   Fri Apr 28 01:04:55 2017 -0400

    added fail test cases and further tweaked testall and buildcarlp

commit 2c510a5e77ae36661035cd26675cc799dc6b4267
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Thu Apr 27 15:16:46 2017 -0400

    strings work without printing quotes

commit 55708588001acb964737d732de0e194244d08d73
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Thu Apr 27 14:42:55 2017 -0400

    add a few tests

commit e7e76ef5fde758c12bf2a96e87b2db1ee6393979
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Thu Apr 27 13:28:53 2017 -0400

    if semant with control flow progress

commit 3a22cbc774c96789fe63231772a81e5e6bfd7a0d
Author: Leon Song <ls3233@columbia.edu>
Date:   Wed Apr 26 19:59:58 2017 -0400

    guy is a genius

commit d3554a7adfd5c3c9f36694c25f74b750fbbe5135
Merge: f21a061 8eb3503
Author: Leon Song <ls3233@columbia.edu>
Date:   Wed Apr 26 19:42:12 2017 -0400

    Merge branch 'master' of https://github.com/keirl/carl

commit f21a061a5d92b53be6cf8b43f97352279229d3b9
```

```
Author: Leon Song <ls3233@columbia.edu>
Date:   Wed Apr 26 19:41:53 2017 -0400

    test many floats broken

commit 8eb350398fc1ea856350565e4ddd486ec7cbde44
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Wed Apr 26 14:22:24 2017 -0400

    update test loop test

commit 802d996ba12bac5e4efa750d54cb6dcec786a1e8
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Wed Apr 26 14:15:12 2017 -0400

    update loop while test

commit 1c4df268f5df445881844dd98d1cc53c1e6de0a9
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Wed Apr 26 12:58:50 2017 -0400

    removing some ints

commit 9be68b011b37b7e8b3409268b12a6db3df9ed4a3
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Wed Apr 26 12:07:53 2017 -0400

    update float subraction init test

commit ce76bb4dce9d85310d6524a60f46e411f71e8db1
Author: Leon Song <ls3233@columbia.edu>
Date:   Wed Apr 26 11:51:47 2017 -0400

    retrieve float testing on mac

commit b9d0835782bae5623fd6f1f9ffe31e0335194c98
Merge: ae41974 fd00c85
Author: Leon Song <ls3233@columbia.edu>
Date:   Wed Apr 26 11:38:05 2017 -0400

    Merge branch 'master' of https://github.com/keirl/carl

commit ae41974368eeac9a6406e691b59a639118f26df1
Author: Leon Song <ls3233@columbia.edu>
Date:   Wed Apr 26 11:37:55 2017 -0400

    float tests

commit fd00c85013f299592c26bf915ef9938909c35c3b
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Wed Apr 26 11:25:44 2017 -0400

    latest float stuff
```

```
commit 06a868564b672214af49cab56e6fa7c9920f3f25
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Tue Apr 25 12:13:27 2017 -0400

    fix float init. binop still needs to be done

commit 8de8cbe33818f08d50fb5a1a2d9ce34ed08d8a51
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Tue Apr 25 12:00:32 2017 -0400

    array string overwrite tests

commit 42499d223ef338a08601b821e041ba7ca2d8e8ae
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Mon Apr 24 14:23:44 2017 -0400

    latest with fixed tests

commit ff081ed1275c6e5e34755f752f83487ee62c94c4
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Mon Apr 24 14:04:27 2017 -0400

    clean up

commit 60c8819aa20a18417012ef47fc84eb04bc871069
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Mon Apr 24 13:47:01 2017 -0400

    add message if no file provided

commit 093cb1ecf106bae2fd92cfa0735c4a0879016f8f
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Mon Apr 24 13:24:57 2017 -0400

    Remove extra prints.

commit e625d4a1f4857dea0237d4a10ca9c52e7d1f8e11
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Mon Apr 24 13:19:36 2017 -0400

    updates

commit 29de8cabd6a2ebb72e61533dbd3b4678019e59ae
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Mon Apr 24 12:58:37 2017 -0400

    add buildcarlc script

commit 47863367ebebf3820f5d62e0900a1530d7ae7d6d
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Mon Apr 24 12:27:21 2017 -0400

    buildcarlp update
```

```
commit d7121b80bc7a7574c33b216c497ba05e0c5a57ac
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Mon Apr 24 11:59:51 2017 -0400

    latest array

commit 186b5ecb40ec03f335dec3830fc91558e4f413a5
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Mon Apr 24 10:45:12 2017 -0400

    ltest

commit a4f0ff5c9466b38d95e0f7f30004c38067bd0105
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Sun Apr 23 14:06:29 2017 -0400

    update test file

commit 6ebfa4ad5bfe3a74e2a5fef9ec30cfe90d36f651
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Sun Apr 23 13:37:45 2017 -0400

    more update

commit fe62f45afc2a4628292197cc78a3cb2cb154a3d1
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Sun Apr 23 11:55:21 2017 -0400

    updatesss

commit 7460b956dc1f0422223b3bcd3c761ec31a9a5b56
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Sun Apr 23 11:34:15 2017 -0400

    updates

commit 7b44b0fe5ae0503a8b9c575e5fcadf8a60a2c745
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Sun Apr 23 11:05:44 2017 -0400

    update

commit 41d5b0fe3f3771312e699e4e4e16a5678ffca0ff
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Sun Apr 23 10:42:12 2017 -0400

    remove reduce reduce errors

commit d1142fd7f05b09c3c839a464998a0cc3da571db2
Author: Darren Hakimi <dh2834@columbia.edu>
Date:   Sun Apr 23 01:51:51 2017 -0400

    fixed testall script, seems fully functional
```

```
commit c6e53eb97621b4fc9f721688f41dec45e2153718
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Fri Apr 21 23:51:21 2017 -0400

    Slightly modified build scripts.

commit db610dc31868f98994e3436a657d6900937f00ac
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Fri Apr 21 21:03:30 2017 -0400

    move some code from arrayb to master

commit d66fb1281c52368b7e62e3c5466a0941d5692595
Merge: 2fe254c 153fec8
Author: Leon <ls3233@columbia.edu>
Date:   Fri Apr 21 20:18:51 2017 -0400

    Merge pull request #9 from keirl/array_lib

    Array lib

commit 153fec85a9563a8c21573a0ce8a3f02ca99d797c
Author: Leon Song <ls3233@columbia.edu>
Date:   Fri Apr 21 20:18:14 2017 -0400

    removed debugging statements

commit a39e76dee770230114ad095f4345ec50b067d158
Author: Leon Song <ls3233@columbia.edu>
Date:   Fri Apr 21 20:15:08 2017 -0400

    thank god, finally no mem leaks

commit 2fe254c7c20ff190d766ee670780180836a252af
Author: Darren Hakimi <dh2834@columbia.edu>
Date:   Fri Apr 21 18:19:48 2017 -0400

    fixed buildcarlp and testall.sh

commit 66ac00e8f971232543e052727a4162094fcc1897
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Thu Apr 20 17:24:41 2017 -0400

    make tmp dir if not there

commit 9afaecdc3124b98c2c4204222046fa8205df2d6d
Merge: 90b04d0 a419c6b
Author: keirl <keirl@users.noreply.github.com>
Date:   Thu Apr 20 15:55:54 2017 -0400

    Merge pull request #8 from keirl/build_script

    Build script
```

```
commit a419c6bd9734d36122cc6f71f25034537986921c
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Thu Apr 20 15:41:28 2017 -0400

    Ready for Darren

commit 656fe5b02d218933561ef67884143d07875273a2
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Thu Apr 20 15:22:48 2017 -0400

    Build code.  Needs to fix testall.sh

commit 213dd32070678902df27d5d452eacf2545163b93
Author: Leon Song <ls3233@columbia.edu>
Date:   Thu Apr 20 15:20:32 2017 -0400

    no mem leaks

commit c605c001c294bb81fd881ea31ec1bd7d7e80fa3d
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Thu Apr 20 14:48:07 2017 -0400

    Reworked file structure

commit c779b244aa0d1624957fe966b45746b7585cb950
Author: Leon Song <ls3233@columbia.edu>
Date:   Thu Apr 20 13:54:58 2017 -0400

    memory leaks

commit 90b04d087f0f8f7e1e34c1d861e900385b78ed05
Author: Darren Hakimi <dh2834@columbia.edu>
Date:   Thu Apr 20 00:25:03 2017 -0400

    moved tests directory and created testall script

commit 26a23dc4b59077fd227187358be8c661eea5b8d7
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Mon Apr 17 22:04:46 2017 -0400

    Minor fix to build

commit 8387c11fe456cdb824c1971491630e6a92b7be9b
Merge: fd083f3 2fe741c
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Mon Apr 17 22:00:03 2017 -0400

    Minor commit

commit fd083f32cda3e6157fe79bd7ea584aa36b807cda
Merge: ef53723 db61570
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Mon Apr 17 20:51:02 2017 -0400
```

```
    Working, integrated carl_engine and carl compiler.

commit db61570016b2c10a2681b7fa820acc485f9296d1
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Mon Apr 17 20:36:45 2017 -0400

    Working integration with carl_engine.

commit 2fe741ca95238894be9043c37c55079cc532a3cb
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Mon Apr 17 14:16:59 2017 -0400

    update codegen Sast var

commit bda831252ace8d0a82f2bf430b9259ef337ade01
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Mon Apr 17 14:02:21 2017 -0400

    Working interface builder.

commit 08de6ec041ef187513934516f056af2e82ca3507
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Mon Apr 17 13:32:56 2017 -0400

    codegen update

commit 3acbf722e32816890ea9cbc5aeb33f70bd2bf2da
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Mon Apr 17 13:01:31 2017 -0400

    Working char* array.

commit 1b82440f0a78471103041bdb2a7cdd891b3e7a77
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Mon Apr 17 12:41:14 2017 -0400

    Partial string arrays.

commit 4eafde675b3189a3198cfc6994b6d2667f4023c9
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Mon Apr 17 11:27:39 2017 -0400

    Working function pointer arrays.

commit 5875ce6570e20e45cf34fd50214b72a1ecd252e7
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Mon Apr 17 10:58:08 2017 -0400

    slight update to scanner float

commit 6ae5dfae03f51ff3b6c842110055943062af72e9
Merge: 8ac1e99 ef53723
Author: Darren Hakimi <dh2834@columbia.edu>
Date:   Mon Apr 17 02:40:42 2017 -0400
```

```
    Merge branch 'master' of https://github.com/keirl/carl

commit 8ac1e9910971a14639c64cd2d619a03ce5497d71
Author: Darren Hakimi <dh2834@columbia.edu>
Date:   Mon Apr 17 02:22:48 2017 -0400

    added more to regex test and made some test cases in .carl files

commit a9674be4d3c7cf771f497a3e5c40993f6710e06f
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Sat Apr 15 20:54:41 2017 -0400

    Working function pointer.

commit 468b3a89616f8c3dc740d5fc32730f1e6e114a55
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Sat Apr 15 12:56:50 2017 -0400

    Work in progress

commit 8032f73fa92cd3c86e3fe03e595fb8a833861d85
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Sat Apr 15 10:01:29 2017 -0400

    Commented out function init.

commit 1a65479d4fdb3bee76e8f1fb176271cd9fdf5fb2
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Fri Apr 14 14:14:48 2017 -0400

    Can make 1 pattern.

commit 2d228cb1746848b03f8afcb05828c6c087c86d82
Author: Darren Hakimi <dh2834@columbia.edu>
Date:   Fri Apr 14 13:30:25 2017 -0400

    built initial regex_test file

commit 1c6bbf9ec430ad5d11252b2bc610d3d5e4397ab4
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Fri Apr 14 13:29:57 2017 -0400

    Code to build init function.

commit ef53723eaa9aa0c043864a6a70f9f1fe3661f31d
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Tue Apr 11 20:52:26 2017 -0400

    Fixed build issue.  Returns strings in Regex.

commit 1b0372fff2cfd5e1b51316da0347b5d6e494ea6e
Merge: cb48557 4a739fd
Author: Keir Lauritzen <kcl2143@columbia.edu>
```

```
Date:   Tue Apr 11 20:46:35 2017 -0400

    Merge branch 'master' of github.com:keirl/carl

commit cb48557973851724040fea5a39e16427dedf74c0
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Tue Apr 11 20:46:32 2017 -0400

    Final codegen change

commit 4a739fdfcad3eb0b63218e48bac282b6cd4124ce
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Tue Apr 11 17:04:30 2017 -0400

    codgen float

commit 41d814b66d19177f6425cab342f43795cddaf445
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Tue Apr 11 16:59:31 2017 -0400

    Patterns now accept initializers.

commit 6f1191710f3d605a6a09d048b24aa2046ffa2192
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Tue Apr 11 16:44:09 2017 -0400

    Patterns are now strings instead of patterns.

commit 103fc6287fc45f3e68a384a8fe836b15162b9bba
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Tue Apr 11 16:24:27 2017 -0400

    some float stuff

commit 45769d2f376d76e9e8f20ffa88cb5ac0499e1f39
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Tue Apr 11 16:03:47 2017 -0400

    Eliminated shift/reduce conflict

commit 554f004bbd8188b6e2d3143d09cf8a3909c6de91
Merge: 89b18bc 6d30cc8
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Tue Apr 11 15:50:22 2017 -0400

    Correcting merge conflicts.

commit 89b18bce829ccf00fc21eb2337ff1ecaf63606d7
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Tue Apr 11 15:48:30 2017 -0400

    Fixing compilation errors from last night pull.

commit 6d30cc8bdda6c629e54a5d2804c0e533a15cb0a9
```

```
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Tue Apr 11 15:46:33 2017 -0400

    minor

commit 49d906a21e3da6e42759063dbb1e6609724786ae
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Tue Apr 11 14:54:54 2017 -0400

    some updates

commit 877338d9e33cd2ac073f93fc175a82c31fc2f826
Merge: 79d7be8 57d6e16
Author: keirl <keirl@users.noreply.github.com>
Date:   Mon Apr 10 18:16:00 2017 -0400

    Merge pull request #7 from keirl/build_rules

    Build rules

commit 57d6e167ce8c3722a0a2295b7f2a57d6af2a2c08
Merge: 504b404 79d7be8
Author: keirl <keirl@users.noreply.github.com>
Date:   Mon Apr 10 18:14:01 2017 -0400

    Merge branch 'master' into build_rules

commit 504b404a2ac837ddef27813f5b42699daab647d5
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Mon Apr 10 17:37:07 2017 -0400

    Compiling SAST with Codegen

commit 6c176d7ba7f3c54f3023748afb3d8e587948c972
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Mon Apr 10 17:21:05 2017 -0400

    Should have been on the previous merge.

commit ee0a27a63a8a20c86487acc7be662cb648a2faac
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Mon Apr 10 17:20:20 2017 -0400

    SAST builder that turns actions into functions and declares patterns.

commit dd0f019f3c75c77339125ed8d432a442c4cd635e
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Mon Apr 10 15:37:09 2017 -0400

    Initial AST to SAST conversion.

commit 217cb5bfbb21d81263d9d7f5f5f5b39af86ca467
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Mon Apr 10 13:49:11 2017 -0400
```

```
    Moved to the old AST.

commit 79d7be8b97d8cf9fb7c0976c077ef794742bc49a
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Mon Apr 10 11:38:05 2017 -0400

    adding tokens for actions and patterns

commit 3f33379733b76bf6e6564e09f073b7c07f4a8bca
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Mon Apr 10 11:06:31 2017 -0400

    some func tests

commit d4f988dd47354e1db1cb21d52fbaa627fe3ce68d
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Mon Apr 10 10:51:26 2017 -0400

    Parser is not working with array, this adds a print statement that proves it.

commit 9a7038fe64864dab6368c555908ad56068740b11
Author: Darren Hakimi <dh2834@columbia.edu>
Date:   Mon Apr 10 01:56:31 2017 -0400

    added test cases for strin/array, and binary op test cases for int/float

commit 6bbb75bb33e3e4350c78dbce19bcdf3a4b222f41
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Sun Apr 9 22:55:42 2017 -0400

    Continuing build rules.

commit 9579b9ed2254b2a160de79642bb7c2ac58c567c6
Author: Darren Hakimi <dh2834@columbia.edu>
Date:   Sun Apr 9 19:57:15 2017 -0400

    test cases for int/float assignment/declaration, ifelse statements, for loops, and
while loops

commit 805690ce01940da36c1998541ed8bde399441086
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Sun Apr 9 15:58:30 2017 -0400

    couple simple tests

commit 8224f3bf1798a0ff329595dbe583a6f457e8d8e1
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Sun Apr 9 14:08:21 2017 -0400

    more updates

commit bdf3452a92724854fcba1bace463dfd6ae0560c7
Author: Guy Yardeni <gy2241@columbia.edu>
```

```
Date:   Sat Apr 8 17:23:51 2017 -0400

    more

commit f9a33d976a3efab8b87c8d6371200c4fc0959b0d
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Sat Apr 8 13:17:06 2017 -0400

    more updates to scanner parser

commit 2d9709de95d9353c92a6ff3675bb1a1c46401ea2
Merge: cf4df0f 3885bfe
Author: gyardeniC <gy2241@columbia.edu>
Date:   Sat Apr 8 12:10:26 2017 -0400

    Merge pull request #6 from keirl/darren-regex

    Darren regex

commit 3885bfeed0a1cbddb0dd51dc079eafccd12730f7
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Sat Apr 8 11:02:58 2017 -0400

    some additions to scanner and parser based off lrm

commit 048354438b5161a8e4cd7a28695f28aa0c1c4308
Author: Leon Song <ls3233@columbia.edu>
Date:   Fri Apr 7 12:10:50 2017 -0400

    great, works wit basic regex, now for more extensive testing

commit b922650b4dc5d351411551dc221ccadb08852543
Author: Leon Song <ls3233@columbia.edu>
Date:   Thu Apr 6 19:48:31 2017 -0400

    merged again from master, and re-added changes to print ast

commit a5752ccbc3516aed7338a3c56ad8654069258a02
Merge: d1b5321 cf4df0f
Author: Leon Song <ls3233@columbia.edu>
Date:   Thu Apr 6 19:27:23 2017 -0400

    fix conflicts from merge with master

commit d1b532146c914bd603bf0b5e8739a49144e6a102
Author: Leon Song <ls3233@columbia.edu>
Date:   Thu Apr 6 18:46:16 2017 -0400

    made entry point into print ast.string_of_program

commit 33059c34ca746f1ad0139c240808446c291fdb2e
Author: Leon Song <ls3233@columbia.edu>
Date:   Thu Apr 6 18:22:19 2017 -0400
```

```
    changed scanner parser to accept regex, still untested. Carl.ml calls
scanner.next_token instead of scanner.token

commit cf4df0f9eb7f1b9fa2c1412d3bf6034681893d8d
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Thu Apr 6 09:10:46 2017 -0400

    Incorporated changes to support strings (not chars).

commit 60194c780c000aa783964ec91783494268dcfe36
Author: Darren Hakimi <dh2834@columbia.edu>
Date:   Thu Apr 6 01:28:20 2017 -0400

    added cpp regex test file

commit 267dd6f83e6b7aee380b8a5c651a3e796ca4e2cb
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Thu Apr 6 00:56:47 2017 -0400

    Modified build to include carl_engine

commit 791bca7c2161135fe3de2cb12815c5b9a64f4cd2
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Wed Apr 5 23:12:29 2017 -0400

    Added codegen code to carl.ml

commit ad3dabb0ae116a8f20a4b0f399d6be834fc72a65
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Wed Apr 5 23:05:12 2017 -0400

    Fixed indentation on codegen.ml.  Might not be perfect, but it is better.

commit 9e70b6173ce2bc301426145d512b72e9973e79b9
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Wed Apr 5 22:50:57 2017 -0400

    MicroC codegen ported to Carl Ast.  Ready to add strings.

commit 4963166c24ef4c5b926e203739916e1252d501a3
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Wed Apr 5 22:35:08 2017 -0400

    Actually working parser and scanner with explicit types.  No floats or arrays.

commit 505cec81802d2a6df7e99f31a0ad29a1bb97f08c
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Wed Apr 5 22:10:05 2017 -0400

    Including codegen diff

commit 4baf5410822fad9651eff081cb31adfc30494353
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Wed Apr 5 21:54:03 2017 -0400
```

```
    Working parser with explicit types and declarations.

commit 1f9340ea96725e1bad469511e90c5edc8d2a352d
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Wed Apr 5 17:02:19 2017 -0400

    Scratch code on LLVM bindings.

commit e2273e34d9baece839fdc865bf5f394dc7b7e8ee
Author: Darren Hakimi <dh2834@columbia.edu>
Date:   Wed Apr 5 02:23:58 2017 -0400

    updated scanner, parser, and ast with initial regex stuff

commit 793255caba9fcce72e61e3658a55258c8b104120
Author: Leon Song <ls3233@columbia.edu>
Date:   Tue Apr 4 16:57:24 2017 -0400

    Macroc could not compile. Restarted AGAIN with microc-llvm. Now confirm can work
with strings, chars, and prints fine

commit 9204ae11210d8015d073bb4eddf7d4ce2016a1be
Merge: 3855471 19b4c41
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Mon Apr 3 22:35:05 2017 -0400

    Merge branch 'master' of github.com:keirl/carl

commit 385547108b0a96379aeb966a2c5b740250e027c0
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Mon Apr 3 22:34:53 2017 -0400

    Updated Makefile to clean more files.

commit 19b4c41f4367a936f7c53b44d982142840fe9e43
Merge: bfe0067 048658f
Author: keirl <keirl@users.noreply.github.com>
Date:   Mon Apr 3 22:08:06 2017 -0400

    Merge pull request #5 from keirl/codegen_string_support

    works with chars and strings

commit bfe0067fb0243f00ffa4a7a243d57d8d07be0a1f
Merge: 14f3c36 fa7d473
Author: keirl <keirl@users.noreply.github.com>
Date:   Mon Apr 3 22:05:18 2017 -0400

    Merge pull request #4 from keirl/sast-research

    Sast research

commit fa7d4738c32090fb903b6fa3d2eb697ba47fd77b
```

```
Merge: 6fd573a 14f3c36
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Mon Apr 3 22:04:48 2017 -0400

    Merged sast-research to master.

commit 14f3c367ab2aa6d5fa83a73c345123e31ede12ae
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Mon Apr 3 21:52:31 2017 -0400

    Undo char and string support

commit 54688ee4719c351cf5d132397bef8e053be0f872
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Mon Apr 3 21:15:25 2017 -0400

    add recent makefile

commit b53dc5183a79311cf6eee9603d1051cb11c8d0b1
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Mon Apr 3 21:07:46 2017 -0400

    Stopped tracking object files.

commit 7d2271b3246c3d42485bc5f8f37b5b12559ef783
Merge: 58c91a9 4b19afd
Author: keirl <keirl@users.noreply.github.com>
Date:   Mon Apr 3 21:04:53 2017 -0400

    Merge pull request #2 from keirl/carl_engine

    Carl engine remerge

commit 4b19afdc091fb79f08ba5efd0cdbb4990fca257b
Merge: cade40f 58c91a9
Author: keirl <keirl@users.noreply.github.com>
Date:   Mon Apr 3 21:04:32 2017 -0400

    Merge branch 'master' into carl_engine

commit 58c91a9322107514e5a2ac6b451a0e0e53038562
Author: Leon Song <ls3233@columbia.edu>
Date:   Mon Apr 3 20:52:36 2017 -0400

    support for chars and strings

commit 048658fdd4a903e4c905770496069488bcbcd0f2
Author: Leon Song <ls3233@columbia.edu>
```

```
Date:   Mon Apr 3 17:22:04 2017 -0400

    works with chars and strings

commit 6fd573ac4c1af9a11b2a8cf2bcafb055b9693be7
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Fri Mar 31 22:58:15 2017 -0400

    Debug gather_variables

commit 78a6ff5dbda16801d37978b249ea29ca6d9f61b3
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Fri Mar 31 14:02:10 2017 -0400

    Enumerated patterns.

commit b7e0dd0e700ac90133778c02f2eb35cad270ab4d
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Thu Mar 30 22:40:06 2017 -0400

    Fixed Makefile

commit 4980f12345c89d9a34df9a8b02703ff224fae5d1
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Thu Mar 30 22:38:01 2017 -0400

    Initial conversion to cast.

commit 76413a1a5943e2d8e45fb10e7621f42984d37c7a
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Thu Mar 30 17:18:20 2017 -0400

    update Makefile order

commit 8fc67b28ca24055b2e24232c62bc99657005b329
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Thu Mar 30 17:02:39 2017 -0400

    Added Ast.

commit 16ab9cc097a98d3bc9607fdcace4690fcbded379
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Thu Mar 30 16:58:33 2017 -0400

    Cleaned parser to get convert_pattern to work.

commit 6092f58ba6497ef7846c6616ddf6bc5c5d6728c2
Author: Leon Song <ls3233@columbia.edu>
Date:   Thu Mar 30 15:49:52 2017 -0400

    mar 30

commit 2e7385e692ed1b289aaa837517e736569db610bb
Author: Leon Song <ls3233@columbia.edu>
```

```
Date:   Thu Mar 30 07:51:53 2017 -0400

    semantics.ml

commit cade40fdc6249d50cc8d7d3ebdaf7b3805f763e2
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Wed Mar 29 15:12:31 2017 -0400

    carl_engine complete.

commit 1dbc368b4f06101ffeef3c5127547c0b62a740a5
Author: Leon Song <ls3233@columbia.edu>
Date:   Wed Mar 29 12:11:45 2017 -0400

    small changes to semantics after meeting graham

commit a0095f3cbfd0d74b9a49904fe1ff70420a525bca
Author: keirl <keirl@users.noreply.github.com>
Date:   Wed Mar 29 12:10:20 2017 -0400

    Fixed minor mistakes in the compilation diagram.

commit bebd74f22735fa33b293320507ea42524b30aa81
Author: keirl <keirl@users.noreply.github.com>
Date:   Wed Mar 29 11:18:18 2017 -0400

    Compilation process

commit e6a1ed650355e4be5533c4f99658ab386af1ec52
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Wed Mar 29 11:15:49 2017 -0400

    Generates LLVM.

commit 848aa43848bbb4dc8b712f377b79278cab728976
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Wed Mar 29 00:17:34 2017 -0400

    Commented out extra functions.

commit c6393fa162e1db7f09f2ad758d0786ddb73d3e71
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Wed Mar 29 00:15:49 2017 -0400

    Working pattern action pointers.

commit 8bb17947456bb8fd3ffb0fba577c84b0b7f89c31
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Tue Mar 28 23:34:26 2017 -0400

    Implemented regex

commit 3e1e38ca5768e6343335c13eaf62b089606a7a6b
Author: Keir Lauritzen <kcl2143@columbia.edu>
```

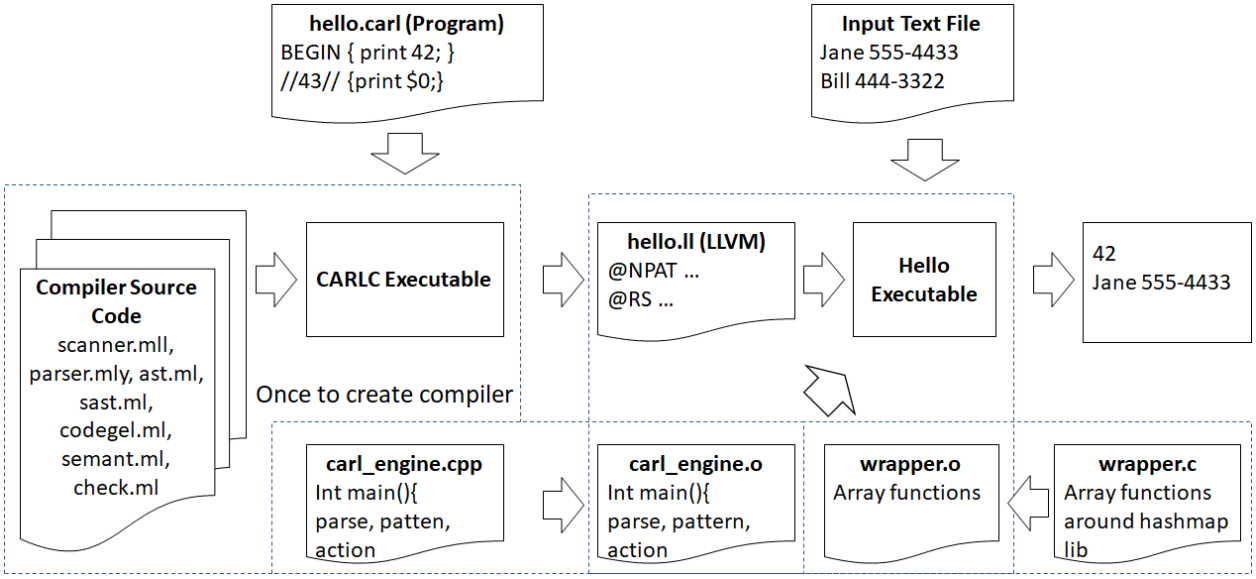
```
Date:   Tue Mar 28 22:48:57 2017 -0400

    Initial CARL engine development

commit dd9cbe491587580777a9fca76c61449080a9770e
Author: Leon Song <ls3233@columbia.edu>
Date:   Tue Mar 28 17:26:42 2017 -0400

    modified scanner, parser, ast, semant.ml to try and support type inference.

commit 188215aa524aacb0c2857840b9a6ed508125d938
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Tue Mar 28 16:37:28 2017 -0400

    latest

commit 160359ef0c23b035289800985dea0374b5d363af
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Tue Mar 28 11:32:29 2017 -0400

    Initial two-pass compiler.

commit 23b1b07d66e16c460957df3f29623b2d10745f86
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Mon Mar 27 14:34:13 2017 -0400

    Cleaned up indentation.

commit b18df80faa7c345206f0fea70becc280612398ce
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Mon Mar 27 13:06:24 2017 -0400

    98% of MicroC

commit f57fc5a6b54f30935c032dbb11eeb4ab96ee5815
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Mon Mar 27 11:14:43 2017 -0400

    Initial branch

commit 55efba0ab0791c35e652b501351fda5e32c5dd09
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Wed Mar 22 13:09:09 2017 -0400

    add a make clean for now

commit af439d2ee8b17e8ec4ae973a75cf38ca0cd744c3
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Tue Mar 21 17:36:33 2017 -0400

    modify scanner parser ast makefile and microc. need to get codegen to work
witheverything

commit ab275835261d1556f83ca6a24298c1a08e2d73c5
```

```
Author: Leon Song <ls3233@columbia.edu>
Date:   Mon Mar 20 16:45:33 2017 -0400

    second commit

commit a03e1a3be1a603bc62d5b4964b8b78cc88947bee
Author: Leon Song <ls3233@columbia.edu>
Date:   Mon Mar 20 14:33:05 2017 -0400

    initial skeleton of parser, very unfinished

commit 114b250fc343f0e2fd4fea0a15abcd576ce9f9b9
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Mon Mar 20 10:36:34 2017 -0400

    Change base directory.

commit 8310c66444962557aac3773c584fb9c96bbdf16d
Author: Keir Lauritzen <kcl2143@columbia.edu>
Date:   Mon Mar 20 10:26:57 2017 -0400

    Add example code to separate drectory for future reference.

commit 844f60bb87a801e39335142f5b5990e73193f558
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Wed Mar 15 22:10:16 2017 -0400

    committing without the make-generated files

commit b869389210dba87268a312b62ab41155cbbb5ed8
Author: Guy Yardeni <gy2241@columbia.edu>
Date:   Wed Mar 15 19:11:19 2017 -0400

    default microc

commit 6f7e8a246dac939b3bc6529b79d097aaa1f4c2c0
Author: keirl <keirl@users.noreply.github.com>
Date:   Wed Mar 1 15:51:58 2017 -0500

    Initial commit
```

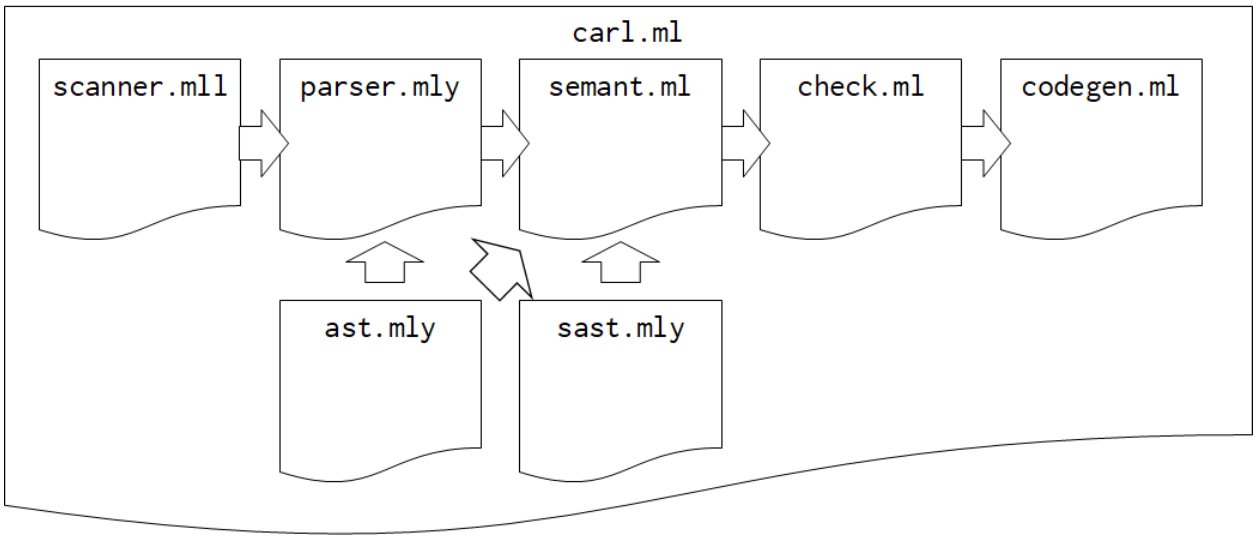
5. Architectural Design

CARL has three major components: carlc, carl_engine, and wrapper. carlc is the CARL compiler which converts .carl programs to LLVM. carl_engine is a C++ library that call functions and variables defined in the LLVM and executes them. wrapper.c is a library to provide array functionality. The diagram belows shows the compilation flow and architecture of the CARL compiler and supporting libraries, carl_engine and wrapper. The CARL compiler is built with a

similar structure to MicroC. Each component, the LLVM carl code, carl_engine, and wrapper are compiled to object code and then linked.



The carl compiler converts .carl files to LLVM for further compilation to as part of a final executable. The diagram below shows the program flow defined in carl.ml. The carl compiler was built using MicroC as its basis, so many of the files are similar.



scanner.mll and parser.mly

The scanner implements a standard OCaml scanner creating token for parsing. The parser converts the tokens into CARL's abstract syntax tree, with types defined in ast.ml. The scanner switches to a REGEX set of tokens for REGEX expressions to help the parser distinguish between REGEX and CARL sysntax. The parser parses both the regular CARL grammar as well the REGEX to verify that it is syntactically correct. The parsed REGEX is provided to

semant as a regular string. The REGEX parser and scanner was done by Leon Song and Darren Hakimi and they reviewed a prior 4115 project HAWK in implementation (http://www.cs.columbia.edu/~%20sedwards/classes/2015/4115-fall/reports/HAWK.pdf). Everyone made modification to both modules to implement other features in the compiler.

semant.ml

Semant converts the CARL AST to a more C-like AST that is more compatible with LLVM generation. The goal was to keep codegen as simple and C-like as possible and put everything complicated in semant. Semant would probably be better named translate.ml, since it only does a little bit of checking but a lot of translation. All AST types are converted to SAST at this stage. Boolean conversion occurs in semant, as well as variable initialization.

The major features of CARL are implemented in semant. CARL arrays created and manipulated using a C library wrapper.c, which is a wrapper around another hashmap library. Items are added, retrieved, created, and destroyed using function calls. In semant.ml, the built-in CARL arrays syntax is converted to function calls. Guy Yardeni and Leon Song implemented arrays.

CARL rules are converted to the carl_engine interface as well in semant. All actions are converted into functions with the protoype: `void __action0(int, char**, int*)` (0, 1, 2…) These functions are then placed in an array of function pointers. All of the patterns become string literals and they are put into an array of C-style char*'s `__patterns`. Additional fields are passed over as well, including the number of patterns (NPAT). The figure below shows the interface. While CARL only supports three types, semant.ml, sast.ml, and codegen.ml all needed to be modified to include many other LLVM/C types up to and including arrays of function pointers. An initialization function `__init()` was also created to handle assignment of function pointers and strings to the arrays. Keir Lauritzen implemented the carl_engine and the initial semant to integrate it.

carl_engine.cpp

```
int main(){
  opens the file;
  parse using RS, FS
  make_c_arrays
  for (int i; i < NPAT, i++){
    regex(pattern[i])
    if regex
      (*action[i]) (int len, char**
cfields, char* cfield_len)
  }
  return 0;
}
```

carl_source.llvm

```
int NPAT = 2;
char RS = '\n';
char FS = ' ';
char* __patterns[NPAT];
void (*__actions[NPAT])(const int,
char**, int*);
__init()
```

check.ml

Check performs additional semantic checking on the C-like semantically-checked abstract syntax tree. The includes checking for the existence of functions and variables. This was largely built using MicroC's semant.ml as the starting point.

codegen.ml

CARL moved from an AWK-like syntax to a C-like syntax in semant.ml. This allowed us to leverage much of MicroC's codegen; in effect all of the hard work was done in semant. The most of the changes were in two arrays: adding built-in functions for CARL arrays and support C-style arrays and function pointers to create the interface with carl_engine. The C-style arrays and function pointers required additional expressions in the SAST to be created. These then had unique LLVM calls that were part of codegen.

carl.ml

Carl.ml is simply the driver to connect all of the other components together to produce LLVM.

carl_engine.cpp

Carl_engine is a C++ file that is a carl executable's main() function. It open up the input file provided on the command line. It then goes through each rule, loading the pattern from the .carl (that has been compiled to LLVM and then .o). The pattern is converted to a C++ string and then into a regex expression (BEGIN and END are handled differently.) The file is then parsed into separate records using the RS variable and then distinct fields by the FS variable. This the standard behavior of AWK. We used the C++ standard library for the REGEX library. Each record is then matched against the standard library and if it matches, then the that action is

called. Once all record have been processed then it goes on the next rule. It loops through all patterns and actions in their arrays until NPAT is reached. Everything was implemented as arrays of pointers in order to create a simple interface that can take an arbitrary number of rules.

carl_engine parse the input file into distinct fields to support AWK's `$0, $1, $2` references; however, we were not able to implement them. Much of the code in carl_engine is to handle the conversion from C++ to C-style code. The interface that carl_engine, codegen, and semant support passes three arguments to action functions. The first is an `int` that defines the number of fields being passed over. The second is a char** that points to an array of C-style strings (char*). These contain the actual values of fields. The third argument is an int* that points to an array of ints that contain the length of each string. The strings are null terminated, but this is still often useful with C string operations. Keir Lauritzen implemented carl_engine.cpp.

wrapper.c

To implement arrays, we built off an existing C library (https://github.com/petewarden/c_hashmap) for hashmaps and compiled that code down an object file before linking it with the engine and compiler.

Because the C library didn't exactly meet our requirements, we have to write wrapper functions to do some extra work that CARL needed. For example, all array data needs to be malloc'd and freed. The C library also tries to work with pointers and void types, both are things which CARL does not support. So we had to write wrapper.c to provide an interface between CARL and this library.

To give you an idea of what these wrapper functions are like, consider our wrapper.h file:

```
#ifndef __WRAPPER_H__

#define __WRAPPER_H__



extern void* create();

char *malloc_string(char *string);

extern void destroy(void* map);

extern int array_add_string(void *map, char *key, char* value);

extern int array_add_float(void* map, char* key, double value);

extern int lookup(void* map, char* key);

extern int array_remove(void* map, char* key);

extern char* array_retrieve_string(void* map, char* key);

extern double array_retrieve_float(void *map, char* key);
```

```
extern void destroy_all();

#endif
```

CARL users would call array code by doing myArray[“string_literal”] = 42.31, while in the backend this wrapper.c code will take these values, malloc them and pass the pointers to the library.

The wrapper code also has to take care of cleanup using destroy and destory_all, which book keeps all the data that a CARL user mallocs, and calls free automatically at the end of program execution.

6. Test Plan

test-program1.carl

```
/* This code was written by Guy Yardeni and Leon Song */

function float slight_increase(float val) {
	return val * 1.1;
}

BEGIN {
	float float_val1 = 4.321;
	string string_val1 = "I'm just a string.";
}

//float//  {
	while (float_val1) {
		print_float(float_val1);
		float_val1 = float_val1 - 1;
		float_val1 = slight_increase(float_val1);
	}
}

//string// {
	if (string_val1) {
		print_string(string_val1);
		while (string_val1) {
			print_string(string_val1);
			string_val1 = "";
		}
	}
}

END {

	if (float_val1 > 4.0) {
		print_string("strings are the best");
	} else {
		print_string("floats are the best");
	}
}
```

test-program1.ll

```
; ModuleID = 'Carl'

@NPAT = global i32 0
@FS = global i8 0
@RS = global i8 0
@__actions = global [4 x void (i32, i8**, i32*)*] zeroinitializer
@__patterns = global [4 x i8*] zeroinitializer
@float_val1 = global double 0.000000e+00
@string_val1 = global i8* null
@fmt = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt2 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@fmt3 = private unnamed_addr constant [4 x i8] c"%f\0A\00"
@fmt1 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt22 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@fmt33 = private unnamed_addr constant [4 x i8] c"%f\0A\00"
@str = private unnamed_addr constant [21 x i8] c"strings are the best\00"
@str4 = private unnamed_addr constant [20 x i8] c"floats are the best\00"
@fmt5 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt26 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@fmt37 = private unnamed_addr constant [4 x i8] c"%f\0A\00"
@str8 = private unnamed_addr constant [1 x i8] zeroinitializer
@fmt9 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt210 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@fmt311 = private unnamed_addr constant [4 x i8] c"%f\0A\00"
@fmt12 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt213 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@fmt314 = private unnamed_addr constant [4 x i8] c"%f\0A\00"
@fmt15 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt216 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@fmt317 = private unnamed_addr constant [4 x i8] c"%f\0A\00"
@str18 = private unnamed_addr constant [19 x i8] c"I'm just a string.\00"
@str19 = private unnamed_addr constant [4 x i8] c"END\00"
@str20 = private unnamed_addr constant [7 x i8] c"string\00"
@str21 = private unnamed_addr constant [6 x i8] c"float\00"
@str22 = private unnamed_addr constant [6 x i8] c"BEGIN\00"

declare i32 @printf(i8*, ...)

declare double @strlen2(i8*)

declare i8* @create()

declare i32 @array_add_string(i8*, i8*, i8*)

declare i8* @array_retrieve_string(i8*, i8*)

declare i32 @array_add_float(i8*, i8*, double)

declare double @array_retrieve_float(i8*, i8*)

define double @slight_increase(double %val) {
entry:
  %val1 = alloca double
  store double %val, double* %val1
```

```
  %val2 = load double* %val1
  %tmp = fmul double %val2, 1.100000e+00
  ret double %tmp
}

define void @__action3(i32 %len, i8** %cfields, i32* %field_len) {
entry:
  %len1 = alloca i32
  store i32 %len, i32* %len1
  %cfields2 = alloca i8**
  store i8** %cfields, i8*** %cfields2
  %field_len3 = alloca i32*
  store i32* %field_len, i32** %field_len3
  %float_val1 = load double* @float_val1
  %tmp = fcmp ogt double %float_val1, 4.000000e+00
  br i1 %tmp, label %then, label %else

merge:                                            ; preds = %else, %then
  ret void

then:                                             ; preds = %entry
  %printf = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([4 x i8]* @fmt22,
i32 0, i32 0), i8* getelementptr inbounds ([21 x i8]* @str, i32 0, i32 0))
  br label %merge

else:                                             ; preds = %entry
  %printf4 = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([4 x i8]*
@fmt22, i32 0, i32 0), i8* getelementptr inbounds ([20 x i8]* @str4, i32 0, i32 0))
  br label %merge
}

define void @__action2(i32 %len, i8** %cfields, i32* %field_len) {
entry:
  %len1 = alloca i32
  store i32 %len, i32* %len1
  %cfields2 = alloca i8**
  store i8** %cfields, i8*** %cfields2
  %field_len3 = alloca i32*
  store i32* %field_len, i32** %field_len3
  %string_val1 = load i8** @string_val1
  %strlen = call double @strlen2(i8* %string_val1)
  %tmp = fcmp ogt double %strlen, 0.000000e+00
  br i1 %tmp, label %then, label %else

merge:                                            ; preds = %else, %merge10
  ret void

then:                                             ; preds = %entry
  %string_val14 = load i8** @string_val1
  %printf = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([4 x i8]* @fmt26,
i32 0, i32 0), i8* %string_val14)
  br label %while

while:                                            ; preds = %while_body, %then
```

```
  %string_val17 = load i8** @string_val1
  %strlen8 = call double @strlen2(i8* %string_val17)
  %tmp9 = fcmp ogt double %strlen8, 0.000000e+00
  br i1 %tmp9, label %while_body, label %merge10

while_body:                                       ; preds = %while
  %string_val15 = load i8** @string_val1
  %printf6 = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([4 x i8]*
@fmt26, i32 0, i32 0), i8* %string_val15)
  store i8* getelementptr inbounds ([1 x i8]* @str8, i32 0, i32 0), i8** @string_val1
  br label %while

merge10:                                          ; preds = %while
  br label %merge

else:                                             ; preds = %entry
  br label %merge
}

define void @__action1(i32 %len, i8** %cfields, i32* %field_len) {
entry:
  %len1 = alloca i32
  store i32 %len, i32* %len1
  %cfields2 = alloca i8**
  store i8** %cfields, i8*** %cfields2
  %field_len3 = alloca i32*
  store i32* %field_len, i32** %field_len3
  br label %while

while:                                            ; preds = %while_body, %entry
  %float_val16 = load double* @float_val1
  %tmp7 = fcmp ogt double %float_val16, 0.000000e+00
  br i1 %tmp7, label %while_body, label %merge

while_body:                                       ; preds = %while
  %float_val1 = load double* @float_val1
  %printf = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([4 x i8]*
@fmt311, i32 0, i32 0), double %float_val1)
  %float_val14 = load double* @float_val1
  %tmp = fsub double %float_val14, 1.000000e+00
  store double %tmp, double* @float_val1
  %float_val15 = load double* @float_val1
  %slight_increase_result = call double @slight_increase(double %float_val15)
  store double %slight_increase_result, double* @float_val1
  br label %while

merge:                                            ; preds = %while
  ret void
}

define void @__action0(i32 %len, i8** %cfields, i32* %field_len) {
entry:
  %len1 = alloca i32
  store i32 %len, i32* %len1
```

```
  %cfields2 = alloca i8**
  store i8** %cfields, i8*** %cfields2
  %field_len3 = alloca i32*
  store i32* %field_len, i32** %field_len3
  ret void
}

define void @__init() {
entry:
  store double 4.321000e+00, double* @float_val1
  store i8* getelementptr inbounds ([19 x i8]* @str18, i32 0, i32 0), i8**
@string_val1
  store i32 4, i32* @NPAT
  store i8 32, i8* @FS
  store i8 10, i8* @RS
  store i8* getelementptr inbounds ([4 x i8]* @str19, i32 0, i32 0), i8**
getelementptr inbounds ([4 x i8*]* @__patterns, i32 0, i32 3)
  store i8* getelementptr inbounds ([7 x i8]* @str20, i32 0, i32 0), i8**
getelementptr inbounds ([4 x i8*]* @__patterns, i32 0, i32 2)
  store i8* getelementptr inbounds ([6 x i8]* @str21, i32 0, i32 0), i8**
getelementptr inbounds ([4 x i8*]* @__patterns, i32 0, i32 1)
  store i8* getelementptr inbounds ([6 x i8]* @str22, i32 0, i32 0), i8**
getelementptr inbounds ([4 x i8*]* @__patterns, i32 0, i32 0)
  store void (i32, i8**, i32*)* @__action3, void (i32, i8**, i32*)** getelementptr
inbounds ([4 x void (i32, i8**, i32*)*]* @__actions, i32 0, i32 3)
  store void (i32, i8**, i32*)* @__action2, void (i32, i8**, i32*)** getelementptr
inbounds ([4 x void (i32, i8**, i32*)*]* @__actions, i32 0, i32 2)
  store void (i32, i8**, i32*)* @__action1, void (i32, i8**, i32*)** getelementptr
inbounds ([4 x void (i32, i8**, i32*)*]* @__actions, i32 0, i32 1)
  store void (i32, i8**, i32*)* @__action0, void (i32, i8**, i32*)** getelementptr
inbounds ([4 x void (i32, i8**, i32*)*]* @__actions, i32 0, i32 0)
  ret void
}
```

test-program2.carl

```
/* This code was written by Guy Yardeni and Leon Song */

BEGIN {
	float temp_float;
	array_float myArrayFloat1 = [];
	array_float myArrayFloat2 = [];
	array_string myArrayString1 = [];
	string str = "key1";
	float val = 1;
	myArrayString1["abc"] = "2";
	myArrayString1["def"] = "MEANING OF LIFE?";
	myArrayFloat1[str] = val;
	myArrayFloat2[str] = myArrayFloat1[str] * 42;
}

END {
	string temp_string;
	temp_string = myArrayString1["abc"];
	print_string(temp_string);
	temp_string = myArrayString1["def"];
	print_string(temp_string);

	temp_float = myArrayFloat2[str];
	print_float(temp_float);
}
```

test-program2.ll

```
; ModuleID = 'Carl'

@NPAT = global i32 0
@FS = global i8 0
@RS = global i8 0
@__actions = global [2 x void (i32, i8**, i32*)*] zeroinitializer
@__patterns = global [2 x i8*] zeroinitializer
@temp_float = global double 0.000000e+00
@myArrayFloat1 = global i8* null
@myArrayFloat2 = global i8* null
@myArrayString1 = global i8* null
@str = global i8* null
@val = global double 0.000000e+00
@temp_string = global i8* null
@fmt = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt2 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@fmt3 = private unnamed_addr constant [4 x i8] c"%f\0A\00"
@str1 = private unnamed_addr constant [4 x i8] c"abc\00"
@str2 = private unnamed_addr constant [4 x i8] c"def\00"
@fmt4 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt25 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@fmt36 = private unnamed_addr constant [4 x i8] c"%f\0A\00"
@str7 = private unnamed_addr constant [2 x i8] c"2\00"
@str8 = private unnamed_addr constant [4 x i8] c"abc\00"
@str9 = private unnamed_addr constant [17 x i8] c"MEANING OF LIFE?\00"
@str10 = private unnamed_addr constant [4 x i8] c"def\00"
@fmt11 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt212 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@fmt313 = private unnamed_addr constant [4 x i8] c"%f\0A\00"
@str14 = private unnamed_addr constant [5 x i8] c"key1\00"
@str15 = private unnamed_addr constant [4 x i8] c"END\00"
@str16 = private unnamed_addr constant [6 x i8] c"BEGIN\00"

declare i32 @printf(i8*, ...)

declare double @strlen2(i8*)

declare i8* @create()

declare i32 @array_add_string(i8*, i8*, i8*)

declare i8* @array_retrieve_string(i8*, i8*)

declare i32 @array_add_float(i8*, i8*, double)

declare double @array_retrieve_float(i8*, i8*)

define void @__action1(i32 %len, i8** %cfields, i32* %field_len) {
entry:
  %len1 = alloca i32
```

```
  store i32 %len, i32* %len1
  %cfields2 = alloca i8**
  store i8** %cfields, i8*** %cfields2
  %field_len3 = alloca i32*
  store i32* %field_len, i32** %field_len3
  %myArrayString1 = load i8** @myArrayString1
  %array_retrieve_string = call i8* @array_retrieve_string(i8* %myArrayString1, i8*
getelementptr inbounds ([4 x i8]* @str1, i32 0, i32 0))
  store i8* %array_retrieve_string, i8** @temp_string
  %temp_string = load i8** @temp_string
  %printf = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([4 x i8]* @fmt2,
i32 0, i32 0), i8* %temp_string)
  %myArrayString14 = load i8** @myArrayString1
  %array_retrieve_string5 = call i8* @array_retrieve_string(i8* %myArrayString14, i8*
getelementptr inbounds ([4 x i8]* @str2, i32 0, i32 0))
  store i8* %array_retrieve_string5, i8** @temp_string
  %temp_string6 = load i8** @temp_string
  %printf7 = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([4 x i8]* @fmt2,
i32 0, i32 0), i8* %temp_string6)
  %str = load i8** @str
  %myArrayFloat2 = load i8** @myArrayFloat2
  %array_retrieve_float = call double @array_retrieve_float(i8* %myArrayFloat2, i8*
%str)
  store double %array_retrieve_float, double* @temp_float
  %temp_float = load double* @temp_float
  %printf8 = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([4 x i8]* @fmt3,
i32 0, i32 0), double %temp_float)
  ret void
}

define void @__action0(i32 %len, i8** %cfields, i32* %field_len) {
entry:
  %len1 = alloca i32
  store i32 %len, i32* %len1
  %cfields2 = alloca i8**
  store i8** %cfields, i8*** %cfields2
  %field_len3 = alloca i32*
  store i32* %field_len, i32** %field_len3
  %myArrayString1 = load i8** @myArrayString1
  %array_add_string = call i32 @array_add_string(i8* %myArrayString1, i8*
getelementptr inbounds ([4 x i8]* @str8, i32 0, i32 0), i8* getelementptr inbounds ([2
x i8]* @str7, i32 0, i32 0))
  %myArrayString14 = load i8** @myArrayString1
  %array_add_string5 = call i32 @array_add_string(i8* %myArrayString14, i8*
getelementptr inbounds ([4 x i8]* @str10, i32 0, i32 0), i8* getelementptr inbounds
([17 x i8]* @str9, i32 0, i32 0))
  %val = load double* @val
  %str = load i8** @str
  %myArrayFloat1 = load i8** @myArrayFloat1
  %array_add_float = call i32 @array_add_float(i8* %myArrayFloat1, i8* %str, double
%val)
  %str6 = load i8** @str
  %myArrayFloat17 = load i8** @myArrayFloat1
```

```
  %array_retrieve_float = call double @array_retrieve_float(i8* %myArrayFloat17, i8*
%str6)
  %tmp = fmul double %array_retrieve_float, 4.200000e+01
  %str8 = load i8** @str
  %myArrayFloat2 = load i8** @myArrayFloat2
  %array_add_float9 = call i32 @array_add_float(i8* %myArrayFloat2, i8* %str8, double
%tmp)
  ret void
}

define void @__init() {
entry:
  %create = call i8* @create()
  store i8* %create, i8** @myArrayFloat1
  %create1 = call i8* @create()
  store i8* %create1, i8** @myArrayFloat2
  %create2 = call i8* @create()
  store i8* %create2, i8** @myArrayString1
  store i8* getelementptr inbounds ([5 x i8]* @str14, i32 0, i32 0), i8** @str
  store double 1.000000e+00, double* @val
  store i32 2, i32* @NPAT
  store i8 32, i8* @FS
  store i8 10, i8* @RS
  store i8* getelementptr inbounds ([4 x i8]* @str15, i32 0, i32 0), i8**
getelementptr inbounds ([2 x i8*]* @__patterns, i32 0, i32 1)
  store i8* getelementptr inbounds ([6 x i8]* @str16, i32 0, i32 0), i8**
getelementptr inbounds ([2 x i8*]* @__patterns, i32 0, i32 0)
  store void (i32, i8**, i32*)* @__action1, void (i32, i8**, i32*)** getelementptr
inbounds ([2 x void (i32, i8**, i32*)*]* @__actions, i32 0, i32 1)
  store void (i32, i8**, i32*)* @__action0, void (i32, i8**, i32*)** getelementptr
inbounds ([2 x void (i32, i8**, i32*)*]* @__actions, i32 0, i32 0)
  ret void
}
```

test-program3.carl

```
/* This code was written by Guy Yardeni and Leon Song */

BEGIN {
	float a = 0;
	float b = 0;
	float c = 0;
	float d = 0;
	float e = 0;
	array_float myArray = [];
	string hiker = "key";
}

//Hitchhiker|Hitch Hiker// {
	a = a+1;
	myArray["Hitchhiker"] = a;
}

//Guide// {
	b = b+1;
	myArray["Guide"] = b;
}

//Galaxy// {
	c = c+1;
	myArray["Galaxy"] = c;
}

//Hitchhiker|Guide|Galaxy|Hitch Hiker//{
	d = d+1;
	myArray["Any"] = d;
}

//a*// {
	e = e+1;
	myArray["total"] = e;
}

END {
	print_string("Hitchhiker:");
	print_float(myArray["Hitchhiker"]);

	print_string("Guide:");
	print_float(myArray["Guide"]);

	print_string("Galaxy:");
	print_float(myArray["Galaxy"]);

	print_string("Any:");
	print_float(myArray["Any"]);
```

```
        print_string("total:");
        print_float(myArray["total"]);
}
```

test-program3.ll

```
; ModuleID = 'Carl'

@NPAT = global i32 0
@FS = global i8 0
@RS = global i8 0
@__actions = global [7 x void (i32, i8**, i32*)*] zeroinitializer
@__patterns = global [7 x i8*] zeroinitializer
@a = global double 0.000000e+00
@b = global double 0.000000e+00
@c = global double 0.000000e+00
@d = global double 0.000000e+00
@e = global double 0.000000e+00
@myArray = global i8* null
@hiker = global i8* null
@fmt = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt2 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@fmt3 = private unnamed_addr constant [4 x i8] c"%f\0A\00"
@str = private unnamed_addr constant [12 x i8] c"Hitchhiker:\00"
@str1 = private unnamed_addr constant [11 x i8] c"Hitchhiker\00"
@str2 = private unnamed_addr constant [7 x i8] c"Guide:\00"
@str3 = private unnamed_addr constant [6 x i8] c"Guide\00"
@str4 = private unnamed_addr constant [8 x i8] c"Galaxy:\00"
@str5 = private unnamed_addr constant [7 x i8] c"Galaxy\00"
@str6 = private unnamed_addr constant [5 x i8] c"Any:\00"
@str7 = private unnamed_addr constant [4 x i8] c"Any\00"
@str8 = private unnamed_addr constant [7 x i8] c"total:\00"
@str9 = private unnamed_addr constant [6 x i8] c"total\00"
@fmt10 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt211 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@fmt312 = private unnamed_addr constant [4 x i8] c"%f\0A\00"
@str13 = private unnamed_addr constant [6 x i8] c"total\00"
@fmt14 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt215 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@fmt316 = private unnamed_addr constant [4 x i8] c"%f\0A\00"
@str17 = private unnamed_addr constant [4 x i8] c"Any\00"
@fmt18 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt219 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@fmt320 = private unnamed_addr constant [4 x i8] c"%f\0A\00"
@str21 = private unnamed_addr constant [7 x i8] c"Galaxy\00"
@fmt22 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt223 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@fmt324 = private unnamed_addr constant [4 x i8] c"%f\0A\00"
@str25 = private unnamed_addr constant [6 x i8] c"Guide\00"
```

```
@fmt26 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt227 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@fmt328 = private unnamed_addr constant [4 x i8] c"%f\0A\00"
@str29 = private unnamed_addr constant [11 x i8] c"Hitchhiker\00"
@fmt30 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt231 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@fmt332 = private unnamed_addr constant [4 x i8] c"%f\0A\00"
@fmt33 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt234 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@fmt335 = private unnamed_addr constant [4 x i8] c"%f\0A\00"
@str36 = private unnamed_addr constant [4 x i8] c"key\00"
@str37 = private unnamed_addr constant [4 x i8] c"END\00"
@str38 = private unnamed_addr constant [3 x i8] c"a*\00"
@str39 = private unnamed_addr constant [36 x i8] c"Hitchhiker|Guide|Galaxy|Hitch
Hiker\00"
@str40 = private unnamed_addr constant [7 x i8] c"Galaxy\00"
@str41 = private unnamed_addr constant [6 x i8] c"Guide\00"
@str42 = private unnamed_addr constant [23 x i8] c"Hitchhiker|Hitch Hiker\00"
@str43 = private unnamed_addr constant [6 x i8] c"BEGIN\00"

declare i32 @printf(i8*, ...)

declare double @strlen2(i8*)

declare i8* @create()

declare i32 @array_add_string(i8*, i8*, i8*)

declare i8* @array_retrieve_string(i8*, i8*)

declare i32 @array_add_float(i8*, i8*, double)

declare double @array_retrieve_float(i8*, i8*)

define void @__action6(i32 %len, i8** %cfields, i32* %field_len) {
entry:
  %len1 = alloca i32
  store i32 %len, i32* %len1
  %cfields2 = alloca i8**
  store i8** %cfields, i8*** %cfields2
  %field_len3 = alloca i32*
  store i32* %field_len, i32** %field_len3
  %printf = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([4 x i8]* @fmt2,
i32 0, i32 0), i8* getelementptr inbounds ([12 x i8]* @str, i32 0, i32 0))
  %myArray = load i8** @myArray
  %array_retrieve_float = call double @array_retrieve_float(i8* %myArray, i8*
getelementptr inbounds ([11 x i8]* @str1, i32 0, i32 0))
  %printf4 = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([4 x i8]* @fmt3,
i32 0, i32 0), double %array_retrieve_float)
  %printf5 = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([4 x i8]* @fmt2,
i32 0, i32 0), i8* getelementptr inbounds ([7 x i8]* @str2, i32 0, i32 0))
  %myArray6 = load i8** @myArray
  %array_retrieve_float7 = call double @array_retrieve_float(i8* %myArray6, i8*
getelementptr inbounds ([6 x i8]* @str3, i32 0, i32 0))
```

```
  %printf8 = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([4 x i8]* @fmt3,
i32 0, i32 0), double %array_retrieve_float7)
  %printf9 = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([4 x i8]* @fmt2,
i32 0, i32 0), i8* getelementptr inbounds ([8 x i8]* @str4, i32 0, i32 0))
  %myArray10 = load i8** @myArray
  %array_retrieve_float11 = call double @array_retrieve_float(i8* %myArray10, i8*
getelementptr inbounds ([7 x i8]* @str5, i32 0, i32 0))
  %printf12 = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([4 x i8]*
@fmt3, i32 0, i32 0), double %array_retrieve_float11)
  %printf13 = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([4 x i8]*
@fmt2, i32 0, i32 0), i8* getelementptr inbounds ([5 x i8]* @str6, i32 0, i32 0))
  %myArray14 = load i8** @myArray
  %array_retrieve_float15 = call double @array_retrieve_float(i8* %myArray14, i8*
getelementptr inbounds ([4 x i8]* @str7, i32 0, i32 0))
  %printf16 = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([4 x i8]*
@fmt3, i32 0, i32 0), double %array_retrieve_float15)
  %printf17 = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([4 x i8]*
@fmt2, i32 0, i32 0), i8* getelementptr inbounds ([7 x i8]* @str8, i32 0, i32 0))
  %myArray18 = load i8** @myArray
  %array_retrieve_float19 = call double @array_retrieve_float(i8* %myArray18, i8*
getelementptr inbounds ([6 x i8]* @str9, i32 0, i32 0))
  %printf20 = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([4 x i8]*
@fmt3, i32 0, i32 0), double %array_retrieve_float19)
  ret void
}

define void @__action5(i32 %len, i8** %cfields, i32* %field_len) {
entry:
  %len1 = alloca i32
  store i32 %len, i32* %len1
  %cfields2 = alloca i8**
  store i8** %cfields, i8*** %cfields2
  %field_len3 = alloca i32*
  store i32* %field_len, i32** %field_len3
  %e = load double* @e
  %tmp = fadd double %e, 1.000000e+00
  store double %tmp, double* @e
  %e4 = load double* @e
  %myArray = load i8** @myArray
  %array_add_float = call i32 @array_add_float(i8* %myArray, i8* getelementptr
inbounds ([6 x i8]* @str13, i32 0, i32 0), double %e4)
  ret void
}

define void @__action4(i32 %len, i8** %cfields, i32* %field_len) {
entry:
  %len1 = alloca i32
  store i32 %len, i32* %len1
  %cfields2 = alloca i8**
  store i8** %cfields, i8*** %cfields2
  %field_len3 = alloca i32*
  store i32* %field_len, i32** %field_len3
  %d = load double* @d
  %tmp = fadd double %d, 1.000000e+00
```

```
  store double %tmp, double* @d
  %d4 = load double* @d
  %myArray = load i8** @myArray
  %array_add_float = call i32 @array_add_float(i8* %myArray, i8* getelementptr
inbounds ([4 x i8]* @str17, i32 0, i32 0), double %d4)
  ret void
}

define void @__action3(i32 %len, i8** %cfields, i32* %field_len) {
entry:
  %len1 = alloca i32
  store i32 %len, i32* %len1
  %cfields2 = alloca i8**
  store i8** %cfields, i8*** %cfields2
  %field_len3 = alloca i32*
  store i32* %field_len, i32** %field_len3
  %c = load double* @c
  %tmp = fadd double %c, 1.000000e+00
  store double %tmp, double* @c
  %c4 = load double* @c
  %myArray = load i8** @myArray
  %array_add_float = call i32 @array_add_float(i8* %myArray, i8* getelementptr
inbounds ([7 x i8]* @str21, i32 0, i32 0), double %c4)
  ret void
}

define void @__action2(i32 %len, i8** %cfields, i32* %field_len) {
entry:
  %len1 = alloca i32
  store i32 %len, i32* %len1
  %cfields2 = alloca i8**
  store i8** %cfields, i8*** %cfields2
  %field_len3 = alloca i32*
  store i32* %field_len, i32** %field_len3
  %b = load double* @b
  %tmp = fadd double %b, 1.000000e+00
  store double %tmp, double* @b
  %b4 = load double* @b
  %myArray = load i8** @myArray
  %array_add_float = call i32 @array_add_float(i8* %myArray, i8* getelementptr
inbounds ([6 x i8]* @str25, i32 0, i32 0), double %b4)
  ret void
}

define void @__action1(i32 %len, i8** %cfields, i32* %field_len) {
entry:
  %len1 = alloca i32
  store i32 %len, i32* %len1
  %cfields2 = alloca i8**
  store i8** %cfields, i8*** %cfields2
  %field_len3 = alloca i32*
  store i32* %field_len, i32** %field_len3
  %a = load double* @a
  %tmp = fadd double %a, 1.000000e+00
```

```
  store double %tmp, double* @a
  %a4 = load double* @a
  %myArray = load i8** @myArray
  %array_add_float = call i32 @array_add_float(i8* %myArray, i8* getelementptr
inbounds ([11 x i8]* @str29, i32 0, i32 0), double %a4)
  ret void
}

define void @__action0(i32 %len, i8** %cfields, i32* %field_len) {
entry:
  %len1 = alloca i32
  store i32 %len, i32* %len1
  %cfields2 = alloca i8**
  store i8** %cfields, i8*** %cfields2
  %field_len3 = alloca i32*
  store i32* %field_len, i32** %field_len3
  ret void
}

define void @__init() {
entry:
  store double 0.000000e+00, double* @a
  store double 0.000000e+00, double* @b
  store double 0.000000e+00, double* @c
  store double 0.000000e+00, double* @d
  store double 0.000000e+00, double* @e
  %create = call i8* @create()
  store i8* %create, i8** @myArray
  store i8* getelementptr inbounds ([4 x i8]* @str36, i32 0, i32 0), i8** @hiker
  store i32 7, i32* @NPAT
  store i8 32, i8* @FS
  store i8 10, i8* @RS
  store i8* getelementptr inbounds ([4 x i8]* @str37, i32 0, i32 0), i8**
getelementptr inbounds ([7 x i8*]* @__patterns, i32 0, i32 6)
  store i8* getelementptr inbounds ([3 x i8]* @str38, i32 0, i32 0), i8**
getelementptr inbounds ([7 x i8*]* @__patterns, i32 0, i32 5)
  store i8* getelementptr inbounds ([36 x i8]* @str39, i32 0, i32 0), i8**
getelementptr inbounds ([7 x i8*]* @__patterns, i32 0, i32 4)
  store i8* getelementptr inbounds ([7 x i8]* @str40, i32 0, i32 0), i8**
getelementptr inbounds ([7 x i8*]* @__patterns, i32 0, i32 3)
  store i8* getelementptr inbounds ([6 x i8]* @str41, i32 0, i32 0), i8**
getelementptr inbounds ([7 x i8*]* @__patterns, i32 0, i32 2)
  store i8* getelementptr inbounds ([23 x i8]* @str42, i32 0, i32 0), i8**
getelementptr inbounds ([7 x i8*]* @__patterns, i32 0, i32 1)
  store i8* getelementptr inbounds ([6 x i8]* @str43, i32 0, i32 0), i8**
getelementptr inbounds ([7 x i8*]* @__patterns, i32 0, i32 0)
  store void (i32, i8**, i32*)* @__action6, void (i32, i8**, i32*)** getelementptr
inbounds ([7 x void (i32, i8**, i32*)*]* @__actions, i32 0, i32 6)
  store void (i32, i8**, i32*)* @__action5, void (i32, i8**, i32*)** getelementptr
inbounds ([7 x void (i32, i8**, i32*)*]* @__actions, i32 0, i32 5)
  store void (i32, i8**, i32*)* @__action4, void (i32, i8**, i32*)** getelementptr
inbounds ([7 x void (i32, i8**, i32*)*]* @__actions, i32 0, i32 4)
  store void (i32, i8**, i32*)* @__action3, void (i32, i8**, i32*)** getelementptr
inbounds ([7 x void (i32, i8**, i32*)*]* @__actions, i32 0, i32 3)
```

```
  store void (i32, i8**, i32*)* @__action2, void (i32, i8**, i32*)** getelementptr
inbounds ([7 x void (i32, i8**, i32*)*]* @__actions, i32 0, i32 2)
  store void (i32, i8**, i32*)* @__action1, void (i32, i8**, i32*)** getelementptr
inbounds ([7 x void (i32, i8**, i32*)*]* @__actions, i32 0, i32 1)
  store void (i32, i8**, i32*)* @__action0, void (i32, i8**, i32*)** getelementptr
inbounds ([7 x void (i32, i8**, i32*)*]* @__actions, i32 0, i32 0)
  ret void
}
```

PASS TESTS	filename
Add Float Key Value to Array	test-array_add_float.carl
Add String Key Value to Array	test-array_add_string.carl
Create an Array	test-array_create.carl
Retrieve Float Value from Array	test-array_retrieve_float.carl
Retrieve Float Values from Array	test-array_retrieve_floats.carl
Retrieve String Values from Array	test-array_retrieve_string.carl
Retrieve String Values from Array	test-array_retrieve_strings.carl
Array Scope	test-array_scope.carl
Array Scope Modification	test-array_scope_modification.carl
Overrite Array String Value	test-array_string_overwrite.carl
Float Array Syntax	test-array_syntax_float.carl
String Array Syntax	test-array_syntax_string.carl
Empty BEGIN	test-empty_begin.carl
Empty BEGIN empty END	test-empty_begin_empty_end.carl
Float Equality	test-equality_float.carl
Float Declaration	test-float1.carl
Float Initialization	test-float1_init.carl
Float Addition Declaration	test-float_addition.carl
Float Addition Initialization	test-float_addition_init.carl
Float Division Declaration	test-float_division.carl
Float Division Initialization	test-float_division_init.carl
Float Multiplication Declaration	test-float_multiplication.carl
Float Multiplication Initialization	test-float_multiplication_init.carl

Float Subtraction Declaration	test-float_subtraction.carl
Float Subtraction Initialization	test-float_subtraction_init.carl
Float Function	test-function_float.carl
String Function	test-function_string.carl
Void Function	test-function_void.carl
Hello World	test-hello.carl
If Statement	test-if.carl
If Else Statement	test-if_else.carl
If Else If statement	test-if_elseif.carl
Float If Control	test-if_float.carl
String If Control	test-if_string.carl
String Variable If Control	test-if_string_var.carl
For Loop	test-loop_for.carl
While Loop	test-loop_while.carl
Demo Program 1	test-program1.carl
Demo Program 2	test-program2.carl
Demo Program 3	test-program3.carl
Single Regex	test-regex_abc.carl
Regex One or More	test-regex_one_or_more.carl
Regex Range Lowercase Letter	test-regex_range_lowercase_letter.carl
Regex Range Numbers and Letters	test-regex_range_number_abc_multi.carl
Regex Range Number	test-regex_range_number.carl
Regex Range Multiple Numbers	test-regex_range_number_multi.carl
Regex Range Uppercase Letters	test-regex_range_uppercase_letter.carl
String	test-string1.carl
String Scope	test-string_scope.carl

FAIL TESTS	filename

Bad Array Assign	`fail-array_bad_assign.carl`
Array Redeclaration	`fail-array_redeclaration.carl`
Empty File	`fail-empty.carl`
End Before Begin	`fail-end_before_begin.carl`
Bad Float Assign	`fail-float_bad_assign.carl`
Float Redeclaration	`fail-float_redeclaration.carl`
Bad Int Assign	`fail-int_bad_assign.carl`
Int Redeclaration	`fail-int_redeclaration.carl`
No Begin	`fail-only_end.carl`
Bad Regex	`fail-regex_abc.carl`
Bad Regex One or More	`fail-regex_one_or_more.carl`
Bad Regex Range Lowercase Letter	`fail-regex_range_lowercase_letter.carl`
Bad Regex Range Number	`fail-regex_range_number.carl`
Bad Regex Range Uppercase Letter	`fail-regex_range_uppercase_letter.carl`
Bad String Assign	`fail-string_bad_assign.carl`
String Redeclaration	`fail-string_redeclaration.carl`

The CARL test suite went through each language feature and tested basic functionality. We began with testing each of the datatypes: float, string, array. The array datatype had tests for each of the value types: float and string. We tested all of the arithmetic operations on float. The if/else statements were tested with boolean, float, and string values. Functions were tested with each return type, float, string, and void. Further, regex was tested with many different types of patterns and with multiple patterns in one program.

The test suite was vital in identifying and troubleshooting bugs in the language. Each feature required continuous testing to identify where the issue was coming from and to make sure that other features remained functional during troubleshooting. We used numerous different methods for troubleshooting bugs. For parser issues, we use the verbose flag with ocamalyacc. We would further use the parser.output file to trace through errors. Our .carl test files also saved the generated files and created .diff files. This allowed us to check what the program was outputting if different from the expected output.

The buildcarl file was used to make the CARL compiler, the carl engine, and the array library. Then, the buildcarlp ran buildcarl and used the components to build a carl executable. The testall.sh file iterated through each test, ran it through buildcarlp to create the executable, and

ran the executable to check if the generated matched with the expected output. Our large pipeline made building carl executables more complicated, but these files simplified the process.

Darren Hakimi wrote all the basic test cases and built the testall.sh script and buildcarlp. Keir Lauritzen wrote the buildcarl file and made modifications to buildcarlp. Guy Yardeni wrote test cases for functions and arrays. Leon Song also worked with guy in writing test cases for arrays.

7. Lessons Learned

Leon Song

Get to a testable version as soon as possible:

Working on our project was especially difficult because it felt like we were coding blind. The CARL engine is more complex than usual, and it took us a long time to put it together. Keir took the lead in setting up the engine, while the rest of us tried to take a stab at modular areas of the compiler e.g. get regex strings working, write wrapper functions for libraries we used. However, while we were writing code, it was near impossible to know if what we were writing was correct. This had several implications, for one it turned out that quite a bit of what we wrote had to be re-done. Writing the code was difficult in the first place, but returning to code I wrote a month ago and trying to figure out why I was now getting errors was even more tedious and slow. Another unexpected implication was that there was much less motivation to code. When you have a test suite you can run and give you feedback, it gives a momentum to your work. I write a couple of lines, run the test, fix errors, and move on to the next portion. If our code is untestable, writing one section of code is hard because you don't know if previous sections are working, and you feel much less confident and motivated to push on.

Make sure to just finish your part, and trust teammates to finish theirs:

I've known that I tend to be easily anxious, but doing this project made me realize that not only am I anxious about my own code, but I can tend to be anxious about my teammate's coding as well. While this might be useful in small doses, I think this really prevented me from doing my own job as well. In a sense it's much better to just focus on delivering your own part, and just let teammates finish theirs in their own time. If someone has bumps and is blocked, of course help them, but don't be a micro-manager.

Darren Hakimi

This project was my first major group programming project. I have never used github in a group before this project. I was the least experienced member in the group. There was a large learning curve for me. I would spend endless nights staring at the MicroC code, in attempt of getting a better understanding. At the start of the project, I was entirely clueless. I had to continuously ask the other group members questions just to get a basic understanding of what I was to work on.

My group members provided me with a lot of help. Additionally, after staring at MicroC code and figuring things out, I began looking at previous group projects that implemented similar features. This resource was a huge help to me. I downloaded all the group projects from Professor Edwards' page, allowing me to search for keywords. By seeing other projects, I learned the different ways of implementing features, and was able come up with ways of implementing features in CARL. Some features only required us to add a few lines, like 3, in addition to the MicroC base. Those few lines would take hours, and, initially, days to figure out.

The last month of the assignment, my group really pulled things together. It took us way longer than expected to get Hello World working because our language had such a large pipeline. Once, we got Hello World running, everything started to move along and I finally started to see the results of my work. Most importantly, this project provided me with knowledge on how to work with github and group members, who I did not know prior to the project, and all that goes into to building a language. My understanding of programming languages has largely expanded. With this new understanding, it is significantly easier for me to learn new programming languages. Prior to this project, I did not know what an LRM was. Now, I wrote an LRM. LRMs are important resources for learning new languages. PLT has prepared me for future languages that I will have to learn, and has provided my with job-like training for group work.

Keir Lauritzen

Through the course and this project, I have become much more comfortable with how programming languages actually work. While I doubt I'll become a compiler writer, it is much easier to learn new programming languages and understand how it would be implemented. In effect, they've been demystified for me and I'm looking to try a couple of new ones this summer. OCaml was a good learning experience because it is so different from C and other imperative languages that it forces you to think in a different way and grow. (I do believe it is much easier to write compilers in OCaml than C.)

We struggled at the beginning of the project to get our Hello World program up because we needed to have carl_engine completely integrated. This required designing the architecture, writing carl_engine.cpp, semant, implementing arrays of function pointers and strings in codegen, and other small changes throughout the chain. This was probably the hardest part of the project and it just took dozens of hours of coding, learning, and research to build. Unfortunately, it took nearly a month to finally get working. During that time, everyone was working on different parts, but without an integrated project we did not get into a good rhythm and everyone's productivity was low. Once we have an integrated product, failing tests and clear picture of the architecture, everyone became much more productive.

I do not know if we could have done it any differently based on this project and the way we ended up doing the design. I would really recommend that teams avoid bottlenecks in the early stages and choose projects and designs where they can start right away. This project has

about a 3 week learning curve to get comfortable with OCaml, compilers, and LLVM and the bottleneck delayed everyone else's progression until later in the project.

Guy Yardeni

Working on this project has given me a better understanding of programming languages and compilers. I learned the multi-step process of building a compiler and designing a language. Learning OCaml was difficult at first, but I became more proficient at it the more time I spent using it. Having a great team to work with was extremely helpful.

This project won't be simple. However, if you take it one task at a time and work together with your teammates, you'll succeed. Frequent discussions and meetings with your teammates is essential for this project. Asking for help from teammates or TAs if you're stuck on a task is much better than wasting hours or days without making any progress. Learning and using OCaml and Llvm will be hard at first, but you'll get better at it. Just keep on truckin'.

8. Appendix

scanner.mll

```
(* Ocamllex scanner for MicroC *)

(* This code was original from MicroC and modified by:
Keir Lauritzen, Leon Song, Darren Hakimi, and Guy Yardeni*)

{
       open Parser
       type is_pat = REGEX | NORMAL
       let state_ref = ref NORMAL

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

let alphabet = ['a'-'z' 'A'-'Z']
let escape = '\\' ['\\' ''' '"' 'n' 'r' 't']
let escape_char = ''' (escape) '''
let ascii = ([' '-'!' '#'-'[' ']'-'~'])
let digit = ['0'-'9']
let string_lit = '"' ( (ascii | escape)* as s) '"'
let character = ''' ( ascii | digit ) '''


(* define some common character classes here *)
let digits = ['0' - '9']+
let id = ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']*
(* Identifiers can start with alphabetic characters only *)

rule token pat = parse
  [' ' '\t' '\r' '\n'] { token pat lexbuf } (* Whitespace *)
| "/*"     { comment pat lexbuf }           (* Comments *)
| '('      { LPAREN }
| ')'      { RPAREN }
| '{'      { LBRACE }
| '}'      { RBRACE }
| ';'      { SEMI }
| ','      { COMMA }
| '+'      { PLUS }
| '-'      { MINUS }
| '['     { LBRACK }
| ']'     { RBRACK }
| '*'      { TIMES }
| '/'      { DIVIDE }
| '='      { ASSIGN }
| "=="     { EQ }
| "!="     { NEQ }
| '<'      { LT }
| "<="     { LEQ }
| ">"      { GT }
| ">="     { GEQ }
```

```
| "&&"     { AND }
| "||"     { OR }
| "!"      { NOT }
| "if"     { IF }
| "else"   { ELSE }
| "for"    { FOR }
| "while"  { WHILE }
| "return" { RETURN }
| "function" { FUNCTION }
| "//"       { pat := REGEX ; FSLASH }
| "BEGIN"   { BEGIN }
| "END"     { END }
| "bool"    { BOOL }
| "TRUE"    { TRUE }
| "FALSE"   { FALSE }
| "void"   { VOID }
| "array_float"  { ARRAY_F }
| "array_string" { ARRAY_S }
| "float"  { FLOAT }
| "string" { STRING }
| "RS"     { RS }
| "FS"     { FS }
| "NPAT"     { NPAT }
| '$''0'     { RECORD }
| '$'['1' - '9']['0' - '9']+  { RFIELD }
| "__patterns"  { PATTERNS }
| "__action"['0'-'9']+   { ACTIONS }
| "__actions"   { ACTIONS2 }
| "destroy()"   { DESTROY }
| "__init()" { INIT}
| ['-']?digits as lxm {FLOAT_L(float_of_string lxm)}
| ['-']?['0' - '9']+['.']['0' - '9']+ as lxm {FLOAT_L(float_of_string lxm)}
| ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']* as lxm { ID(lxm) }
| '"' [^ '"']* '"' as lxm { STRING_L(lxm) }
| eof { EOF }
| _ as char { raise (Failure("illegal character " ^ Char.escaped char)) }

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

and regex pat = parse
| "//" { pat := NORMAL ; FSLASH }
| ['\\']['"' '.' '?' '|' '^' ']' '[' '(' ')' '-' '\\' '$' '*'] as lxm{
REGEX_STRING(lxm) }
| '.' { PERIOD }
| '?' { QUEST }
| '^' { CARROT }
| '|' { VERT }
| '-' { RMINUS }
| '*' { TIMES }
| '[' { LBRACK }
| ']' { RBRACK }
| '(' { LPAREN }
```

```
| ')' { RPAREN }
| [^'"' '/' '.' '?' '|' '^' ']' '[' '(' ')' '-' '\\' '$' '*'] as lxm {
REGEX_STRING((Char.escaped lxm)) }


(*The function to get the next token; checks to see if it is scanning a regex pattern
block*)
{
       let next_token lexbuf = match !state_ref with
           | NORMAL -> token state_ref lexbuf
           | REGEX -> regex state_ref lexbuf
}
```

parser.mly

```
/* Ocamlyacc parser for CARL
This code was original from MicroC and modified by:
Keir Lauritzen, Leon Song, Guy Yardeni, and Darren Hakimi

*/

%{
open Ast
let quote_remover a = String.sub a 1 ((String.length a) - 2);;

%}

%token SEMI LPAREN RPAREN LBRACE RBRACE COMMA
%token PLUS MINUS TIMES DIVIDE ASSIGN NOT
%token PERIOD QUEST CARROT VERT LBRACK RBRACK
%token EQ NEQ LT LEQ GT GEQ TRUE FALSE AND OR
%token IF ELSEIF ELSE
%token RETURN BOOL VOID FLOAT
%token FOR WHILE CONCAT
%token RS FS NR NF RSTART RLENGTH
%token EOF
%token BEGIN END FUNCTION FSLASH STRING ARRAY_F ARRAY_S
%token RECORD RFIELD PATTERNS ACTIONS
%token RMINUS
%token <string> ID
%token <float> FLOAT_L
%token <string> STRING_L
%token <string> REGEX_STRING

%nonassoc NOELSE
%nonassoc ELSE
%right ASSIGN
%left OR
```

```
%left AND
%left EQ NEQ
%left LT GT LEQ GEQ
%left PLUS MINUS
%left TIMES DIVIDE
%right NOT NEG

%start program
%type <Ast.program> program

%%

program:
    fdecl_list rule_list EOF { { func_decls =List.rev $1; rules= List.rev $2 } }


fdecl_list:
    /* nothing */  { [] }
  | fdecl_list fdecl { $2 :: $1 }

fdecl:
   FUNCTION typ ID LPAREN formals_opt RPAREN LBRACE vdecl_list stmt_list RBRACE
     { { typ = $2;
   fname = $3;
   formals = $5;
   locals = List.rev $8;
   body = List.rev $9 } }

typ:
  | VOID { Void }
  | STRING { String_t }
  | FLOAT { Float }
  | ARRAY_F { Array_f }
  | ARRAY_S { Array_s }

formals_opt:
    /* nothing */ { [] }
  | formal_list   { List.rev $1 }

formal_list:
    typ ID                   { [($1,$2)] }
  | formal_list COMMA typ ID { ($3,$4) :: $1 }

vdecl_list:
    /* nothing */    { [] }
  | vdecl_list vdecl { $2 :: $1 }

vdecl:
    typ ID assign_opt SEMI { ($1, $2, $3) }

assign_opt:
    /* nothing */  { Noexpr }
  | ASSIGN expr { $2 }
```

```
rule_list:
    /* nothing */  { [] }
  | rule_list rdecl { $2 :: $1 }

rdecl:
    pattern action { $1, $2 }

pattern:
    BEGIN { Begin }
    | END { End }
    | FSLASH regex_sequence FSLASH { RegexPattern($2) }

action:
    LBRACE vdecl_list stmt_list RBRACE  { List.rev $2, List.rev $3  }

stmt_list:
    /* nothing */  { [] }
  | stmt_list stmt { $2 :: $1 }

stmt:
    | expr SEMI { Expr $1 }
    | RETURN expr SEMI { Return($2) }
    | LBRACE stmt_list RBRACE { Block(List.rev $2) }
    | IF LPAREN expr RPAREN stmt %prec NOELSE { If($3, $5, Block([])) }
    | IF LPAREN expr RPAREN stmt ELSE stmt    { If($3, $5, $7) }
    | FOR LPAREN expr_opt SEMI expr SEMI expr_opt RPAREN stmt
       { For($3, $5, $7, $9) }
    | WHILE LPAREN expr RPAREN stmt { While($3, $5) }

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

expr:
      STRING_L          { String_Lit(quote_remover($1)) }
    | FLOAT_L           { Float_Lit($1) }
    | LBRACK kvps_f RBRACK { Array_F_Lit($2) }
    | LBRACK kvps_s RBRACK { Array_S_Lit($2) }
    | ID               { Id($1) }
    | expr PLUS   expr { Binop($1, Add,   $3) }
    | expr MINUS  expr { Binop($1, Sub,   $3) }
    | expr TIMES  expr { Binop($1, Mult,  $3) }
    | expr DIVIDE expr { Binop($1, Div,   $3) }
    | expr EQ     expr { Binop($1, Equal, $3) }
    | expr NEQ    expr { Binop($1, Neq,   $3) }
    | expr LT     expr { Binop($1, Less,  $3) }
    | expr LEQ    expr { Binop($1, Leq,   $3) }
    | expr GT     expr { Binop($1, Greater, $3) }
    | expr GEQ    expr { Binop($1, Geq,   $3) }
    | expr AND    expr { Binop($1, And,   $3) }
    | expr OR     expr { Binop($1, Or,    $3) }
    | MINUS expr %prec NEG { Unop(Neg, $2) }
```

```
    | NOT expr         { Unop(Not, $2) }
    | ID ASSIGN expr   { Assign($1, $3) }
    | ID LPAREN actuals_opt RPAREN { Call($1, $3) }
    | LPAREN expr RPAREN { $2 }
    | LBRACK RBRACK     { Call("create", [])}
    | ID LBRACK expr RBRACK { Retrieve($1, $3)}
    | ID LBRACK expr RBRACK ASSIGN expr {Array_Assign($1, $3, $6)}

kvps_f:
    kvp_f        { [$1] }
  | kvps_f COMMA kvp_f { $3 :: $1 }

kvps_s:
    kvp_s        { [$1] }
  | kvps_s COMMA kvp_s { $3 :: $1 }

kvp_f:
  STRING_L ASSIGN FLOAT_L { $1, $3 }

kvp_s:
  STRING_L ASSIGN STRING_L { $1, $3 }

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

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


/*Start of Regex*/

regex_sequence:
    regex {$1}
  | regex regex_sequence {$1^$2}

regex:
    REGEX_STRING {$1}
  | PERIOD {"."}
  | LPAREN regex_sequence RPAREN {"("^$2^")"}
  | LBRACK regex_set_sequence RBRACK {"["^$2^"]"}
  | regex QUEST {$1^"?"}
  | regex PLUS {$1^"+"}
  | regex TIMES {$1^"*"}
  | regex VERT  {$1^"|"}

regex_set:
    REGEX_STRING {$1}
  | REGEX_STRING RMINUS REGEX_STRING{$1^"-"^$3}
  | CARROT regex_set {"^"^$2}
  | LBRACK regex_set_sequence RBRACK {"["^$2^"]"}

regex_set_sequence:
```

```
    regex_set {$1}
  | regex_set regex_set_sequence {$1^$2}

/*End of Regex */
```

ast.ml

```
(* Abstract Syntax Tree and functions for printing it *)

(* This code was original from MicroC and modified by:
Keir Lauritzen, Leon Song, Guy Yardeni, Darren Hakimi*)

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

type uop = Neg | Not

type typ = String_t | Void | Float | Array_f | Array_s

type bind = typ * string

type expr =
    | String_Lit of string
    | Float_Lit of float
    | Array_F_Lit of (string * float) list
    | Array_S_Lit of (string * string) list
    | Id of string
    | Binop of expr * op * expr
    | Unop of uop * expr
    | Assign of string * expr
    | Call of string * expr list
    | Noexpr
    | Retrieve of string * expr
    | Array_Assign of string * expr * expr

type bind_init = typ * string * expr

type stmt =
    | Block of stmt list
    | Expr of expr
    | Return of expr
    | If of expr * stmt * stmt
    | For of expr * expr * expr * stmt
    | While of expr * stmt

type action =
    bind_init list * stmt list

type pattern = Begin | End | RegexPattern of string
```

```
type rule = pattern * action

type func_decl = {
    typ : typ;
    fname : string;
    formals : bind list;
    locals : bind_init list;
    body : stmt list;
}

type program = {
    func_decls: func_decl list;
    rules: rule list;
}

let string_of_program input_program = "42"
```

semant.ml

```
(* This code was written by:
Keir Lauritzen, Leon Song, and Guy Yardeni*)


module A = Ast
module S = Sast

module StringMap = Map.Make(String)

let convert program =
    let conv_typ = function
    | A.String_t -> S.String_t
    | A.Void -> S.Void
    | A.Bool -> S.Bool
    | A.Float -> S.Float
    | A.Array_f -> S.Array_f
    | A.Array_s -> S.Array_s in

    let conv_bind = function
    | (A.String_t, x) -> (S.String_t, x)
    | (A.Void, x) -> (S.Void, x)
    | (A.Bool, x) -> (S.Bool, x)
    | (A.Float, x) -> (S.Float, x)
    | (A.Array_f, x) -> (S.Array_f, x)
    | (A.Array_s, x) -> (S.Array_s, x) in

    let conv_bind_init = function
    | (A.String_t, x, _) -> (S.String_t, x)
    | (A.Void, x, _) -> (S.Void, x)
    | (A.Bool, x, _) -> (S.Bool, x)
```

```
    | (A.Float, x, _) -> (S.Float, x)
    | (A.Array_f, x, _) -> (S.Array_f, x)
    | (A.Array_s, x, _) -> (S.Array_s, x) in


    let rec conv_binop = function
    | A.Add     -> S.Add
    | A.Sub     -> S.Sub
    | A.Mult    -> S.Mult
    | A.Div     -> S.Div
    | A.And     -> S.And
    | A.Or      -> S.Or
    | A.Equal   -> S.Equal
    | A.Neq     -> S.Neq
    | A.Less    -> S.Less
    | A.Leq     -> S.Leq
    | A.Greater -> S.Greater
    | A.Geq     -> S.Geq in

    let rec conv_unop = function
    | A.Neg     -> S.Neg
    | A.Not     -> S.Not in

    let global_vars =
        let extract_globals l (pattern, action) =
            let extr_glob_action l (binds, stmts) =
            l@List.map conv_bind_init binds in
        extr_glob_action l action in
    List.fold_left extract_globals [] program.Ast.rules in

    let conv_patt = function
    | A.Begin -> S.String_Lit "BEGIN"
    | A.End -> S.String_Lit "END"
    | A.RegexPattern s -> S.String_Lit s in

    (* this can be used to declare function pointers*)
    let n_pat = List.fold_left (fun x l -> x+1) 0 program.Ast.rules in

    let interface_globals = [(S.Int, "NPAT"); (S.Char, "FS"); (S.Char, "RS");
        ( S.S_arr_t (S.Action_t, n_pat), "__actions");
        ( S.S_arr_t (S.String_t, n_pat), "__patterns") ] in

    let assign_patterns = fun
        (l,n) (pattern, action) ->
            let patt = conv_patt pattern in
        (S.Expr (S.Assign_array ("__patterns", n, patt))::l, n+1) in
    let (assigned_patts,_) = List.fold_left assign_patterns ([],0) program.Ast.rules
in


    (*let enum_patts =
            let enum_patt = fun (x,l) patt -> (x-1, patt:l) in
    List_fold_left enum_patts (n_pat, []) assigned_patts in
```

```
    let checked_patts =
        let check_patt
    List.map check_patt enum_patts *)


    let symbols = List.fold_left (fun m (t, n) -> StringMap.add n t m)
    StringMap.empty (interface_globals @ global_vars)
    in

    let rec conv_expr = function
    | A.Bool_Lit b -> S.Bool_Lit b
    | A.String_Lit s -> S.String_Lit s
    | A.Float_Lit f -> S.Float_Lit f
    | A.Array_F_Lit afl -> S.Array_F_Lit afl
    | A.Array_S_Lit asl -> S.Array_S_Lit asl
    | A.Retrieve (name, key) -> (match (StringMap.find name symbols) with
        | S.Array_f -> S.Retrieve_Float(name, conv_expr key)
        | S.Array_s -> S.Retrieve_String(name, conv_expr key)
        | _ -> raise (Invalid_argument "Not Valid"))
    | A.Array_Assign (name, key, value) -> (match (StringMap.find name symbols) with
        | S.Array_f -> S.Array_Assign_Float (name, conv_expr key, conv_expr value)
        | S.Array_s -> S.Array_Assign_String (name, conv_expr key, conv_expr value)
        | _ -> raise (Invalid_argument "Not Valid"))
    | A.Noexpr -> S.Noexpr
    | A.Id s -> S.Id s
    | A.Binop (e1, op, e2) ->
        let e1' = conv_expr e1
        and e2' = conv_expr e2 in
        S.Binop (e1', (conv_binop op), e2')
    | A.Unop(op, e) ->
        let e' = conv_expr e in
        S.Unop ((conv_unop op), e')
    | A.Assign (s, e) -> let e' = conv_expr e in
         S.Assign (s,e')
    | A.Call (f, act) -> S.Call (f, List.map conv_expr act) in


    let convert_bool = function
    | S.Bool_Lit b -> S.Bool_Lit b
    | S.String_Lit s -> if (String.length s) > 0 then S.Bool_Lit true else S.Bool_Lit
false
    | S.Float_Lit f -> if f >= 1.0 then S.Bool_Lit true else S.Bool_Lit false
    | S.Array_F_Lit afl -> raise (Invalid_argument "Not Valid")
    | S.Array_S_Lit asl -> raise (Invalid_argument "Not Valid")
    | S.Noexpr -> raise (Invalid_argument "Must provide an expression")
    | S.Id s -> (match (StringMap.find s symbols) with
                | S.Float -> S.Binop (S.Id s, S.Greater, S.Float_Lit 0.0)
                | S.String_t -> S.Binop (S.Call ("strlen", [(S.Id s)]), S.Greater,
S.Float_Lit 0.0)
                | _ -> raise (Invalid_argument "Not Valid"))
    | S.Binop (e1, op, e2) -> S.Binop (e1, op, e2) (*S.Binop (e1', op, e2')*)
    | S.Unop (op, e) -> S.Unop (op, e) (*
```

```
        | S.Unop(op, e) ->
            let e' = expr builder e in
            (match op with
            | S.Neg     -> S.Binop ()
            | S.Not     -> S.Not    *)
    | S.Assign (s,e) ->  S.Assign (s,e)  (*S.Binop (s, >=, 1.0) *)
    | S.Call (f, act) -> S.Call (f, act) (*S.Bionp () *)in

    let rec conv_stmt = function
    | A.Block l -> S.Block (List.map conv_stmt l)
    | A.Expr x -> S.Expr (conv_expr x)
    | A.Return x -> S.Return (conv_expr x)
    | A.If (x, s1, s2) -> S.If (convert_bool (conv_expr x), conv_stmt s1, conv_stmt
s2)
    | A.For (x1, x2, x3, s) -> S.For ( conv_expr x1, convert_bool (conv_expr
x2),conv_expr x3, conv_stmt s)
    | A.While (x, s) -> S.While (convert_bool (conv_expr x), conv_stmt s) in

    let init_assign = function
    | (_, s, A.Noexpr) -> S.Expr (S.Noexpr)
    | (_, s, e) -> S.Expr (S.Assign(s, conv_expr e)) in



    let conv_func_decl = function
    func_decl ->
    {   S.typ = conv_typ func_decl.A.typ;
        S.fname = func_decl.A.fname;
        S.formals = List.map conv_bind func_decl.A.formals;
        S.locals = List.map conv_bind_init func_decl.A.locals;
        S.body = (List.map init_assign func_decl.A.locals) @ (List.map conv_stmt
func_decl.A.body);
    } in

    let func_decls = List.map conv_func_decl program.Ast.func_decls in
(*
    let declare_patterns = fun
        (l,n) (pattern, action) ->  (
            (S.String_t, "__pattern" ^ (string_of_int n))
    ::l, n-1) in
    let (decl_patterns,_) = List.fold_left declare_patterns ([],) program.Ast.rules in
*)

    (*let action_formals = [(S.Int,"len"); (S.Int, "cfields"); (S.Int, "field_len")]
in *)
    let action_formals = [(S.Int,"len"); (S.Charpp, "cfields"); (S.Intp, "field_len")]
in
    let define_actions = fun
        (l,n) (pattern, action) ->
        ({  S.typ = S.Void;
            S.fname = "__action" ^ string_of_int n;
            S.formals = action_formals;
            S.locals = [];
            S.body = List.map conv_stmt (snd action);
```

```
        } :: l, n+1) in
    let (action_decls,_) = List.fold_left define_actions ([],0) program.Ast.rules in


    let assign_actions = fun
        (l,n) (pattern, action) ->
        let action_name = "__action" ^ string_of_int n in
        (S.Expr (S.Assign_func_array ("__actions", n, action_name)) :: l, n+1) in
    let (assigned_actions,_) = List.fold_left assign_actions ([],0) program.Ast.rules
in

(*
    let patt1 = S.String_Lit "BEGIN" in
    let pll = S.Assign ("__pattern2", patt1) in
    let pll2 =  in
    let init_patt_assign =
         [S.Expr pll; S.Expr pll2]  in *)

(*

    let act1 = (S.Action_t, "__act1") in


    let all = S.Assign_func ("__act1", "__action1") in
    let all2 = S.Assign_func_array ("__actions", 1, "__action1") in
    let init_act_assign =
         [S.Expr all] @ [S.Expr all2]   in *)


    (*let blue = S.Assign_func_array *)


    let assigned_globals =
        let extract_globals l (pattern, action) =
            let extr_glob_action l (binds, stmts) =
            l@List.map init_assign binds in
        extr_glob_action l action in
    List.fold_left extract_globals [] program.Ast.rules in

    let assigned_vars = [
        S.Expr(S.Assign("NPAT", S.Int_Lit n_pat ));
        S.Expr(S.Assign("FS", S.Char_Lit ' ' ));
        S.Expr(S.Assign("RS", S.Char_Lit '\n' ));
    ] in

    let init_function = {
        S.typ = S.Void;
        S.fname = "__init";
        S.formals = [];
        S.locals = [];
        S.body = assigned_globals @ assigned_vars @ assigned_patts @ assigned_actions;
    } in

{ S.var_decls = interface_globals @ global_vars;
```

```
  S.func_decls = func_decls @ action_decls @ [init_function] ;
(*S.func_decls = func_decls @ action_decls; *)
}
```

check.ml

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

(* This code was original from MicroC and modified by:
Keir Lauritzen, Leon Song, and Guy Yardeni*)

open Sast

module StringMap = Map.Make(String)

(* Semantic checking of a program. Returns void if successful,
   throws an exception if something is wrong.

   Check each global variable, then check each function *)

let check program =

  (* Raise an exception if the given list has a duplicate *)
  let report_duplicate exceptf list =
    let rec helper = function
       n1 :: n2 :: _ when n1 = n2 -> raise (Failure (exceptf n1))
     | _ :: t -> helper t
     | [] -> ()
    in helper (List.sort compare list)
  in

  (* Raise an exception if a given binding is to a void type *)
  let check_not_void exceptf = function
      (Void, n) -> raise (Failure (exceptf n))
    | _ -> ()
  in

  (* Raise an exception of the given rvalue type cannot be assigned to
     the given lvalue type *)
  let check_assign lvaluet rvaluet err =
     if lvaluet == rvaluet || lvaluet == Array_f || lvaluet == Array_s
     then lvaluet
   else raise err
  in

  (**** Checking Global Variables ****)


  List.iter (check_not_void (fun n -> "illegal void global " ^ n))
program.Sast.var_decls;
```

```
  report_duplicate (fun n -> "duplicate global " ^ n) (List.map snd
program.Sast.var_decls);

  (**** Checking Functions ****)

  if List.mem "print" (List.map (fun fd -> fd.fname) program.Sast.func_decls)
  then raise (Failure ("function print may not be defined")) else ();

  report_duplicate (fun n -> "duplicate function " ^ n)
    (List.map (fun fd -> fd.fname) program.Sast.func_decls);

  (* Function declaration for a named function *)
  let built_in_decls =  StringMap.add "print"
     { typ = Void; fname = "print"; formals = [(Int, "x")];
       locals = []; body = [] } (StringMap.add "printb"
     { typ = Void; fname = "printb"; formals = [(Bool, "x")];
       locals = []; body = [] } (StringMap.add "printc"
     { typ = Void; fname = "printc"; formals = [(Char, "x")];
       locals = []; body = [] } (StringMap.add "print_string"
     { typ = Void; fname = "print_string"; formals = [(String_t, "x")];
       locals = []; body = [] } (StringMap.add "print_float"
     { typ = Void; fname = "print_float"; formals = [(Float, "x")];
       locals = []; body = [] } (StringMap.add "create"
     { typ = Void; fname = "create"; formals = [];
       locals = []; body = [] } (StringMap.add "array_add_float"
     { typ = Float; fname = "array_add_float"; formals = [(Void, "x"); (String_t,
"y"); (Float, "z")];
       locals = []; body = [] } (StringMap.add "array_add_string"
     { typ = Float; fname = "array_add_string"; formals = [(Void, "x"); (String_t,
"y"); (String_t, "z")];
       locals = []; body = [] } (StringMap.add "array_retrieve_float"
     { typ = Float; fname = "array_retrieve_float"; formals = [(Void, "x"); (String_t,
"y")];
       locals = []; body = [] } (StringMap.add "array_retrieve_string"
     { typ = String_t; fname = "array_retrieve_string"; formals = [(Void, "x");
(String_t, "y")];
       locals = []; body = [] } (StringMap.add "strlen"
     { typ = Void; fname = "strlen"; formals = [(String_t, "x")];
       locals = []; body = [] } (StringMap.singleton "printbig"
     { typ = Void; fname = "printbig"; formals = [(Int, "x")];
       locals = []; body = [] }))))))))))))

   in

  let function_decls = List.fold_left (fun m fd -> StringMap.add fd.fname fd m)
                       built_in_decls program.Sast.func_decls
  in

  let function_decl s = try StringMap.find s function_decls
       with Not_found -> raise (Failure ("unrecognized function " ^ s))
  in

  let check_function func =
```

```
    List.iter (check_not_void (fun n -> "illegal void formal " ^ n ^
      " in " ^ func.fname)) func.formals;

    report_duplicate (fun n -> "duplicate formal " ^ n ^ " in " ^ func.fname)
      (List.map snd func.formals);

    List.iter (check_not_void (fun n -> "illegal void local " ^ n ^
      " in " ^ func.fname)) func.locals;

    report_duplicate (fun n -> "duplicate local " ^ n ^ " in " ^ func.fname)
      (List.map snd func.locals);

    (* Type of each variable (global, formal, or local *)
    let symbols = List.fold_left (fun m (t, n) -> StringMap.add n t m)
       StringMap.empty (program.Sast.var_decls @ func.formals @ func.locals )
    in

    let type_of_identifier s =
      try StringMap.find s symbols
      with Not_found -> raise (Failure ("undeclared identifier " ^ s))
    in

    (* Return the type of an expression or throw an exception *)


    let rec expr = function
    | Int_Lit _ -> Int
    | Bool_Lit _ -> Bool
    | String_Lit _ -> String_t
    | Float_Lit _ -> Float
    | Char_Lit _ -> Char
    | Id s -> type_of_identifier s
    | Binop(e1, op, e2) as e -> let t1 = expr e1 and t2 = expr e2 in
       (match op with
                Add | Sub | Mult | Div when t1 = Float && t2 = Float -> Float
       | Equal | Neq when t1 = t2 -> Bool
       | Less | Leq | Greater | Geq when (t1 = Float || t1 = Void) && t2 = Float ->
Bool
       | And | Or when t1 = Bool && t2 = Bool -> Bool
             | _ -> raise (Failure ("illegal binary operator " ^
                   string_of_typ t1 ^ " " ^ string_of_op op ^ " " ^
                   string_of_typ t2 ^ " in " ^ string_of_expr e))
        )
    | Unop(op, e) as ex -> let t = expr e in
        (match op with
          Neg when t = Float -> Float
        | Not when t = Bool -> Bool
            | _ -> raise (Failure ("illegal unary operator " ^ string_of_uop op ^
                        string_of_typ t ^ " in " ^ string_of_expr ex))
        )
    | Noexpr -> Void
    | Assign(var, e) as ex -> let lt = type_of_identifier var
                              and rt = expr e in
```

```
      check_assign lt rt (Failure ("illegal assignment " ^ string_of_typ lt ^
                                   " = " ^ string_of_typ rt ^ " in " ^
                                   string_of_expr ex))
  | Call(fname, actuals) as call -> let fd = function_decl fname in
       if List.length actuals != List.length fd.formals then
         raise (Failure ("expecting " ^ string_of_int
           (List.length fd.formals) ^ " arguments in " ^ string_of_expr call))
       else
         (*List.iter2 (fun (ft, _) e -> let et = expr e in
            ignore (check_assign ft et
              (Failure ("illegal actual argument found " ^ string_of_typ et ^
              " expected " ^ string_of_typ ft ^ " in " ^ string_of_expr e))))
           fd.formals actuals; *)
         fd.typ
  | Assign_func (string1, string2) -> Bool
  | Assign_func_array (string1, int1, string2) -> Bool
  | Assign_array (string1, int1, e) -> Bool
  | Retrieve_Float (name, key) -> Float
  | Retrieve_String (name, key) -> String_t
  | Array_Assign_Float (name, key, value) -> Float
  | Array_Assign_String (name, key, value) -> String_t
  | _ -> ignore(print_endline "JUST CHECKING"); Void
  in

  let check_bool_expr e = if expr e != Bool
   then raise (Failure ("expected Boolean expression in " ^ string_of_expr e))
   else () in

  (* Verify a statement or throw an exception *)
  let rec stmt = function
     Block sl -> let rec check_block = function
         [Return _ as s] -> stmt s
       | Return _ :: _ -> raise (Failure "nothing may follow a return")
       | Block sl :: ss -> check_block (sl @ ss)
       | s :: ss -> stmt s ; check_block ss
       | [] -> ()
      in check_block sl
    | Expr e -> ignore (expr e)
    | Return e -> let t = expr e in if t = func.typ then () else
       raise (Failure ("return gives " ^ string_of_typ t ^ " expected " ^
                       string_of_typ func.typ ^ " in " ^ string_of_expr e))

    | If(p, b1, b2) -> check_bool_expr p; stmt b1; stmt b2
    | For(e1, e2, e3, st) -> ignore (expr e1); check_bool_expr e2;
                             ignore (expr e3); stmt st
    | While(p, s) -> check_bool_expr p; stmt s
  in

  stmt (Block func.body)

in
List.iter check_function program.Sast.func_decls
```

sast.ml

```
(* Abstract Syntax Tree and functions for printing it *)

(* This code was original from MicroC (by way of ast.ml) and modified by:
Keir Lauritzen, Leon Song, and Guy Yardeni*)

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

type uop = Neg | Not

type typ = Int | String_t | Void | Bool | Char | Charp | Charpp | Intp | Float | Array_f |
Array_s | Action_t | S_arr_t of typ * int

type bind = typ * string

type expr =
    | Int_Lit of int
    | Bool_Lit of bool
    | String_Lit of string
    | Float_Lit of float
    | Array_F_Lit of (string * float) list
    | Array_S_Lit of (string * string) list
    | Retrieve_Float of string * expr
    | Retrieve_String of string * expr
    | Array_Assign_Float of string * expr * expr
    | Array_Assign_String of string * expr * expr
    | Char_Lit of char
    | Id of string
    | Binop of expr * op * expr
    | Unop of uop * expr
    | Assign of string * expr
    | Assign_array of string * int * expr
    | Call of string * expr list
    | Noexpr
    | Assign_func of string * string
    | Assign_func_array of string * int * string

type stmt =
    | Block of stmt list
    | Expr of expr
    | Return of expr
    | If of expr * stmt * stmt
    | For of expr * expr * expr * stmt
    | While of expr * stmt


type func_decl = {
    typ : typ;
    fname : string;
```

```
    formals : bind list;
    locals : bind list;
    body : stmt list;
}

type program = {
    var_decls : bind list;
    func_decls : func_decl list;
}

let rec string_of_typ = function
    | Int -> "int"
    | Bool -> "bool"
    | Void -> "void"
    | Char -> "char"
    | String_t -> "string"
    | Charp -> "char*"
    | Charpp -> "char**"
    | Intp -> "int*"
    | Float -> "float"
    | Array_f -> "array float"
    | Array_s -> "array string"
    | Action_t -> "function pointer"

let string_of_op = function
    | Add -> "+"
    | Sub -> "-"
    | Mult -> "*"
    | Div -> "/"
    | Equal -> "=="
    | Neq -> "!="
    | Less -> "<"
    | Leq -> "<="
    | Greater -> ">"
    | Geq -> ">="
    | And -> "&&"
    | Or -> "||"

let string_of_uop = function
    | Neg -> "-"
    | Not -> "!"

let rec string_of_expr = function
    | Int_Lit(l) -> string_of_int l
    | Bool_Lit(true) -> "true"
    | Bool_Lit(false) -> "false"
    | String_Lit(str) -> "" ^ str
    | Float_Lit(l) -> string_of_float l
    | Char_Lit(c) -> Char.escaped c
    | Id(s) -> s
    | Binop(e1, o, e2) ->
        string_of_expr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_expr e2
    | Unop(o, e) -> string_of_uop o ^ string_of_expr e
    | Assign(v, e) -> v ^ " = " ^ string_of_expr e
```

```
    | Call(f, el) ->
        f ^ "(" ^ String.concat ", " (List.map string_of_expr el) ^ ")"
    | Noexpr -> ""
    | _ -> "No_match"
```

codegen.ml

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

LLVM tutorial: Make sure to read the OCaml version of the tutorial

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

Detailed documentation on the OCaml LLVM library:

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

This code was original from MicroC and modified by:
Keir Lauritzen, Leon Song, and Guy Yardeni

*)

module L = Llvm
module S = Sast

module StringMap = Map.Make(String)

let context = L.global_context () ;;
let the_module = L.create_module context "Carl"
and i32_t   = L.i32_type  context
and i8_t    = L.i8_type   context
and i1_t    = L.i1_type   context
and f_t     = L.double_type context
and void_t  = L.void_type context;;

let void_p  = L.pointer_type i8_t;;
let str_t   = L.pointer_type i8_t ;;
let charpp_t = L.pointer_type str_t;;
let intp_t = L.pointer_type i32_t;;
let arr_t   = L.i64_type context ;;
let action_t = L.pointer_type (L.function_type void_t [| i32_t; charpp_t; intp_t |]);;
(* let actions_t = L.pointer_type (L.pointer_type (L.function_type void_t [| i32_t; |]));; *)

(*L.dump_type str_t;;*)

let translate program =

    let ltype_of_typ = function
```

```
    | S.Int -> i32_t
    | S.Bool -> i1_t
    | S.String_t -> str_t
    | S.Void -> void_t
    | S.Action_t -> action_t
    | S.Charpp -> charpp_t
    | S.Intp -> intp_t
    | S.Char -> i8_t
    | S.Float -> f_t
    | S.Array_f -> void_p
    | S.Array_s -> void_p in

   (* | S.Float -> i32_t *)
   (* | S.Array -> arr_t *)

    let rec global_init = function
    | S.Int -> L.const_int i32_t 0
    | S.Bool -> L.const_int i1_t 0
    | S.Char -> L.const_int i8_t 0
    | S.String_t -> L.const_pointer_null str_t
    | S.Void -> L.const_int void_t 0
    | S.Float -> L.const_float f_t 0.0
    | S.Action_t -> L.const_pointer_null action_t
    | S.S_arr_t (t, i) -> L.const_array (ltype_of_typ t) (Array.make i (global_init t))
    | S.Array_f -> L.const_pointer_null void_p
    | S.Array_s -> L.const_pointer_null void_p in


    (* Declare each global variable; remember its value in a map *)
    let global_vars =
        let global_var m (t, n) =
            let init = global_init t in
        StringMap.add n (L.define_global n init the_module) m in
    List.fold_left global_var StringMap.empty program.Sast.var_decls in

    (* Declare printf(), which the print built-in function will call *)
    let printf_t = L.var_arg_function_type i32_t [| L.pointer_type i8_t |] in
    let printf_func = L.declare_function "printf" printf_t the_module in

    let strlen_t = L.function_type f_t [| str_t |] in
    let strlen_func = L.declare_function "strlen2" strlen_t the_module in

    let create_t = L.function_type void_p [||] in
    let create_func = L.declare_function "create" create_t the_module in

    let array_add_string_t = L.function_type i32_t [|void_p ; str_t ; str_t|] in
    let array_add_string_func = L.declare_function "array_add_string" array_add_string_t
the_module in

    let array_retrieve_string_t = L.function_type str_t [|void_p ; str_t|] in
    let array_retrieve_string_func = L.declare_function "array_retrieve_string"
array_retrieve_string_t the_module in

    let array_add_float_t = L.function_type i32_t [|void_p ; str_t ; f_t|] in
```

```
    let array_add_float_func = L.declare_function "array_add_float" array_add_float_t
the_module in

    let array_retrieve_float_t = L.function_type f_t [|void_p ; str_t|] in
    let array_retrieve_float_func = L.declare_function "array_retrieve_float"
array_retrieve_float_t the_module in

    (* Define each function (arguments and return type) so we can call it *)
    let function_decls =
        let function_decl m fdecl =
            let name = fdecl.S.fname
            and formal_types =
               Array.of_list (List.map (fun (t,_) -> ltype_of_typ t) fdecl.S.formals) in
            let ftype = L.function_type (ltype_of_typ fdecl.S.typ) formal_types in
        StringMap.add name (L.define_function name ftype the_module, fdecl) m in
    List.fold_left function_decl StringMap.empty program.S.func_decls in

    (* Fill in the body of the given function *)
    let build_function_body fdecl =
        let (the_function, _) = StringMap.find fdecl.S.fname function_decls in
        let builder = L.builder_at_end context (L.entry_block the_function) in

        let int_format_str = L.build_global_stringptr "%d\n" "fmt" builder in
        let str_format_str = L.build_global_stringptr "%s\n" "fmt2" builder in
        let flt_format_str = L.build_global_stringptr "%f\n" "fmt3" builder in

        (* Construct the function's "locals": formal arguments and locally
        declared variables.  Allocate each on the stack, initialize their
        value, if appropriate, and remember their values in the "locals" map *)
        let local_vars =
            let add_formal m (t, n) p = L.set_value_name n p;
            let local = L.build_alloca (ltype_of_typ t) n builder in
           ignore (L.build_store p local builder);
           StringMap.add n local m in

        let add_local m (t, n) =
               let local_var = L.build_alloca (ltype_of_typ t) n builder in
        StringMap.add n local_var m in

        let formals = List.fold_left2 add_formal StringMap.empty fdecl.S.formals
            (Array.to_list (L.params the_function)) in
        List.fold_left add_local formals fdecl.S.locals in

        (* Return the value for a variable or formal argument *)
        let lookup n =  try StringMap.find n local_vars
                        with Not_found -> try StringMap.find n global_vars
                         with Not_found -> (*ignore(print_endline "in lookup");*)
StringMap.find n local_vars
        (*with Not_found -> fst (StringMap.find n function_decls) *)in

        let lookup_func n = fst (StringMap.find n function_decls) in

        let build_string e builder =
            let str = L.build_global_stringptr e "str" builder in
```

```
    let null = L.const_int i32_t 0 in
L.build_in_bounds_gep str [| null |] "str" builder in


(* Construct code for an expression; return its value *)
let rec expr builder = function
| S.Int_Lit i -> L.const_int i32_t i
| S.Bool_Lit b -> L.const_int i1_t (if b then 1 else 0)
| S.String_Lit s -> build_string s builder
| S.Float_Lit f -> L.const_float f_t f
| S.Char_Lit c -> L.const_int i8_t (Char.code c)
| S.Noexpr -> L.const_int i32_t 0
| S.Id s -> L.build_load (lookup s) s builder
| S.Binop (e1, op, e2) -> (*(try
       (let e1' = expr builder e1
       and e2' = expr builder e2 in
       (match op with
       | S.Add     -> L.build_add
       | S.Sub     -> L.build_sub
       | S.Mult    -> L.build_mul
    | S.Div     -> L.build_sdiv
       | S.And     -> L.build_and
       | S.Or      -> L.build_or
       | S.Equal   -> L.build_icmp L.Icmp.Eq
       | S.Neq     -> L.build_icmp L.Icmp.Ne
       | S.Less    -> L.build_icmp L.Icmp.Slt
       | S.Leq     -> L.build_icmp L.Icmp.Sle
       | S.Greater -> L.build_icmp L.Icmp.Sgt
       | S.Geq     -> L.build_icmp L.Icmp.Sge
       ) e1' e2' "tmp" builder) with _ ->
    ignore(print_endline "INEXPR2"); *)
    let e1' = expr builder e1
    and e2' = expr builder e2 in
    (match op with
    | S.Add     -> L.build_fadd
    | S.Sub     -> L.build_fsub
    | S.Mult    -> L.build_fmul
    | S.Div     -> L.build_fdiv
    | S.And     -> L.build_and
    | S.Or      -> L.build_or
    | S.Equal   -> L.build_fcmp L.Fcmp.Oeq
    | S.Neq     -> L.build_fcmp L.Fcmp.One
    | S.Less    -> L.build_fcmp L.Fcmp.Olt
    | S.Leq     -> L.build_fcmp L.Fcmp.Ole
    | S.Greater -> L.build_fcmp L.Fcmp.Ogt
    | S.Geq     -> L.build_fcmp L.Fcmp.Oge
    ) e1' e2' "tmp" builder  (* )     *)
| S.Unop(op, e) ->
   let e' = expr builder e in
       (match op with
       | S.Neg     -> L.build_neg
    | S.Not     -> L.build_not) e' "tmp" builder
| S.Assign (s, e) -> let e' = expr builder e in (*ignore(print_endline s);*)
                  ignore (L.build_store e' (lookup s) builder); e'
```

```
        | S.Assign_array (s, i, e) -> let e' = expr builder e in
            let index =L.build_in_bounds_gep (lookup s) [| L.const_int i32_t 0; L.const_int
i32_t i |] "str" builder in
            ignore (L.build_store e' index builder); e'
        | S.Call ("print_int", [e]) ->
              L.build_call printf_func [| int_format_str ; (expr builder e) |]
              "printf" builder
        | S.Call ("print_string", [e]) ->
            L.build_call printf_func [| str_format_str ; (expr builder e) |]
           "printf" builder
        | S.Call ("print_float", [e]) ->
            L.build_call printf_func [| flt_format_str ; (expr builder e) |]
           "printf" builder
        | S.Call ("strlen", [a]) ->
            L.build_call strlen_func [|expr builder a|] "strlen" builder
        | S.Call ("create", act) -> (*ignore(print_endline "create");*)
            L.build_call create_func [||] "create" builder
        | S.Call ("array_add_string", [a; b; c]) ->
            L.build_call array_add_string_func [|(expr builder a) ; (expr builder b) ; (expr
builder c)|]
            "array_add_string" builder
        | S.Call ("array_retrieve_string", [a; b]) ->
            L.build_call array_retrieve_string_func [|(expr builder a) ; (expr builder b)|]
            "array_retrieve_string" builder
        | S.Call ("array_add_float", [a; b; c]) ->
            L.build_call array_add_float_func [|(expr builder a) ; (expr builder b) ; (expr
builder c)|]
            "array_add_float" builder
        | S.Array_Assign_Float (name, key, value) ->
            L.build_call array_add_float_func [|(L.build_load (lookup name) name builder) ;
(expr builder key) ; (expr builder value)|]
            "array_add_float" builder
        | S.Array_Assign_String (name, key, value) ->
            L.build_call array_add_string_func [|(L.build_load (lookup name) name builder) ;
(expr builder key) ; (expr builder value)|]
            "array_add_string" builder
        | S.Call ("array_retrieve_float", [a; b]) ->
            L.build_call array_retrieve_float_func [|(expr builder a) ; (expr builder b)|]
            "array_retrieve_float" builder
        | S.Retrieve_Float (name, key) ->
            L.build_call array_retrieve_float_func [|(L.build_load (lookup name) name
builder) ; (expr builder key)|]
            "array_retrieve_float" builder
        | S.Retrieve_String (name, key) ->
            L.build_call array_retrieve_string_func [|(L.build_load (lookup name) name
builder) ; (expr builder key)|]
            "array_retrieve_string" builder
        | S.Call (f, act) -> (*ignore(print_endline ("Function:"^f));*)
               let (fdef, fdecl) = StringMap.find f function_decls in
                  let actuals = List.rev (List.map (expr builder) (List.rev act)) in
                  let result = (match fdecl.S.typ with S.Void -> ""
                                                     | _ -> f ^ "_result") in
            L.build_call fdef (Array.of_list actuals) result builder
        | S.Assign_func (s, f) -> ignore(L.build_store (lookup_func f) (lookup s) builder);
```

```
                L.const_int i32_t 0
        | S.Assign_func_array (s, i, f) ->
            let index =L.build_in_bounds_gep (lookup s) [| L.const_int i32_t 0; L.const_int
i32_t i |] "str" builder in
            ignore(L.build_store (lookup_func f) index builder);
                    L.const_int i32_t 0
        in

        (* Invoke "f builder" if the current block doesn't already
        have a terminal (e.g., a branch). *)
        let add_terminal builder f =
            match L.block_terminator (L.insertion_block builder) with
              | Some _ -> ()
            | None -> ignore (f builder)
        in

        (* Build the code for the given statement; return the builder for
           the statement's successor *)
        let rec stmt builder = function
           | S.Block sl -> List.fold_left stmt builder sl
            | S.Expr e -> ignore (expr builder e); builder
            | S.Return e -> ignore (match fdecl.S.typ with
               | S.Void -> L.build_ret_void builder
               | _ -> L.build_ret (expr builder e) builder); builder
            | S.If (predicate, then_stmt, else_stmt) ->
                let bool_val = expr builder predicate in
               let merge_bb = L.append_block context "merge" the_function in

               let then_bb = L.append_block context "then" the_function in
               add_terminal (stmt (L.builder_at_end context then_bb) then_stmt)
                   (L.build_br merge_bb);

               let else_bb = L.append_block context "else" the_function in
               add_terminal (stmt (L.builder_at_end context else_bb) else_stmt)
                   (L.build_br merge_bb);

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

            | S.While (predicate, body) ->
               let pred_bb = L.append_block context "while" the_function in
               ignore (L.build_br pred_bb builder);

               let body_bb = L.append_block context "while_body" the_function in
               add_terminal (stmt (L.builder_at_end context body_bb) body)
                   (L.build_br pred_bb);

               let pred_builder = L.builder_at_end context pred_bb in
               let bool_val = expr pred_builder predicate in

               let merge_bb = L.append_block context "merge" the_function in
               ignore (L.build_cond_br bool_val body_bb merge_bb pred_builder);
               L.builder_at_end context merge_bb
```

```
            | S.For (e1, e2, e3, body) -> stmt builder
                  ( S.Block [S.Expr e1 ; S.While (e2, S.Block [body ; S.Expr e3]) ] )
        in

        (* Build the code for each statement in the function *)
        let builder = stmt builder (S.Block fdecl.S.body) in

        (* Add a return if the last block falls off the end *)
    add_terminal builder (match fdecl.S.typ with
    | S.Void -> L.build_ret_void
    | t -> L.build_ret (L.const_int (ltype_of_typ t) 0))
    in

List.iter build_function_body program.Sast.func_decls;
the_module
```

carl.ml

```
       (* Top-level of the MicroC compiler: scan & parse the input,
          check the resulting AST, generate LLVM IR, and dump the module *)

       (* This code was original from MicroC and modified by:
       Keir Lauritzen, Leon Song, and Guy Yardeni*)


       type action = Ast | Compile


       let _ =
           let action = if Array.length Sys.argv > 1 then
               List.assoc Sys.argv.(1) [ ("-a", Ast);
                                         ("-c", Compile) ]
           else Compile in
           let lexbuf = Lexing.from_channel stdin in
           let ast = Parser.program Scanner.next_token lexbuf in
           match action with
               Ast -> print_string (Ast. string_of_program ast)
               | Compile ->
                   let sast = Semant.convert ast in
                   let _ = Check.check sast in
                   let m = Codegen.translate sast in
                   Llvm_analysis.assert_valid_module m;
                   print_string (Llvm.string_of_llmodule m);;
```

Makefile (carl)

```
# This code was original from MicroC and modified by:
# Keir Lauritzen, Leon Song and Guy Yardeni

#OBJS = ast.cmx codegen.cmx parser.cmx scanner.cmx carl.cmx turn_to_c.cmx
#OBJS = ast.cmx sast.cmx parser.cmx scanner.cmx semant.cmx carl.cmx
OBJS = ast.cmx sast.cmx parser.cmx scanner.cmx semant.cmx codegen.cmx check.cmx
carl.cmx

carlc : $(OBJS)
	ocamlfind ocamlopt -linkpkg -package llvm -package llvm.analysis $(OBJS) -o
carlc

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 $<

%.cmx : %.ml
	ocamlfind ocamlopt -c -package llvm $<

.PHONY : clean
clean :
	ocamlbuild -clean
	rm -rf scanner.ml parser.ml parser.mli carlc
	rm -rf *.cmx *.cmi *.cmo *.o *~ *.s *.output *.ll

engine : carl_engine.cpp
	g++ -std=c++11 -g -c carl_engine.cpp

ast.cmo :
ast.cmx :
sast.cmo : ast.cmo
sast.cmx : ast.cmx
codegen.cmo : sast.cmo
codegen.cmx : sast.cmx
carl.cmo : scanner.cmo parser.cmi sast.cmo codegen.cmo ast.cmo check.cmo semant.cmo
carl.cmx : scanner.cmx parser.cmx sast.cmx codegen.cmx ast.cmx check.cmx semant.cmx
#carl.cmo : scanner.cmo parser.cmi ast.cmo sast.cmo
#carl.cmx : scanner.cmx parser.cmx ast.cmx sast.cmx
parser.cmo : ast.cmo parser.cmi
parser.cmx : ast.cmx parser.cmi
scanner.cmo : parser.cmi
scanner.cmx : parser.cmx
parser.ci : ast.cmo
check.cmo : sast.cmo parser.cmi
check.cmx : sast.cmx parser.cmi
semant.cmo : sast.cmo parser.cmi
```

```
semant.cmx : sast.cmx parser.cmi
```

carl_engine.cpp

```
#include "carl_engine.h"
#include <stdio.h>

/*
This code was written by Keir Lauritzen
*/

using namespace std;

extern "C" void destroy_all();

void make_c_arrays(vector<string> fields, string record, char**& cfields, int*&
cfield_len){
    int n_fields = fields.size()+1;
    //cout << n_fields << '\n';
    cfields = (char **) malloc(sizeof(char *) * n_fields);
    cfield_len = (int*) malloc(sizeof(int) * n_fields);

    int n_char = record.size()+1;
    cfields[0] = (char*) malloc(sizeof(char) * n_char);
    cfield_len[0] = n_char;
    record.copy( cfields[0],record.size() );
    cfields[0][n_char-1]='\0';

    int i = 1;
    for (string field : fields){
        n_char = field.size()+1;
        cfields[i] = (char*) malloc(sizeof(char) * n_char);
        cfield_len[i] = n_char;
        field.copy( cfields[i],field.size() );
        cfields[i][n_char-1]='\0';
        ++i;
    }
    return;
}

void print_c_arrays(int len, char** cfields, int* cfield_len){
    for (int i = 0; i < len; i++){
        for (int j = 0; j < cfield_len[i]; j++)
            cout << cfields[i][j];
        cout << '\n';
    }
    return;
```

```
}

void delete_c_arrays(int len, char** cfields, int* cfield_len){
    for (int i = 0; i < len; i++){
        free(cfields[i]);
    }
    free(cfields);
    free(cfield_len);
}

vector<string> generate_fields(string record) {
    vector<string> fields;
    string field = "";
    for (auto r : record){
        if (r != FS)
            field.push_back(r);
        else {
            if (field.size() > 0) {
                fields.push_back(field);
                field.clear();
            }
        }
    }
    fields.push_back(field); // No terminator on the last field
    return fields;
}

int main(int argc, char *argv[] ){
    if (argc == 1) {
        cout<< "No filename provided.\n" << '\n';
        return 0;
    }
    __init();
    ifstream ifs(argv[1]);
    if (!ifs) {
        cout << "Couldn't open file: " << argv[1] << '\n';
        throw;
    }

    for (int i = 0; i < NPAT; ++i){
        ifs.clear();         //http://www.cplusplus.com/forum/beginner/30644/
        ifs.seekg(0, ios::beg);

        //cout<< "Before string" << '\n';
        //printf("%s\n", __patterns[i]);
        //cout << __patterns[i] << '\n';
        string pat (__patterns[i]);


        if (!pat.compare("BEGIN")){
            if (i != 0)
                throw "BEGIN must be first pattern";
          //cout<< "Before action" << '\n';
          (*__actions[i])(0, NULL, NULL);
```

```
        }
        else if (!pat.compare("END")) {
            if (i != NPAT-1)
                 throw "END must be last pattern";
            (*__actions[i])(0, NULL, NULL);
        }
        else {
            for(string record; getline(ifs, record, RS); ){
                vector<string> fields = generate_fields(record);

                int len = fields.size()+1;
                char **cfields;
                int *cfield_len;
                make_c_arrays(fields, record, cfields, cfield_len);
                //print_c_arrays(len, cfields, cfield_len);

                regex expr{pat};

                if (regex_search(record, expr)){
                    (*__actions[i])(len, cfields, cfield_len);
                }
                delete_c_arrays(len,cfields,cfield_len);
            }
        }
    }
    //delete_pattern_actions();
    destroy_all();
    return 0;
}
```

carl_engine.h

```
#include <string>
#include <iostream>
#include <fstream>
#include <vector>
#include <regex>
#include <exception>

/* This code was written by Keir Lauritzen */

extern int NPAT;
extern char RS;
extern char FS;
extern char* __patterns[10000];
//extern char* pattern1;
extern void (*__actions[10000])(int ,char**, int*);

extern "C" {
    void __init(void);
```

```
//    void delete_pattern_actions(void);
    //void action1(const int len,char** fields, int* field_len);
    //void action2(const int len,char** fields, int* field_len);
}
//extern void action2(const char* fields, ...);
```

(carl_engine) Makefile

```
carl_engine.o : carl_engine.cpp
	g++ -std=c++11 -g -c carl_engine.cpp

.PHONY : clean
clean :
	rm -rf *.cmx *.cmi *.cmo *.o *~ *.ll carl_engine
```

Hashmap.c

https://github.com/petewarden/c_hashmap

```
/*
 * Generic map implementation.
 */
#include "hashmap.h"

#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#define INITIAL_SIZE (256)
#define MAX_CHAIN_LENGTH (8)

/* We need to keep keys and values */
typedef struct _hashmap_element{
	char* key;
	int in_use;
	any_t data;
} hashmap_element;

/* A hashmap has some maximum size and current size,
 * as well as the data to hold. */
typedef struct _hashmap_map{
	int table_size;
	int size;
	hashmap_element *data;
} hashmap_map;

/*
 * Return an empty hashmap, or NULL on failure.
 */
map_t hashmap_new() {
	hashmap_map* m = (hashmap_map*) malloc(sizeof(hashmap_map));
	if(!m) goto err;

	m->data = (hashmap_element*) calloc(INITIAL_SIZE, sizeof(hashmap_element));
	if(!m->data) goto err;

	m->table_size = INITIAL_SIZE;
	m->size = 0;

	return m;
	err:
		if (m)
			hashmap_free(m);
		return NULL;
}

/* The implementation here was originally done by Gary S. Brown.  I have
   borrowed the tables directly, and made some minor changes to the
   crc32-function (including changing the interface). //ylo */

  /* ============================================================= */
  /*  COPYRIGHT (C) 1986 Gary S. Brown.  You may use this program, or       */
```

```
  /*  code or tables extracted from it, as desired without restriction.     */
  /*                                                                        */
  /*  First, the polynomial itself and its table of feedback terms.  The    */
  /*  polynomial is                                                         */
  /*  X^32+X^26+X^23+X^22+X^16+X^12+X^11+X^10+X^8+X^7+X^5+X^4+X^2+X^1+X^0   */
  /*                                                                        */
  /*  Note that we take it "backwards" and put the highest-order term in    */
  /*  the lowest-order bit.  The X^32 term is "implied"; the LSB is the     */
  /*  X^31 term, etc.  The X^0 term (usually shown as "+1") results in      */
  /*  the MSB being 1.                                                      */
  /*                                                                        */
  /*  Note that the usual hardware shift register implementation, which     */
  /*  is what we're using (we're merely optimizing it by doing eight-bit    */
  /*  chunks at a time) shifts bits into the lowest-order term.  In our     */
  /*  implementation, that means shifting towards the right.  Why do we     */
  /*  do it this way?  Because the calculated CRC must be transmitted in    */
  /*  order from highest-order term to lowest-order term.  UARTs transmit   */
  /*  characters in order from LSB to MSB.  By storing the CRC this way,    */
  /*  we hand it to the UART in the order low-byte to high-byte; the UART   */
  /*  sends each low-bit to hight-bit; and the result is transmission bit   */
  /*  by bit from highest- to lowest-order term without requiring any bit   */
  /*  shuffling on our part.  Reception works similarly.                    */
  /*                                                                        */
  /*  The feedback terms table consists of 256, 32-bit entries.  Notes:     */
  /*                                                                        */
  /*      The table can be generated at runtime if desired; code to do so   */
  /*      is shown later.  It might not be obvious, but the feedback        */
  /*      terms simply represent the results of eight shift/xor opera-      */
  /*      tions for all combinations of data and CRC register values.       */
  /*                                                                        */
  /*      The values must be right-shifted by eight bits by the "updcrc"    */
  /*      logic; the shift must be unsigned (bring in zeroes).  On some     */
  /*      hardware you could probably optimize the shift in assembler by    */
  /*      using byte-swap instructions.                                     */
  /*      polynomial $edb88320                                              */
  /*                                                                        */
  /*  ------------------------------------------------------------------  */

static unsigned long crc32_tab[] = {
      0x00000000L, 0x77073096L, 0xee0e612cL, 0x990951baL, 0x076dc419L,
      0x706af48fL, 0xe963a535L, 0x9e6495a3L, 0x0edb8832L, 0x79dcb8a4L,
      0xe0d5e91eL, 0x97d2d988L, 0x09b64c2bL, 0x7eb17cbdL, 0xe7b82d07L,
      0x90bf1d91L, 0x1db71064L, 0x6ab020f2L, 0xf3b97148L, 0x84be41deL,
      0x1adad47dL, 0x6ddde4ebL, 0xf4d4b551L, 0x83d385c7L, 0x136c9856L,
      0x646ba8c0L, 0xfd62f97aL, 0x8a65c9ecL, 0x14015c4fL, 0x63066cd9L,
      0xfa0f3d63L, 0x8d080df5L, 0x3b6e20c8L, 0x4c69105eL, 0xd56041e4L,
      0xa2677172L, 0x3c03e4d1L, 0x4b04d447L, 0xd20d85fdL, 0xa50ab56bL,
      0x35b5a8faL, 0x42b2986cL, 0xdbbbc9d6L, 0xacbcf940L, 0x32d86ce3L,
      0x45df5c75L, 0xdcd60dcfL, 0xabd13d59L, 0x26d930acL, 0x51de003aL,
      0xc8d75180L, 0xbfd06116L, 0x21b4f4b5L, 0x56b3c423L, 0xcfba9599L,
      0xb8bda50fL, 0x2802b89eL, 0x5f058808L, 0xc60cd9b2L, 0xb10be924L,
      0x2f6f7c87L, 0x58684c11L, 0xc1611dabL, 0xb6662d3dL, 0x76dc4190L,
      0x01db7106L, 0x98d220bcL, 0xefd5102aL, 0x71b18589L, 0x06b6b51fL,
      0x9fbfe4a5L, 0xe8b8d433L, 0x7807c9a2L, 0x0f00f934L, 0x9609a88eL,
```

```
      0xe10e9818L, 0x7f6a0dbbL, 0x086d3d2dL, 0x91646c97L, 0xe6635c01L,
      0x6b6b51f4L, 0x1c6c6162L, 0x856530d8L, 0xf262004eL, 0x6c0695edL,
      0x1b01a57bL, 0x8208f4c1L, 0xf50fc457L, 0x65b0d9c6L, 0x12b7e950L,
      0x8bbeb8eaL, 0xfcb9887cL, 0x62dd1ddfL, 0x15da2d49L, 0x8cd37cf3L,
      0xfbd44c65L, 0x4db26158L, 0x3ab551ceL, 0xa3bc0074L, 0xd4bb30e2L,
      0x4adfa541L, 0x3dd895d7L, 0xa4d1c46dL, 0xd3d6f4fbL, 0x4369e96aL,
      0x346ed9fcL, 0xad678846L, 0xda60b8d0L, 0x44042d73L, 0x33031de5L,
      0xaa0a4c5fL, 0xdd0d7cc9L, 0x5005713cL, 0x270241aaL, 0xbe0b1010L,
      0xc90c2086L, 0x5768b525L, 0x206f85b3L, 0xb966d409L, 0xce61e49fL,
      0x5edef90eL, 0x29d9c998L, 0xb0d09822L, 0xc7d7a8b4L, 0x59b33d17L,
      0x2eb40d81L, 0xb7bd5c3bL, 0xc0ba6cadL, 0xedb88320L, 0x9abfb3b6L,
      0x03b6e20cL, 0x74b1d29aL, 0xead54739L, 0x9dd277afL, 0x04db2615L,
      0x73dc1683L, 0xe3630b12L, 0x94643b84L, 0x0d6d6a3eL, 0x7a6a5aa8L,
      0xe40ecf0bL, 0x9309ff9dL, 0x0a00ae27L, 0x7d079eb1L, 0xf00f9344L,
      0x8708a3d2L, 0x1e01f268L, 0x6906c2feL, 0xf762575dL, 0x806567cbL,
      0x196c3671L, 0x6e6b06e7L, 0xfed41b76L, 0x89d32be0L, 0x10da7a5aL,
      0x67dd4accL, 0xf9b9df6fL, 0x8ebeeff9L, 0x17b7be43L, 0x60b08ed5L,
      0xd6d6a3e8L, 0xa1d1937eL, 0x38d8c2c4L, 0x4fdff252L, 0xd1bb67f1L,
      0xa6bc5767L, 0x3fb506ddL, 0x48b2364bL, 0xd80d2bdaL, 0xaf0a1b4cL,
      0x36034af6L, 0x41047a60L, 0xdf60efc3L, 0xa867df55L, 0x316e8eefL,
      0x4669be79L, 0xcb61b38cL, 0xbc66831aL, 0x256fd2a0L, 0x5268e236L,
      0xcc0c7795L, 0xbb0b4703L, 0x220216b9L, 0x5505262fL, 0xc5ba3bbeL,
      0xb2bd0b28L, 0x2bb45a92L, 0x5cb36a04L, 0xc2d7ffa7L, 0xb5d0cf31L,
      0x2cd99e8bL, 0x5bdeae1dL, 0x9b64c2b0L, 0xec63f226L, 0x756aa39cL,
      0x026d930aL, 0x9c0906a9L, 0xeb0e363fL, 0x72076785L, 0x05005713L,
      0x95bf4a82L, 0xe2b87a14L, 0x7bb12baeL, 0x0cb61b38L, 0x92d28e9bL,
      0xe5d5be0dL, 0x7cdcefb7L, 0x0bdbdf21L, 0x86d3d2d4L, 0xf1d4e242L,
      0x68ddb3f8L, 0x1fda836eL, 0x81be16cdL, 0xf6b9265bL, 0x6fb077e1L,
      0x18b74777L, 0x88085ae6L, 0xff0f6a70L, 0x66063bcaL, 0x11010b5cL,
      0x8f659effL, 0xf862ae69L, 0x616bffd3L, 0x166ccf45L, 0xa00ae278L,
      0xd70dd2eeL, 0x4e048354L, 0x3903b3c2L, 0xa7672661L, 0xd06016f7L,
      0x4969474dL, 0x3e6e77dbL, 0xaed16a4aL, 0xd9d65adcL, 0x40df0b66L,
      0x37d83bf0L, 0xa9bcae53L, 0xdebb9ec5L, 0x47b2cf7fL, 0x30b5ffe9L,
      0xbdbdf21cL, 0xcabac28aL, 0x53b39330L, 0x24b4a3a6L, 0xbad03605L,
      0xcdd70693L, 0x54de5729L, 0x23d967bfL, 0xb3667a2eL, 0xc4614ab8L,
      0x5d681b02L, 0x2a6f2b94L, 0xb40bbe37L, 0xc30c8ea1L, 0x5a05df1bL,
      0x2d02ef8dL
   };

/* Return a 32-bit CRC of the contents of the buffer. */

unsigned long crc32(const unsigned char *s, unsigned int len)
{
  unsigned int i;
  unsigned long crc32val;

  crc32val = 0;
  for (i = 0;  i < len;  i ++)
    {
      crc32val =
       crc32_tab[(crc32val ^ s[i]) & 0xff] ^
         (crc32val >> 8);
    }
  return crc32val;
```

```
}

/*
 * Hashing function for a string
 */
unsigned int hashmap_hash_int(hashmap_map * m, char* keystring){

    unsigned long key = crc32((unsigned char*)(keystring), strlen(keystring));

        /* Robert Jenkins' 32 bit Mix Function */
        key += (key << 12);
        key ^= (key >> 22);
        key += (key << 4);
        key ^= (key >> 9);
        key += (key << 10);
        key ^= (key >> 2);
        key += (key << 7);
        key ^= (key >> 12);

        /* Knuth's Multiplicative Method */
        key = (key >> 3) * 2654435761;

        return key % m->table_size;
}

/*
 * Return the integer of the location in data
 * to store the point to the item, or MAP_FULL.
 */
int hashmap_hash(map_t in, char* key){
        int curr;
        int i;

        /* Cast the hashmap */
        hashmap_map* m = (hashmap_map *) in;

        /* If full, return immediately */
        if(m->size >= (m->table_size/2)) return MAP_FULL;

        /* Find the best index */
        curr = hashmap_hash_int(m, key);

        /* Linear probing */
        for(i = 0; i< MAX_CHAIN_LENGTH; i++){
                if(m->data[curr].in_use == 0)
                        return curr;

                if(m->data[curr].in_use == 1 && (strcmp(m->data[curr].key,key)==0))
                        return curr;

                curr = (curr + 1) % m->table_size;
        }

        return MAP_FULL;
```

```
}

/*
 * Doubles the size of the hashmap, and rehashes all the elements
 */
int hashmap_rehash(map_t in){
	int i;
	int old_size;
	hashmap_element* curr;

	/* Setup the new elements */
	hashmap_map *m = (hashmap_map *) in;
	hashmap_element* temp = (hashmap_element *)
		calloc(2 * m->table_size, sizeof(hashmap_element));
	if(!temp) return MAP_OMEM;

	/* Update the array */
	curr = m->data;
	m->data = temp;

	/* Update the size */
	old_size = m->table_size;
	m->table_size = 2 * m->table_size;
	m->size = 0;

	/* Rehash the elements */
	for(i = 0; i < old_size; i++){
	 int status;

	 if (curr[i].in_use == 0)
	     continue;

		status = hashmap_put(m, curr[i].key, curr[i].data);
		if (status != MAP_OK)
			return status;
	}

	free(curr);

	return MAP_OK;
}

/*
 * Add a pointer to the hashmap with some key
 */
int hashmap_put(map_t in, char* key, any_t value){
	int index;
	hashmap_map* m;

	/* Cast the hashmap */
	m = (hashmap_map *) in;

	/* Find a place to put our value */
	index = hashmap_hash(in, key);
```

```
        while(index == MAP_FULL){
                if (hashmap_rehash(in) == MAP_OMEM) {
                        return MAP_OMEM;
                }
                index = hashmap_hash(in, key);
        }

        /* Set the data */
        m->data[index].data = value;
        m->data[index].key = key;
        m->data[index].in_use = 1;
        m->size++;

        return MAP_OK;
}

/*
 * Get your pointer out of the hashmap with a key
 */
int hashmap_get(map_t in, char* key, any_t *arg){
        int curr;
        int i;
        hashmap_map* m;

        /* Cast the hashmap */
        m = (hashmap_map *) in;

        /* Find data location */
        curr = hashmap_hash_int(m, key);

        /* Linear probing, if necessary */
        for(i = 0; i<MAX_CHAIN_LENGTH; i++){

         int in_use = m->data[curr].in_use;
         if (in_use == 1){
             if (strcmp(m->data[curr].key,key)==0){
                 *arg = (m->data[curr].data);
                 return MAP_OK;
             }
                }

                curr = (curr + 1) % m->table_size;
        }

        *arg = NULL;

        /* Not found */
        return MAP_MISSING;
}

/*
 * Iterate the function parameter over each element in the hashmap.  The
 * additional any_t argument is passed to the function as its first
 * argument and the hashmap element is the second.
```

```
 */
int hashmap_iterate(map_t in, PFany f, any_t item) {
	int i;

	/* Cast the hashmap */
	hashmap_map* m = (hashmap_map*) in;

	/* On empty hashmap, return immediately */
	if (hashmap_length(m) <= 0)
		return MAP_MISSING;

	/* Linear probing */
	for(i = 0; i< m->table_size; i++)
		if(m->data[i].in_use != 0) {
			any_t data = (any_t) (m->data[i].data);
			int status = f(item, data);
			if (status != MAP_OK) {
				return status;
			}
		}

    return MAP_OK;
}

/*
 * Remove an element with that key from the map
 */
int hashmap_remove(map_t in, char* key){
	int i;
	int curr;
	hashmap_map* m;

	/* Cast the hashmap */
	m = (hashmap_map *) in;

	/* Find key */
	curr = hashmap_hash_int(m, key);

	/* Linear probing, if necessary */
	for(i = 0; i<MAX_CHAIN_LENGTH; i++){

	 int in_use = m->data[curr].in_use;
	 if (in_use == 1){
	     if (strcmp(m->data[curr].key,key)==0){
	         /* Blank out the fields */
	         m->data[curr].in_use = 0;
	         m->data[curr].data = NULL;
	         m->data[curr].key = NULL;

	         /* Reduce the size */
	         m->size--;
	         return MAP_OK;
	     }
		}
```

```
                curr = (curr + 1) % m->table_size;
        }

        /* Data not found */
        return MAP_MISSING;
}

/* Deallocate the hashmap */
void hashmap_free(map_t in){
        hashmap_map* m = (hashmap_map*) in;
        free(m->data);
        free(m);
}

/* Return the length of the hashmap */
int hashmap_length(map_t in){
        hashmap_map* m = (hashmap_map *) in;
        if(m != NULL) return m->size;
        else return 0;
}
```

hashmap.h

https://github.com/petewarden/c_hashmap

```
/*
 * Generic hashmap manipulation functions
 *
 * Originally by Elliot C Back -
http://elliottback.com/wp/hashmap-implementation-in-c/
 *
 * Modified by Pete Warden to fix a serious performance problem, support strings as
keys
 * and removed thread synchronization - http://petewarden.typepad.com
 */
#ifndef __HASHMAP_H__
#define __HASHMAP_H__

#define MAP_MISSING -3  /* No such element */
#define MAP_FULL -2   /* Hashmap is full */
#define MAP_OMEM -1   /* Out of Memory */
#define MAP_OK 0      /* OK */

/*
 * any_t is a pointer.  This allows you to put arbitrary structures in
 * the hashmap.
 */
typedef void *any_t;

/*
 * PFany is a pointer to a function that can take two any_t arguments
 * and return an integer. Returns status code..
 */
typedef int (*PFany)(any_t, any_t);

/*
 * map_t is a pointer to an internally maintained data structure.
 * Clients of this package do not need to know how hashmaps are
 * represented.  They see and manipulate only map_t's.
 */
typedef any_t map_t;

/*
 * Return an empty hashmap. Returns NULL if empty.
*/
extern map_t hashmap_new();

/*
 * Iteratively call f with argument (item, data) for
 * each element data in the hashmap. The function must
 * return a map status code. If it returns anything other
 * than MAP_OK the traversal is terminated. f must
 * not reenter any hashmap functions, or deadlock may arise.
 */
extern int hashmap_iterate(map_t in, PFany f, any_t item);

/*
```

```
 * Add an element to the hashmap. Return MAP_OK or MAP_OMEM.
 */
extern int hashmap_put(map_t in, char* key, any_t value);

/*
 * Get an element from the hashmap. Return MAP_OK or MAP_MISSING.
 */
extern int hashmap_get(map_t in, char* key, any_t *arg);

/*
 * Remove an element from the hashmap. Return MAP_OK or MAP_MISSING.
 */
extern int hashmap_remove(map_t in, char* key);

/*
 * Get any element. Return MAP_OK or MAP_MISSING.
 * remove - should the element be removed from the hashmap
 */
extern int hashmap_get_one(map_t in, any_t *arg, int remove);

/*
 * Free the hashmap
 */
extern void hashmap_free(map_t in);

/*
 * Get the current size of a hashmap
 */
extern int hashmap_length(map_t in);

#endif
```

Main.c

https://github.com/petewarden/c_hashmap

```
/*
 * A unit test and example of how to use the simple C hashmap
 */

#include <stdlib.h>
#include <stdio.h>
#include <assert.h>

#include "hashmap.h"

#define KEY_MAX_LENGTH (256)
#define KEY_PREFIX ("somekey")
#define KEY_COUNT (1024)

typedef struct data_struct_s
{
    char key_string[KEY_MAX_LENGTH];
    int number;
} data_struct_t;

int main(int argc, char** argv)
{
    int index;
    int error;
    map_t mymap;
    char key_string[KEY_MAX_LENGTH];
    data_struct_t* value;

    mymap = hashmap_new();

    /* First, populate the hash map with ascending values */
    for (index=0; index<KEY_COUNT; index+=1)
    {
        /* Store the key string along side the numerical value so we can free it later
*/
        value = malloc(sizeof(data_struct_t));
        snprintf(value->key_string, KEY_MAX_LENGTH, "%s%d", KEY_PREFIX, index);
        value->number = index;

        error = hashmap_put(mymap, value->key_string, value);
        assert(error==MAP_OK);
    }

    /* Now, check all of the expected values are there */
    for (index=0; index<KEY_COUNT; index+=1)
    {
        snprintf(key_string, KEY_MAX_LENGTH, "%s%d", KEY_PREFIX, index);

        error = hashmap_get(mymap, key_string, (void**)(&value));

        /* Make sure the value was both found and the correct number */
        assert(error==MAP_OK);
        assert(value->number==index);
```

```
    }

    /* Make sure that a value that wasn't in the map can't be found */
    snprintf(key_string, KEY_MAX_LENGTH, "%s%d", KEY_PREFIX, KEY_COUNT);

    error = hashmap_get(mymap, key_string, (void**)(&value));

    /* Make sure the value was not found */
    assert(error==MAP_MISSING);

    /* Free all of the values we allocated and remove them from the map */
    for (index=0; index<KEY_COUNT; index+=1)
    {
        snprintf(key_string, KEY_MAX_LENGTH, "%s%d", KEY_PREFIX, index);

        error = hashmap_get(mymap, key_string, (void**)(&value));
        assert(error==MAP_OK);

        error = hashmap_remove(mymap, key_string);
        assert(error==MAP_OK);

        free(value);
    }

    /* Now, destroy the map */
    hashmap_free(mymap);

    return 1;
}
```

Makefile (wrapper.c)

```
wrapper.o : wrapper.c
	gcc -c wrapper.c

.PHONY : clean
clean :
	rm -f *.o
```

README

https://github.com/petewarden/c_hashmap

```
the CARL PLT group for the class of Spring 2017 is adopting this simple C library for
hashmaps to use when implemeting our CARL language arrays.

This is the README from the original project:

This is a simple C hashmap, using strings for the keys.

Originally based on code by Eliot Back at
http://elliottback.com/wp/hashmap-implementation-in-c/
Reworked by Pete Warden -
http://petewarden.typepad.com/searchbrowser/2010/01/c-hashmap.html

main.c contains an example that tests the functionality of the hashmap module.
To compile it, run something like this on your system:
gcc main.c hashmap.c -o hashmaptest

There are no restrictions on how you reuse this code.
```

test.c

```
int main() {
        void *map;
        map = create();
        return 0;
}
```

test_wrapper.c

```
/* This code was written by:
Keir Lauritzen, Leon Song, and Guy Yardeni */

#include "wrapper.h"
#include <stdio.h>



int main(int argc, char** argv) {
        void *map;
        map = create();

        char *key = "testaaaaa";
        float value = 0.421;
        array_add_float(map, key, value);
```

```
        float test = array_retrieve_float(map, key);
        printf("test float : %f\n", test);

        char *key1 = "test1";
        float value1 = 0.23125123421;
        array_add_float(map, key1, value1);

        char *key2 = "test2";
        float value2 = 6341.421;
        array_add_float(map, key2, value2);

        char *key3 = "testaaaaa";
        float value3 = 0.421;
        array_add_float(map, key3, value3);

        char *key4 = "testaaaaa";
        float value4 = 9.99999;
        array_add_float(map, key4, value4);

        char *key5 = "testkeir";
        char *value5 = "testing1";
        array_add_string(map, key5, value5);

        char *key6 = "testkeir";
        char *value6 = "testing2";
        array_add_string(map, key6, value6);

        char *key7 = "testkeir";
        char *value7 = "testing3";
        array_add_string(map, key7, value7);

        char *key8 = "t";
        char *value8 = "testing4";
        array_add_string(map, key8, value8);

        destroy(map);
        return 0;
}
```

wrapper.c

```
/* This code was written by:
Keir Lauritzen, Leon Song, and Guy Yardeni */

#include <stdio.h>
#include "hashmap.h"
#include "hashmap.c"
#include "wrapper.h"
#include <string.h>
```

```
void *map_list[1024];
int list_index = 0;

double strlen2(char str[]) {
	return (double)strlen(str);
}

void* create() {
	void *map;
	map = hashmap_new();
	map_list[list_index] = map;
	list_index++;
	return map;
}

int array_remove(void *map, char *key){
	return hashmap_remove(map, key);
}

char *malloc_string(char *string) {
	int len = strlen(string) + 1;
	char *str = (char *) malloc(len * sizeof(char));
	return str;
}

void destroy(void* map) {
	int i;
	hashmap_map *m = (hashmap_map *) map;
	int size = hashmap_length(m);

	if (hashmap_length(m) <= 0) {
		hashmap_free(map);
		return;
	}


	for (i = 0; i < m->table_size; i++) {
		if(m->data[i].in_use != 0){
			void *data = (void *) m->data[i].data;
			char *key = m->data[i].key;
			free(key);
			free(data);
		}
	}
	hashmap_free(map);
	return;
}

extern void destroy_all(){
	int i;
	for(int i=0; i < list_index; i++){
		void *map = map_list[i];
		destroy(map);
	}
```

```
}

char *get_key(void *map, char *key){
	int i;
	hashmap_map *m = (hashmap_map *) map;

	if (hashmap_length(m) <= 0) {
		hashmap_free(map);
		return NULL;
	}

	for (i = 0; i < m->table_size; i++) {
		if(m->data[i].in_use != 0){
			if (strcmp(m->data[i].key,key)==0) {
				char *key = m->data[i].key;
				return key;
			}
		}
	}
	return NULL;
}

int array_add_string(void* map, char* key, char* value) {
	int check;
	char *value_ptr = NULL;
	char *key_ptr = NULL;

	check = lookup(map, key);
	if (check == 1) {
		int error;
		char *old_key = get_key(map, key);
		char *string = NULL;
		void **data = (void **) &string;
		error = hashmap_get(map, key, data);

		hashmap_remove(map, key);
		free(string);
		free(old_key);
	}

	key_ptr = malloc_string(key);
	value_ptr = malloc_string(value);
	strcpy(value_ptr, value);
	strcpy(key_ptr, key);

	return hashmap_put(map, key_ptr, (void *)value_ptr);

}

int array_add_float(void* map, char* key, double value){
	int check;
	double *value_ptr;
	char *key_ptr = NULL;
```

```
	check = lookup(map, key);
	if (check == 1) {
		int error;
		char *old_key = get_key(map, key);
		double *float_ptr = NULL;
		void **data = (void **) &float_ptr;
		error = hashmap_get(map, key, data);

		hashmap_remove(map, key);
		free(float_ptr);
		free(old_key);
	}

	key_ptr = malloc_string(key); //only malloc if its a new key
	strcpy(key_ptr, key);
	value_ptr = (double *) malloc(sizeof(double));
	*value_ptr = value;


	return hashmap_put(map, key_ptr, (void *)value_ptr);
}

double array_retrieve_float(void *map, char *key){
	int error;
	double *float_ptr = NULL;
	void **data = (void **) &float_ptr;
	error = hashmap_get(map, key, data);
	return *float_ptr;
}

char *array_retrieve_string(void* map, char* key){
	int error;
	char *string = NULL;
	void **data = (void **) &string;
	error = hashmap_get(map, key, data);
	return string;
}

int lookup(void* map, char* key) {
	int error;
	void *lookup = NULL;
	void **data = (void **) &lookup;
	error = hashmap_get(map, key, data);
	if (error == MAP_OK){
		return 1;
	}
	return 0;
}
```

wrapper.h

```
/* This code was written by:
Keir Lauritzen, Leon Song, and Guy Yardeni */

#ifndef __WRAPPER_H__
#define __WRAPPER_H__

extern void* create();

char *malloc_string(char *string);

extern void destroy(void* map);

extern int array_add_string(void *map, char *key, char* value);

extern int array_add_float(void* map, char* key, double value);

extern int lookup(void* map, char* key);

extern int array_remove(void* map, char* key);

extern char* array_retrieve_string(void* map, char* key);

extern double array_retrieve_float(void *map, char* key);

extern void destroy_all();

#endif
```

buildcarl

```
#!/bin/bash
# This code was written by Keir Lauritzen


CARL_DIR="./carl"
CARL_ENGINE_DIR="./carl_engine"
ARRAY_LIB_DIR="./array_lib"

rm -f carlc carl_engine.o wrapper.o

cd $CARL_DIR
make clean
make
mv -f "carlc" ../
make clean
cd ..

cd $CARL_ENGINE_DIR
make clean
make
mv -f "carl_engine.o" ../
make clean
cd ..

cd $ARRAY_LIB_DIR
make clean
make
mv -f "wrapper.o" ../
make clean
cd ..

#echo "Carl compiler ready"
```

buildcarlp

```
#!/bin/bash

CARLC="./carlc"
CARL_ENGINE="carl_engine.o"
ARRAY_LIB="wrapper.o"
TMP_DIR="./tmp"
TEST_DIR="./tests"

if [ ! -d ./tmp ];
then
    mkdir ./tmp
fi

if [ ! -f $CARLC ] || [ ! -f $CARL_ENGINE ];
then
    ./buildcarl
fi

echo "#######  Testing $1";
$CARLC < $TEST_DIR/$1.carl > $TMP_DIR/$1.ll;
if [ -s $TMP_DIR/$1.ll ]
then
       llc $TMP_DIR/$1.ll #Converts LLVM to assembly .s file
       gcc -c $TMP_DIR/$1.s -o $TMP_DIR/$1.o

       g++ -o $TMP_DIR/$1.test $ARRAY_LIB $CARL_ENGINE $TMP_DIR/$1.o
else
       rm $TMP_DIR/$1.ll
fi
```

buildcarlc

```
#!/bin/bash
# This code was written by Darren Hakimi and Keir Lauritzen

CARLC="./carlc"
CARL_ENGINE="carl_engine.o"
ARRAY_LIB="wrapper.o"
TMP_STALE="./stale"
TESTDIR="./tests"

if [ ! -d ./tmp ];
then
    mkdir ./tmp
fi

if [ ! -f $CARLC ] || [ ! -f $CARL_ENGINE ];
then
    ./buildcarl
fi

echo "#######  Testing $1";
clang -S -emit-llvm $1.c
llc $1.ll #Converts LLVM to assembly .s file
gcc -c $1.s -o $1.o

g++ -o $1.test $CARL_ENGINE $ARRAY_LIB $1.o
```

testall.sh

```
#!/bin/bash
# This code was written by Darren Hakimi and Keir Lauritzen

CARLC="./carlc"
CARL_ENGINE="carl_engine.o"
TMP_DIR="./tmp"
TEST_DIR="./tests"
files=`ls $TEST_DIR/test-*.carl`
PASS_COUNTER=0
FAIL_COUNTER=0

rm -rf ./tmp

if [ ! -d ./tmp ];
then
    mkdir ./tmp
fi
```

```
if [ ! -f $CARLC ] || [ ! -f $CARL_ENGINE ];
then
    ./buildcarl
fi

for file in $files
do
    basename=`basename $file`;
    basename=`echo $basename | sed 's/.carl$//'`
    ./buildcarlp $basename;

    if [ -f ./$TMP_DIR/$basename.test ];
       then
       ./$TMP_DIR/$basename.test $TEST_DIR/$basename.txt >
$TMP_DIR/$basename.generated;
       if diff -b $TEST_DIR/$basename.out $TMP_DIR/$basename.generated >
$TMP_DIR/$basename.diff;
       then
            let PASS_COUNTER=PASS_COUNTER+1
              echo "PASS";
       else
            let FAIL_COUNTER=FAIL_COUNTER+1
              echo "FAIL";
       fi
       else
        let FAIL_COUNTER=FAIL_COUNTER+1
        echo "FAIL";
    fi
    echo;

done

echo;
echo "number of passed tests:";
echo $PASS_COUNTER;
echo "number of failed tests:";
echo $FAIL_COUNTER;
echo;

files=`ls $TEST_DIR/fail-*.carl`
PASS_COUNTER=0
FAIL_COUNTER=0

for file in $files
do
    basename=`basename $file`;
    basename=`echo $basename | sed 's/.carl$//'`
    ./buildcarlp $basename;

    if [ -f ./$TMP_DIR/$basename.test ];
    then
        ./$TMP_DIR/$basename.test $TEST_DIR/$basename.txt >
$TMP_DIR/$basename.generated;
```

```
        if diff -b $TEST_DIR/$basename.out $TMP_DIR/$basename.generated >
$TMP_DIR/$basename.diff;
        then
            let FAIL_COUNTER=FAIL_COUNTER+1
            echo "FAIL";
        else
            let PASS_COUNTER=PASS_COUNTER+1
            echo "PASS";
        fi
    else
        let PASS_COUNTER=PASS_COUNTER+1
        echo "PASS";
    fi
    echo;

done

echo;
echo "number of passed FAIL tests:";
echo $PASS_COUNTER;
echo "number of failed FAIL tests:";
echo $FAIL_COUNTER;
echo;
```

fail-array_bad_assign.carl

```
/* This code was written by Darren Hakimi */

BEGIN {
       array_string x;
       x = 2.0;
}
```

fail-array_bad_assign.err

```
Fatal error: exception Failure("bad assignment")
```

fail-array_bad_assign.txt

fail-array_redeclaration.carl

```
/* This code was written by Darren Hakimi */

BEGIN
{
	array_string my_array;
	my_array = [1 : “P”, 2 : “A”, 3 : “S”, 4 : “S”];

	array_string my_array;
	my_array = [1 : “F”, 2 : “A”, 3 : “I”, 4 : “L”];
}
END {}
```

fail-array_redeclaration.err

```
Fatal error: exception Failure(”re-delaration of same identifier”)
```

fail-array_redeclaration.txt

fail-empty.carl

```
/* This code was written by Darren Hakimi */
```

fail-empty.err

```
Fatal error: exception Failure(”empty file”)
```

fail-empty.txt

fail-end_before_begin.carl

```
/* This code was written by Darren Hakimi */

END{}

BEGIN{}
```

fail-end_before_begin.err

```
Fatal error: exception Failure("end before begin")
```

fail-end_before_begin.txt

fail-float_bad_assign.carl

```
/* This code was written by Darren Hakimi */

BEGIN {
	float x;
	x = "fail";
}
```

fail-float_bad_assign.err

```
Fatal error: exception Failure("bad assignment")
```

fail-float_bad_assign.txt

fail-float_redeclaration.carl

```
/* This code was written by Darren Hakimi */

BEGIN {
	float x;
	x = 1.0;
	float x;
	x = 2.0;
}
```

fail-float_redeclaration.err

```
Fatal error: exception Failure("re-delaration of same identifier")
```

fail-float_redeclaration.txt

fail-int_bad_assign.carl

```
/* This code was written by Darren Hakimi */

BEGIN {
	int x;
	x = "fail";
}
```

fail-int_bad_assign.err

```
Fatal error: exception Failure("bad assignment")
```

fail-int_bad_assign.txt

fail-int_redeclaration.carl

```
/* This code was written by Darren Hakimi */

BEGIN {
	int x;
	x = 1;
	int x;
	x = 2;
}
```

fail-int_redeclaration.err

```
Fatal error: exception Failure("re-delaration of same identifier")
```

fail-int_redeclaration.txt

fail-only_end.carl

```
/* This code was written by Darren Hakimi */

END{}
```

fail-only_end.err

```
Fatal error: exception Failure("only end, no begin")
```

fail-only_end.txt

fail-regex_abc.carl

```
/* This code was written by Darren Hakimi */

BEGIN {}

//abc//  {     print_string("found an instance of abc"); }

END{}
```

fail-regex_abc.out

```
found an instance of abc
```

fail-regex_abc.txt

```
09342
```

fail-regex_one_or_more.carl

```
/* This code was written by Darren Hakimi */

BEGIN {}

//a+//  {      print_string("found atleast one instance of the letter a"); }

END{}
```

fail-regex_one_or_more.out

```
found atleast one instance of the letter a
```

fail-regex_one_or_more.txt

```
b
```

fail-regex_range_lowercase_letter.carl

```
/* This code was written by Darren Hakimi */
BEGIN {}
//[a-z]//  {   print_string("found one instance of a lowercase letter"); }
END{}
```

fail-regex_range_lowercase_letter.out

```
found one instance of a lowercase letter
```

fail-regex_range_lowercase_letter.txt

```
A
```

fail-regex_range_number.carl

```
/* This code was written by Darren Hakimi */
BEGIN {}
//[0-9]//  {   print_string("found one instance of a single digit number"); }
END{}
```

fail-regex_range_number.out

```
found one instance of a single digit number
```

fail-regex_range_number.txt

```
fkd
```

fail-regex_range_uppercase_letter.carl

```
/* This code was written by Darren Hakimi */

BEGIN {}

//[A-Z]//  {   print_string("found one instance of an uppercase letter"); }

END{}
```

fail-regex_range_uppercase_letter.out

```
found one instance of an uppercase letter
```

fail-regex_range_uppercase_letter.txt

```
a
```

fail-string_bad_assign.carl

```
/* This code was written by Darren Hakimi */

BEGIN {
       string x;
       x = 2.0;
```

```
}
```

fail-string_bad_assign.err

```
Fatal error: exception Failure("bad assignment")
```

fail-string_bad_assign.txt

fail-string_redeclaration.carl

```
/* This code was written by Darren Hakimi */

BEGIN {
	string x;
	x = "pass";
	string x;
	x = "fail";
}
```

fail-string_redeclaration.err

```
Fatal error: exception Failure("re-delaration of same identifier")
```

fail-string_redeclaration.txt

hello_world.carl

```
/* This code was written by Keir Lauritzen */

BEGIN { print_string("hello world!"); }
```

hello_world.txt

test-array_add_float.carl

```
/* This code was written by Guy Yardeni */

BEGIN {
   array_string myArray;
   string key;
   float value;

   key = "first-key";
   value = 42.14159;
   myArray = create();
   array_add_float(myArray, key, value);
}
```

test-array_add_float.out

test-array_add_float.txt

test-array_add_string.carl

```
/* This code was written by Guy Yardeni */

BEGIN {
   array_string myArray;
   string key;
   string value;
   key = "first-key";
   value = "first-value";
```

```
   myArray = create();
   array_add_string(myArray, key, value);
}
```

test-array_add_string.out

test-array_add_string.txt

test-array_create.carl

```
/* This code was written by Guy Yardeni */

BEGIN {
    array_float myArrayFloat;
    myArrayFloat = create();
}
```

test-array_create.out

test-array_create.txt

test-array_retrieve_float.carl

```
/* This code was written by Guy Yardeni */

BEGIN {
   array_string myArray;
```

```
   string key;
   float value;
   float new;

   key = "first-key";
   value = 987.123456;
   new = 1.1423;

   myArray = create();

   array_add_float(myArray, key, value);
   new = array_retrieve_float(myArray, key);

   print_float(new);
}
```

test-array_retrieve_float.out

```
987.123456
```

test-array_retrieve_float.txt

test-array_retrieve_floats.carl

```
/* This code was written by Guy Yardeni */

BEGIN {
       array_string myArray;

       string key;
       float value;
       float new;

       string key2;
       float value2;
       float new2;

       string key3;
       float value3;
       float new3;

       string key4;
```

```
        float value4;
        float new4;

        key = "first-key";
        value = 1;
        new = 0.12657;

        key2 = "second-key";
        value2 = 23.53264;
        new2 = -31.312;

        key3 = "third-key";
        value3 = -1;
        new3 = -31.312;

        key4 = "fourth-key";
        value4 = -123.53264;
        new4 = -31.312;


        myArray = create();

        array_add_float(myArray, key, value);
        array_add_float(myArray, key2, value2);
        array_add_float(myArray, key3, value3);
        array_add_float(myArray, key4, value4);

        new = array_retrieve_float(myArray, key);
        new2 = array_retrieve_float(myArray, key2);
        new3 = array_retrieve_float(myArray, key3);
        new4 = array_retrieve_float(myArray, key4);


        print_float(new);
        print_float(new2);
        print_float(new3);
        print_float(new4);

}
```

test-array_retrieve_floats.out

```
1.000000
23.532640
-1.000000
-123.532640
```

test-array_retrieve_floats.txt

test-array_retrieve_string.carl

```
/* This code was written by Guy Yardeni */

BEGIN {
   array_string myArray;
   string key;
   string value;
   string newString;
   key = "first-key";
   value = "first-value";
   newString = "";
   myArray = create();
   array_add_string(myArray, key, value);
   newString = array_retrieve_string(myArray, key);

   print_string(newString);
}
```

test-array_retrieve_string.out

```
first-value
```

test-array_retrieve_string.txt

test-array_retrieve_strings.carl

```
/* This code was written by Guy Yardeni */

BEGIN {
   array_string myArray;
   string key;
```

```
    string value;
    string newString;

    string key2;
    string value2;
    string newString2;

    string key3;
    string value3;
    string newString3;

    string key4;
    string value4;
    string newString4;

    key = "first-key";
    value = "first-value";
    newString = "";

    key2 = "second-key";
    value2 = "second-value";
    newString2 = "";

    key3 = "third-key";
    value3 = "third-value";
    newString3 = "";

    key4 = "fourth-key";
    value4 = "fourth-value";
    newString4 = "";

    myArray = create();
    array_add_string(myArray, key, value);
    array_add_string(myArray, key2, value2);
    array_add_string(myArray, key3, value3);
    array_add_string(myArray, key4, value4);
    newString = array_retrieve_string(myArray, key);
    newString2 = array_retrieve_string(myArray, key2);
    newString3 = array_retrieve_string(myArray, key3);
    newString4 = array_retrieve_string(myArray, key4);
    print_string(newString);
    print_string(newString2);
    print_string(newString3);
    print_string(newString4);
}
```

test-array_retrieve_strings.out

```
first-value
second-value
```

```
third-value
fourth-value
```

test-array_retrieve_strings.txt

test-array_scope.carl

```
/* This code was written by Guy Yardeni */

BEGIN {
	array_string myArray;
	array_string myArray2;

	string key;
	float value;

	string key2;
	float value2;

	string test;

	key = "first-key";
	value = 1;
	new = 0.12657;

	key2 = "second-key";
	value2 = 23.53264;
	new2 = -31.312;

	myArray = create();
	myArray2 = create();

	array_add_float(myArray, key, value);
	array_add_float(myArray2, key2, value2);

}

END {
	float new;
	float new2;

	new = array_retrieve_float(myArray, key);
	new2 = array_retrieve_float(myArray2, key2);

	print_float(new);
```

```
        print_float(new2);

}
```

test-array_scope.out

```
1.000000
23.532640
```

test-array_scope.txt

test-array_scope_modification.carl

```
/* This code was written by Guy Yardeni */

BEGIN {
        array_string myArray;

        string key;
        float value;

        key = "first-key";
        value = 1;
        new = 0.12657;


        myArray = create();

        array_add_float(myArray, key, value);

}

END {

        string key2;
        float value2;

        float new;
        float new2;

        key2 = "second-key";
        value2 = 23.53264;
```

```
        new2 = -31.312;

        array_add_float(myArray, key2, value2);


        new = array_retrieve_float(myArray, key);
        new2 = array_retrieve_float(myArray, key2);

        print_float(new);
        print_float(new2);

}
```

test-array_scope_modification.out

```
1.000000
23.532640
```

test-array_scope_modification.txt

test-array_string_overwrite.carl

```
/* This code was written by Guy Yardeni */

BEGIN {
   array_string myArray;
   string key;
   string value;
   string newString;

   string key2;
   string value2;
   string newString2;

   string key3;
   string value3;
   string newString3;

   string key4;
   string value4;
   string newString4;
```

```
    key = "first-key";
    value = "first-value";
    newString = "";

    key2 = "second-key";
    value2 = "second-value";
    newString2 = "";

    key3 = "third-key";
    value3 = "third-value";
    newString3 = "";

    key4 = "fourth-key";
    value4 = "fourth-value";
    newString4 = "";

    myArray = create();
    array_add_string(myArray, key, value);
    newString = array_retrieve_string(myArray, key);
    print_string(newString);

    array_add_string(myArray, key2, value2);
    newString2 = array_retrieve_string(myArray, key2);
    print_string(newString2);

    array_add_string(myArray, key, value3);
    newString3 = array_retrieve_string(myArray, key);
    print_string(newString3);

    array_add_string(myArray, key2, value4);
    newString4 = array_retrieve_string(myArray, key2);
    print_string(newString4);
}
```

test-array_string_overwrite.out

```
first-value
second-value
third-value
fourth-value
```

test-array_string_overwrite.txt

test-array_syntax_float.carl

```
/* This code was written by Guy Yardeni */

BEGIN {
	array_float myArray;
	string key;
	float value;
	float temp;

	myArray = [];
	key = "key";
	value = 44.0;
	myArray[key] = value;

	temp = myArray[key];
	print_float(temp);

}
```

test-array_syntax_float.out

```
44.000000
```

test-array_syntax_float.txt

test-array_syntax_string.carl

```
/* This code was written by Guy Yardeni */

BEGIN {
        array_string myArray;
        string key;
        string value;
        string temp;

        myArray = [];
        key = "key";
        value = "Hello World!";
        myArray[key] = value;

        temp = myArray[key];
        print_string(temp);

}
```

test-array_syntax_string.out

```
Hello World!
```

test-array_syntax_string.txt

test-empty_begin.carl

```
/* This code was written by Darren Hakimi */

BEGIN {

}
```

test-empty_begin.out

test-empty_begin.txt

test-empty_begin_empty_end.carl

```
/* This code was written by Darren Hakimi */

BEGIN {}
END {}
```

test-empty_begin_empty_end.out

test-empty_begin_empty_end.txt

test-equality_float.carl

```
/* This code was written by Darren Hakimi */

BEGIN {
	float a;
	float b;

	a = 3.14;
	b = 3.14;

	if (a == b) {
		print_string("hello world");
	}

}
```

test-equality_float.out

```
hello world
```

test-equality_float.txt

test-float1.carl

```
/* This code was written by Darren Hakimi */

BEGIN
{
	float f = 1.0;

	print_float(f);
}
END {}
```

test-float1.out

```
1.000000
```

test-float1.txt

test-float1_init.carl

```
/* This code was written by Darren Hakimi */

BEGIN
{
```

```
        float f;
        f = 1.0;

        print_float(f);
}
END {}
```

test-float1_init.out

```
1.000000
```

test-float1_init.txt

test-float_addition.carl

```
/* This code was written by Darren Hakimi */

BEGIN
{
        float a = 1.0;
        float b = 2.0;
        float c = a + b;

        print_float(c);
}
END {}
```

test-float_addition.out

```
3.000000
```

test-float_addition.txt

test-float_addition_init.carl

```
/* This code was written by Darren Hakimi */

BEGIN
{
	float a;
	float b;
	float c;
	a = 1.0;
	b = 2.0;
	c = a + b;

	print_float(c);
}
END {}
```

test-float_addition_init.out

```
3.000000
```

test-float_addition_init.txt

test-float_division.carl

```
/* This code was written by Darren Hakimi */

BEGIN
{
	float a = 1.0;
	float b = 2.0;
	float c = a / b;

	print_float(c);
}
END {}
```

test-float_division.out

```
0.500000
```

test-float_division.txt

test-float_division_init.carl

```
/* This code was written by Darren Hakimi */

BEGIN
{
	float a;
	float b;
	float c;
	a = 1.0;
	b = 2.0;
	c = a / b;

	print_float(c);
}
END {}
```

test-float_division_init.out

```
0.500000
```

test-float_division_init.txt

test-float_multiplication.carl

```
/* This code was written by Darren Hakimi */

BEGIN
{
	float a = 1.0;
	float b = 2.0;
	float c = a * b;

	print_float(c);
}
END {}
```

test-float_multiplication.out

```
2.000000
```

test-float_multiplication.txt

test-float_multiplication_init.carl

```
/* This code was written by Darren Hakimi */

BEGIN
{
	float a;
	float b;
	float c;
	a = 1.0;
	b = 2.0;
	c = a * b;

	print_float(c);
}
END {}
```

test-float_multiplication_init.out

```
2.000000
```

test-float_multiplication_init.txt

test-float_subtraction.carl

```
/* This code was written by Darren Hakimi */

BEGIN
{
	float a = 1.0;
	float b = 2.0;
	float c = a - b;

	print_float(c);
}
END {}
```

test-float_subtraction.out

```
-1.000000
```

test-float_subtraction.txt

test-float_subtraction_init.carl

```
/* This code was written by Darren Hakimi */

BEGIN
{
	float a;
	float b;
	float c;
	a = -1.0;
	b = 2.0;
```

```
        c = a - b;

        print_float(c);
}
END {}
```

test-float_subtraction_init.out

```
-3.000000
```

test-float_subtraction_init.txt

test-function_float.carl

```
/* This code was written by Darren Hakimi */

function float my_func() {
        return 5.444000;
}

BEGIN {
        float number;
        number = my_func();
        print_float(number);
}
```

test-function_float.out

```
5.444000
```

test-function_float.txt

test-function_string.carl

```
/* This code was written by Guy Yardeni */

function string my_func() {
	return "function string works";
}

BEGIN {
	string my_string;
	my_string = my_func();
	print_string(my_string);
}
```

test-function_string.out

```
function string works
```

test-function_string.txt

test-function_void.carl

```
/* This code was written by Guy Yardeni */

function void my_func() {
	print_string("void func");
```

```
}

BEGIN {
        my_func();
        print_string("after func");
}
```

test-function_void.out

```
void func
after func
```

test-function_void.txt

test-hello.carl

```
/* This code was written by Darren Hakimi */

BEGIN
{
        print_string("Hello World!");
}
END {}
```

test-hello.out

```
Hello World!
```

test-hello.txt

test-if.carl

```
/* This code was written by Darren Hakimi */

BEGIN
{
        if (5>4)
        {
                print_string("if works");
        }
}
END {}
```

test-if.out

```
if works
```

test-if.txt

test-if_else.carl

```
/* This code was written by Darren Hakimi */

BEGIN
{
	if (0)
	{
		print_string("false1");
	}
	else if (0)
	{
		print_string("false2");
	}
	else
	{
		print_string("else works");
	}
}
END {}
```

test-if_else.out

```
else works
```

test-if_else.txt

test-if_elseif.carl

```
/* This code was written by Darren Hakimi */

BEGIN
{
	if (0)
	{
		print_string("false");
	}
	else if (1)
	{
		print_string("else if works");
```

```
        }
}
END {}
```

test-if_elseif.out

```
else if works
```

test-if_elseif.txt

test-if_float.carl

```
/* This code was written by Darren Hakimi */

BEGIN
{
        if (5.0)
        {
                print_string("if works");
        }

        if (0)
        {
                print_string("if doesn't work");
        }
}
END {}
```

test-if_float.out

```
if works
```

test-if_float.txt

test-if_string.carl

```
/* This code was written by Darren Hakimi */

BEGIN
{
	if ("abc")
	{
		print_string("if works");
	}

	if ("")
	{
		print_string("if doesn't work");
	}
}
END {}
```

test-if_string.out

```
if works
```

test-if_string.txt

test-if_string_var.carl

```
/* This code was written by Darren Hakimi */

BEGIN
{
    string a;
	a = "CARL";

	if (a)
	{
		print_string("if works");
	}
```

```
        a = "";
        if (a)
        {
                print_string("if doesn't work");
        }
}
END {}
```

test-if_string_var.out

```
if works
```

test-if_string_var.txt

test-loop_for.carl

```
/* This code was written by Darren Hakimi */

BEGIN
{
        float i;
        for (i = 0; i < 3; i = i + 1)
        {
                print_float(i);
        }
}
END {}
```

test-loop_for.out

```
0.000000
1.000000
2.000000
```

test-loop_for.txt

test-loop_while.carl

```
/* This code was written by Darren Hakimi */

BEGIN
{
	float i;
	i = 0;
	while(i < 3)
	{
		print_float(i);
		i = i + 1;
	}
}
END {}
```

test-loop_while.out

```
0.000000
1.000000
2.000000
```

test-loop_while.txt

test-program1.carl

```
/* This code was written by Guy Yardeni and Leon Song */
```

```
function float slight_increase(float val) {
	return val * 1.1;
}

BEGIN {
	float float_val1 = 4.321;
	string string_val1 = "I'm just a string.";
}

//float//  {
	while (float_val1) {
		print_float(float_val1);
		float_val1 = float_val1 - 1;
		float_val1 = slight_increase(float_val1);
	}
}
```

```
//string// {
	if (string_val1) {
		print_string(string_val1);
		while (string_val1) {
			print_string(string_val1);
			string_val1 = "";
		}
	}
}

END {

	if (float_val1 > 4.0) {
		print_string("strings are the best");
	} else {
		print_string("floats are the best");
	}
}
```

test-program1.out

```
strings are the best
```

test-program1.txt

test-program1_float.txt

```
float
```

test-program1_string.txt

```
string
```

test-program2.carl

```
/* This code was written by Guy Yardeni and Leon Song */

BEGIN {
	float temp_float;
	array_float myArrayFloat1 = [];
	array_float myArrayFloat2 = [];
	array_string myArrayString1 = [];
	string str = "key1";
	float val = 1;
	myArrayString1["abc"] = "2";
	myArrayString1["def"] = "MEANING OF LIFE?";
	myArrayFloat1[str] = val;
	myArrayFloat2[str] = myArrayFloat1[str] * 42;
}

END {
	string temp_string;
	temp_string = myArrayString1["abc"];
	print_string(temp_string);
	temp_string = myArrayString1["def"];
	print_string(temp_string);

	temp_float = myArrayFloat2[str];
	print_float(temp_float);
}
```

test-program2.out

```
2
MEANING OF LIFE?
42.000000
```

test-program2.txt

test-program3.carl

```
/* This code was written by Guy Yardeni and Leon Song */

BEGIN {
	float a = 0;
	float b = 0;
```

```
        float c = 0;
        float d = 0;
        float e = 0;
        array_float myArray = [];
        string hiker = "key";
}

//Hitchhiker|Hitch Hiker// {
        a = a+1;
        myArray["Hitchhiker"] = a;
}

//Guide// {
        b = b+1;
        myArray["Guide"] = b;
}

//Galaxy// {
        c = c+1;
        myArray["Galaxy"] = c;
}

//Hitchhiker|Guide|Galaxy|Hitch Hiker//{
        d = d+1;
        myArray["Any"] = d;
}

//a*// {
        e = e+1;
        myArray["total"] = e;
}

END {
        print_string("Hitchhiker:");
        print_float(myArray["Hitchhiker"]);

        print_string("Guide:");
        print_float(myArray["Guide"]);

        print_string("Galaxy:");
        print_float(myArray["Galaxy"]);

        print_string("Any:");
        print_float(myArray["Any"]);

        print_string("total:");
        print_float(myArray["total"]);
}
```

test-program3.out

```
Hitchhiker:
16.000000
Guide:
23.000000
Galaxy:
58.000000
Any:
63.000000
total:
7373.000000
```

test-program3.txt

```
The Hitch Hiker’s Guide to the Galaxy

for Jonny Brock and Clare Gorst
and all other Arlingtoniansfor tea, sympathy, and a sofa



Far out in the uncharted backwaters of the unfashionable  end  of
the  western  spiral  arm  of  the Galaxy lies a small unregarded
yellow sun.

Orbiting this at a distance of roughly ninety-two  million  miles
is  an  utterly insignificant little blue green planet whose ape-
descended life forms are so amazingly primitive that  they  still
think digital watches are a pretty neat idea.
…
```

test-regex_abc.carl

```
/* This code was written by Darren Hakimi */

BEGIN {}

//abc//  {     print_string("found an instance of abc"); }

END{}
```

test-regex_abc.out

```
found an instance of abc
```

test-regex_abc.txt

```
abc
```

test-regex_one_or_more.carl

```
/* This code was written by Darren Hakimi */

BEGIN {}

//a+//  {      print_string("found atleast one instance of the letter a"); }

END{}
```

test-regex_one_or_more.out

```
found atleast one instance of the letter a
```

test-regex_one_or_more.txt

```
a
```

test-regex_range_lowercase_letter.carl

```
/* This code was written by Darren Hakimi */

BEGIN {}

//[a-z]//  {   print_string("found one instance of a lowercase letter"); }

END{}
```

test-regex_range_lowercase_letter.out

```
found one instance of a lowercase letter
```

test-regex_range_lowercase_letter.txt

```
a
```

test-regex_range_number.carl

```
/* This code was written by Darren Hakimi */
BEGIN {}
//[0-9]//  {   print_string("found one instance of a single digit number"); }
END{}
```

test-regex_range_number.out

```
found one instance of a single digit number
```

test-regex_range_number.txt

```
9
```

test-regex_range_number_abc_multi.carl

```
/* This code was written by Darren Hakimi */
BEGIN {}
//[0-9]//  {   print_string("found one instance of a single digit number"); }
//abc//  {     print_string("found an instance of abc"); }
END{}
```

test-regex_range_number_abc_multi.out

```
found one instance of a single digit number
found one instance of a single digit number
found an instance of abc
```

test-regex_range_number_abc_multi.txt

```
3

abc

2
```

test-regex_range_number_multi.carl

```
/* This code was written by Darren Hakimi */

BEGIN {}

//[0-9]//  {   print_string("found one instance of a single digit number"); }

END{}
```

test-regex_range_number_multi.out

```
found one instance of a single digit number
found one instance of a single digit number
```

test-regex_range_number_multi.txt

```
3
0
```

test-regex_range_uppercase_letter.carl

```
/* This code was written by Darren Hakimi */

BEGIN {}

//[A-Z]//  {   print_string("found one instance of an uppercase letter"); }

END{}
```

test-regex_range_uppercase_letter.out

```
found one instance of an uppercase letter
```

test-regex_range_uppercase_letter.txt

```
A
```

test-string1.carl

```
/* This code was written by Darren Hakimi */

BEGIN
{
        string s;
        s = "CARL";

        print_string(s);
}
END {}
```

test-string1.out

```
CARL
```

test-string1.txt

test-string_scope.carl

```
BEGIN {
        string test;

        test = "hello";

}

END {

        print_string(test);
}
```

test-string_scope.out

```
hello
```

test-string_scope.txt