C+ Final Report

By Alexander Stein & Eric Johnson

May 10th 2017

Columbia University
Dr. Stephen A. Edwards
COMS W4115 S2017 – Programming Languages & Translators

Introduction

The goal of this project was originally to build a high-performance implementation of a subset of C for the purpose of creating an in memory graph database and graph analytics package. Using a subset of the ANSI C standard for the we intend to extend C to support bother RDF Graph and Property Graph structures, something we have coined the name of C+ Graph. In acknowledging that compiler design is an extraordinarily complex subject we acknowledge we will need to scale back the implementation back quite a bit since two people cannot cover all of C and the various graph structures and analytics we intend to create. The problem we reduced to implementing the basic types and operators of C, and to implementing pointers and structures. This was to enable us to effectively design C+ versions of our graph algorithms. The choice of these features bore in mind the sample programs we hoped to demonstrate: data structures, algorithms, and design / output for graph structures and graph analytics.

We developed a robust shell application which achieves interesting graphing applications in the C language. Unfortunately, we severely underestimated the scope of our undertaking, and were not able to use C+ to make these same graph representations and algorithms as we had hoped. The C+ language is still functional however, and does possess working pointers, structures, and arrays -- which allow for some interesting programs.

C+ Language Reference Manual

Identifiers

In CPlus, identifiers are combinations of characters, numbers, and the special character '_'; they must begin with a letter, and they are case-sensitive. There are two types of identifiers:

(1) Value and Function Identifiers: these must begin with a lower-case letter; they represent variable values of all datatypes as well as function names. Defined formally as
`['a'-'z']['a'-'z' 'A'-'Z' '0'-'9' '_']*`

(2) Struct Identifiers: these must begin with a capital letter; they represent globally-defined structure types, used to group several datatypes into a single record. Defined formally
`['A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']*`

Keywords

The reserved keywords in CPlus include types, control flow indicators, and built-in functions:

`if`	`else`	`for`	`while`	`return`
`int`	`char`	`size_t`	`string`	`char`
`bool`	`void`	`true`	`false`	`sizeof`
`struct`	`NULL`	`printf`	`atoi`	`strdup`
`printb`	`print`	`printbig`	`malloc`	`free`

Literals

Literals are the primitive values which represent the core data of any language, in this language there are only five

1. Integer Literals: Any combination of one or more numerals - `['0'-'9']+`
2. Character Literals: Any ASCII Character, including all specials - `['\x00'-'\x7F']`
3. Boolean Literals : Either of `true` or `false`.
4. String Literals: Any combination of characters found between "" marks; with supported escape sequences:
   - \ - escape character
   - \b – backspace character
   - \n – newline character
   - \f – form feed
   - \r – carriage return
   - \t – tab character

5 Separators: Used to help build the abstract syntax tree and determine semantics, discarded after parsing; in this language we use:

‘(‘ – LPAREN
‘)’ – RPAREN
‘{‘ – LBRACE
‘}’ – RBRACE
‘[‘ – LSQUARE
‘]’ – RSQUARE
‘;’ – SEMI
‘,’ – COMMA
‘/*’ – open comment
‘*/’ – close comment

Operators

There are three categories of operators in CPlus, which dictate all the mathematical, memory-access, and other logical functionality of the language:

(1) Binary Operators : these operate on two expressions, and the order of which expression is on the left or right is significant

+	addition of two signed 32-bit integers – returns 32-bit int
-	subtraction of two signed 32-bit integers – returns 32-bit int
*	multiplication of two signed 32-bit integers – returns 32-bit int
/	division of two signed 32-bit integers – returns 32-bit int
%	modulus of two signed 32-bit integers – retruns 32-bit int
==	physical equality of two primitive literals of same type – returns bool
!=	physical inequality of two primitive literals of same type – returns bool
<	less than of two 32-bit integers – returns bool
<=	less-than-equal-to of two 32-bit integers – returns bool
>	greater than of two 32-bit integers – returns bool
>=	greater-than-equal-to of two 32-bit integers – returns bool
&&	logical AND of two bools – returns bool
||	logical OR of two bools – returns bool

(2) Unary Operators : these operate on a single expression, and significance is placed on whether the operator comes before (prefix) or after (postfix) the expression

-	Applied before integer, makes it negative (cannot be chained together)
!	Applied after bool, inverts its value
++	Applied before or after integer, increases its value by one
--	Applied before or after integer, decrements its value by one
*	Applied before an identifier, retrieves its value from memory
&	Applied before an identifier, returns its memory address
->	Applied after struct identifier, retrieve struct from mem and get member
.	Applied after struct identifier, get member

(3) Assignment Operators : these operate an two expressions, the lefthand of which must evaluate to an *lvalue* and the righthand of which must evaluate to an *rvalue.* We have only implemented two for our simple purposes

= store *rvalue* at the address *lvalue*

%= *lvalue* must point to an integer, performs 'm[lvalue] = m[lvalue] % rvalue'

Operator Precedence & Associativity

In decreasing order from top to bottom:

OPERATORS	ASSOCIATIVITY
*** & SIZEOF ()**	Right-to-left
() [] . -> ++ -- (POSTFIX)	Left-to-right
! ++ -- - (PREFIX)	Right-to-left
() (CAST)	Right-to-left
*** / %**	Left-to-right
+ -	Left-to-right
< > <= >=	Left-to-right
== !=	Left-to-right
&&	Left-to-right
||	Left-to-right
= %=	Right-to-left

Data Types

The data, this is the good stuff. What can we store, manipulate, understand? There are 8 datatypes in CPlus, six of which are primitive, and two of which are complex

Int	32-bit integer
Size_t	64-bit integer
Bool	Boolean values true or false
Char	any of the asci values described as charlit above
String	any string between two "" quotes described above
Void	The type we return when we don't want to return anything
Struct(ID)	A struct with the name ID, cannot exist without ID
Pointer(typ)	A pointer to any of the above datatypes

It is worth noting that we have implemented arrays in CPlus, but have done so using pointers under the hood, instead of defining an explicit array type.

Declarations

CPlus requires that – inside of a function – all declarations (even with initializers) for any variable used later on in that function must take place first, before any operations. Declarations are explicitly defined in the Context Free Grammar section, and are summarized below:

(1) Literal Variables: Datatype followed by Value Identifier. Cannot directly declare a void variable. Legal variable declarations look like this:

```
int a; bool b; char c; string d; Node n; Edge e;
```

(2) Functions: Datatype followed by Function Identifier followed by LBRACE function-body RBRACE. Datatype can be either primitive or complex. Legal function declarations look like this:

```
int x { … }
bool* { … }
Node** { … } – where Node was previously defined as a struct
```

(3) Structures: `struct` keyword followed by a Struct Identifier then LBRACE struct-body RBRACE SEMI. Note that the struct body can only contain un-initialized Literal Variable declarations. Legal struct declarations look like this:

```
struct Node {
        int x;
        char* y;
        Edge z; /* previously declared struct Edge */
};
```

(4) Pointers: Datatype followed by * followed by Value Identifier. Can chain multiple * together in a single declaration. Legal pointer declarations look like this:

```
int* x;
bool** y;
Node* np;
```

(5) Arrays: Arrays are not explicit datatypes, and they resolve to pointers when referenced. They do, however, have their own declarations as syntactic sugar. Datatype – possibly complex – followed by Value Identifier the RSQUARE integer-literal LSQUARE. The integer-literal must be greater than 0 and is not optional. Variable length arrays are not supported. Legal array declarations look like this:

```
int a[10];
bool* b[10];
Node* nodeIndices[100];
```

(6) Initialization: With the exception of structs and arrays, all of the above declarations can optionally come paired with initializers. Furthermore, declarations of the same type can be chained together in a list through commas. Memory allocation is also possible in the initialization of a declaration. Legal initializations for declarations look like this:

```
int x = 5;
int y = (300 * 21 + 4 ) / 2;
int*x = (int*) malloc(20*sizeof(int)); /* defacto 20-int array */
Node* nodes = (Node*) malloc(sizeof(Node));
int a = 5 , b = 6, q = x + 1;
```

Scope Rules

Standard static scoping rules apply, i.e. it is determined spatially based on position within the code at compile time.

Context-Free Grammar

TERMINALS: in uppercase bold

EOF	Epsilon	ID	LPAREN	RPAREN
LBRACE	RBRACE	COMMA	STRUCT	STRUCT_ID
SEMI	INT	SIZE_T	BOOL	STRING
CHAR	VOID	TIMES	LSQUARE	RSQUARE
ASSIGN	RETURN	IF	ELSE	FOR
WHILE	MOD_ASSIGN	OR	AND	EQ
NEQ	LT	GT	GEQ	LEQ
NOT	MINUS	INC	DEC	DOT
ARROW	PRINTF	PRINT	PRINTB	PRINTBIG
MALLOC	FREE	STRDUP	ATOI	PLUS
TIMES	DIVIDE	MOD	LITERAL	STRINGLIT
CHARLIT	TRUE	FALSE	AMP	SIZEOF
NULL				

nonterminals: in lowercase italics

program	decls	declaration	func_decl	struct_decl
typ	formals_opt	formal_list	declaration_list	stmt_list
init_declarator_list	expr	add_expr	init_declarator	stmt
selection_statement	iteration_statement	expr_opt	assignment_expression	logical_or_expr
postfix_expr	logical_and_expr	equality_expr	relational_expr	add_expr
unary_expr	cast_expr	unary_operator	postfix_expr	built_in_expr
actuals_list_opt	primary_expr	actuals_list	mult_expr	

Productions:

program →
decls **EOF**

decls →
decls declaration
| *decls func_decl*
| *decls struct_decl*
| **Epsilon**

func_decl →
typ **ID LPAREN** *formals_opt* **RPAREN LBRACE** *declaration_list stmt_list* **RBRACE**

formals_opt →
Epsilon
| *formal_list*

formal_list →
typ **ID**
| *formal_list* **COMMA** *typ* **ID**

struct_decl →
STRUCT STRUCT_ID LBRACE *declaration_list* **RBRACE SEMI**

typ →
INT
| **SIZE_T**
| **STRING**
| **BOOL**
| **CHAR**
| **VOID**
| **STRUCT_ID**
| *typ* **TIMES**

declaration_list →
Epsilon
| *declaration_list declaration*

declaration →
typ init_declarator_list **SEMI**

init_declarator →
ID
| **ID LSQUARE** *add_expr* **RSQUARE**
| **ID ASSIGN** *expr*

init_declarator_list →
init_declarator
| *init_declarator_list* **COMMA** *init_declarator*

stmt_list →
Epsilon
| *stmt_list stmt*

stmt →
expr **SEMI**
| *selection_stmt*
| *iteration_stmt*
| **RETURN SEMI**
| **RETURN** expr **SEMI**
| **LBRACE** *stmt_list* **RBRACE**

selection_stmt →
IF LPAREN *expr* **RPAREN** *stmt* (%prec **NOELSE**)
| **IF LPAREN** *expr* **RPAREN** *stmt* **ELSE** *stmt*

iteration_stmt →
FOR LPAREN *expr_opt* **SEMI** *expr* **SEMI** *expr_opt* **RPAREN** *stmt*
| **WHILE LPAREN** *expr* **RPAREN** *stmt*

expr_opt →
Epsilon
| *expr*

expr →
assignment_expr

assignment_operator →
ASSIGN
| **MOD_ASSIGN**

assignment_expr →
logical_or_expr
| *postfix_expr assignment_operator expr*

logical_or_expr →
logical_and_expr
| *logical_or_expr* **OR** *logical_and_expr*

logical_and_expr →
equality_expr
| *logical_and_expr* **AND** *equality_expr*

equality_expr →
relational_expr
| *equality_expr* **EQ** *relational_expr*
| *equality_expr* **NEQ** *relational_expr*

relational_expr →
add_expr
| *relational_expr* LT *add_expr*
| *relational_expr* GT *add_expr*
| *relational_expr* LEQ *add_expr*
| *relational_expr* GEQ *add_expr*

cast_expr →
unary_expr
| **LPAREN** *typ* **RPAREN** *cast_expr*

unary_operator →
NOT
| **MINUS**

unary_expr →
postfix_expr
| *unary_operator postfix_expr*
| **INC** *postfix_expr*
| **DEC** *postfix_expr*

postfix_expr →
built_in_expr
| *postfix_expr* **INC**
| *postfix_expr* **DEC**
| *postfix_expr* **LPAREN** *actuals_list_opt* **RPAREN**
| *postfix_expr* **LSQUARE** *postfix_expr* **RSQUARE**
| *postfix_expr* **DOT ID**
| *postfix_expr* **ARROW ID**

built_in_expr →
primary_expr
| **PRINTF LPAREN** *actuals_list_opt* **RPAREN**
| **PRINT LPAREN** *actuals_list_opt* **RPAREN**
| **PRINTB LPAREN** *actuals_list_opt* **RPAREN**
| **PRINTBIG LPAREN** *actuals_list_opt* **RPAREN**
| **MALLOC LPAREN** *actuals_list_opt* **RPAREN**
| **FREE LPAREN** *actuals_list_opt* **RPAREN**
| **ATOI LPAREN** *actuals_list_opt* **RPAREN**
| **STRDUP LPAREN** *actuals_list_opt* **RPAREN**

actuals_list_opt →
Epsilon
| *actuals_list*

actuals_list →
expr
| *actuals_list* **COMMA** *expr*

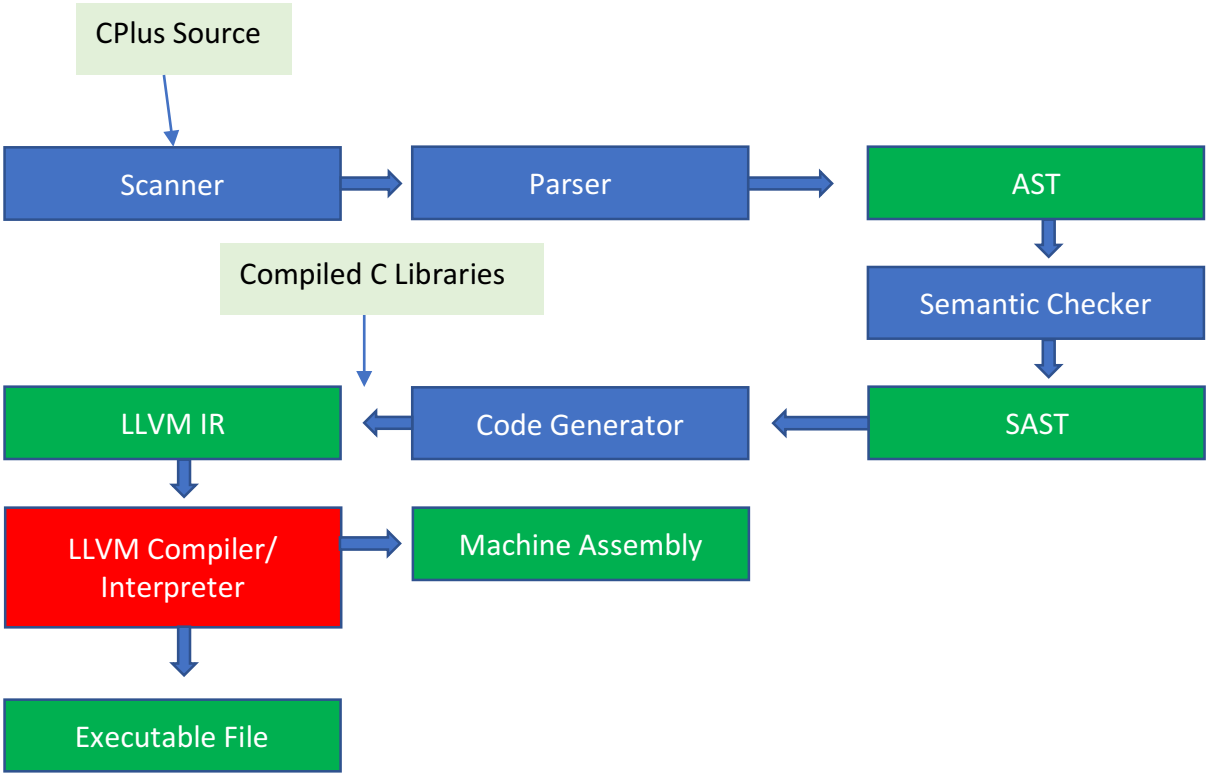
add_expr →
mult_expr
| *add_expr* **PLUS** *mult_expr*
| *add_expr* **MINUS** *mult_expr*

mult_expr →
cast_expr
| *mult_expr* **TIMES** *cast_expr*
| *mult_expr* **DIVIDE** *cast_expr*
| *mult_expr* **MOD** *cast_expr*

primary_expr →
LPAREN *expr* **RPAREN**
| **LITERAL**
| **STRINGLIT**
| **CHARLIT**
| **TRUE**
| **FALSE**
| **ID**
| **AMP** primary_expr
| **TIMES** primary_expr
| **SIZEOF LPAREN** *typ* **RPAREN**
| **NULL**

Architecture

The CPlus compiler makes 4 passes which are detailed in this section. This is modeled after the MicroC compiler provided by Dr. Edwards to the PLT class. All of these components were implemented by Alexander Stein. This block diagram provides an overview



Scanner

This is the simplest part of the CPlus compiler, it simply reads the input file and creates tokens (all listed in the table of nonterminals from the context free grammar section of the LRM) to feed line-by-line into the Parser. There was little innovation of note in the construction of our scanner due to the relatively small number of keywords, built-in functions, and terminals in CPlus.

Parser

The parser takes the token input from the scanner and determines whether to shift in new tokens or reduce by the rules it contains (mirroring the context-free grammar). Successful parsing produces an Abstract Syntax Tree. The Abstract Syntax Tree is defined in a separate file (see appendix), and enforces the rules set out in the parser.

Our parser is vastly different from the base MicroC parser which we built upon. It closely follows the format of the Kernighan & Ritchie “C Programming Language” manual. For example, while the Ocamlyacc paradigm allows us to explicitly force operator precedence, we do not use this functionality because our operator precedence is implicit in the grammar.

Semantic Checker

The semantic checker traverses the abstract syntax tree and determines whether the constructs it holds adhere to the intended meaning of the language, which cannot be implicitly encoded in the context-free grammar. If there are issues, the program is rejected and meaningful errors are sent to stdout, otherwise, the compiler moves on to the code generation phase.
Our language is “weakly typed.” That is code for not semantically-checked enough.

Code Generator

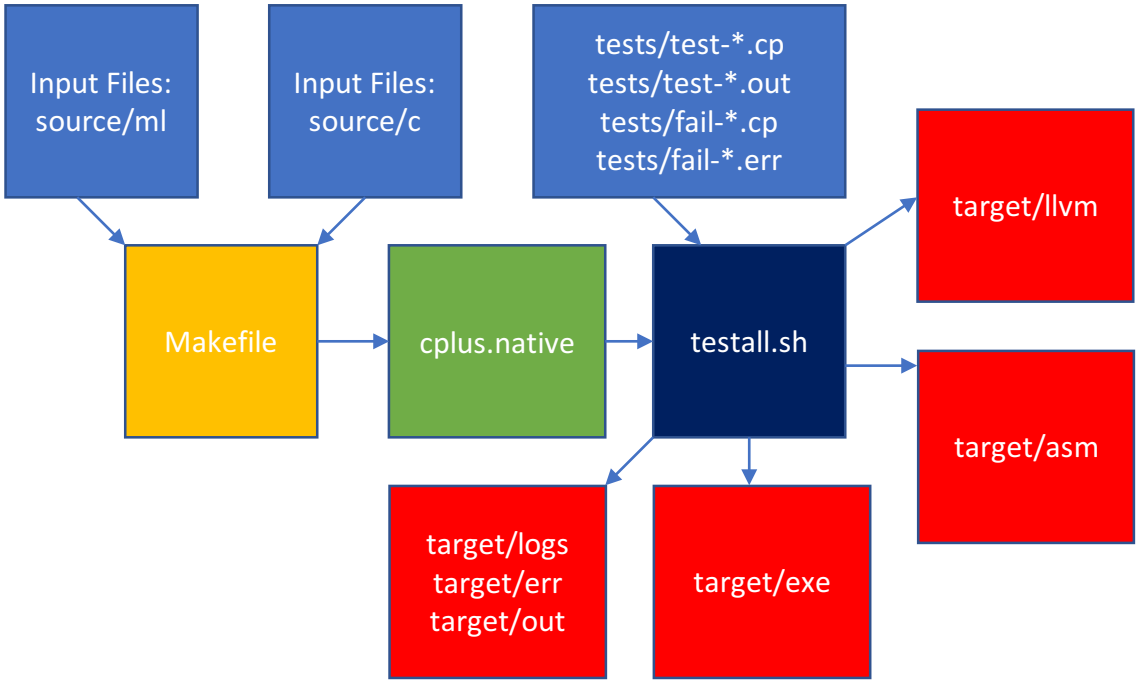
The code generator takes the semantically-checked AST as input, and attempts to produce LLVM intermediate representation code. This is the most time-consuming and difficult part of building the CPlus compiler, as it involves thinking in registers, pointers, and bit-widths instead of high-level programming constructs. Allocating memory, offsetting pointers, casting between data-types, and other such needs make it a daunting challenge.

Testing Suite

Our testing methodology is also borrowed directly from MicroC. The tests and automated test-bench were developed by Alexander Stein. Not much has changed from MicroC's test-bench other than some housekeeping: We created a **source** directory to gather all the Ocaml and C input files, and a **target** directory to gather all of the outputs from **testall.sh** so that the run area does not get as cluttered. To run our test-bench, and see all of the outputs described below, run these commands in the untarred directory:

```
make clean
make
./testall.sh -k
```

Here is a small block diagram showing how the flow works



The original intentions (severely ambitious) of this language were to implement all of the needed features of C in order to create, load, and sort graph databases. While we never got anywhere close to achieving that goal, it was a major driver in determining the features we had a chance to develop, and by extension the tests we wrote to prove them. The major achievements of this language are structures, pointers, and arrays. As such, we include here 3 source language tests used to prove these features and their corresponding LLVM outputs.

test-application.cp

This test is by no means one of the earlier ones, but it demonstrates the use of many important language features. There are both local and global variables with initializers, there is a structure declaration, and there is an array of structures. This test also demonstrates more subtle features such as the *lvalue* vs *rvalue* versions of the ArrayAccess operator [], structure record retrieval, and looping.

```
int size = 1000;

struct Record {
    char* name;
    int student_id;
    int grade;
};

int main()
{
    Record class[size];
    Record tmp;
    int i = 0;

    for (i = 0; i < size; i++){
        tmp.student_id = size % (2 * i + 3);
        class[i] = tmp;
    }

    tmp = class[7];
    print(tmp.student_id);

    return 0;
}
```

test-application.ll

```
; ModuleID = 'CPlus'

@size = global i32 1000
@fmt = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.1 = private unnamed_addr constant [4 x i8] c"%s\0A\00"

declare i32 @printf(i8*, ...)
declare i8* @atoi(i8*)
declare i8* @strdup(i8*)
declare i32 @printbig(i32)

define i32 @main() {
entry:
  %class = alloca { i32, i32, i8* }*
  %tmp = alloca { i32, i32, i8* }
  %i = alloca i32
  store i32 0, i32* %i
  %size = load i32, i32* @size
  %mallocsize = mul i32 %size, ptrtoint ({ i32, i32, i8* }* getelementptr ({ i32, i32, i8* }, { i32,
i32, i8* }* null, i32 1) to i32)
  %malloccall = tail call i8* @malloc(i32 %mallocsize)
  %ary.malloc = bitcast i8* %malloccall to { i32, i32, i8* }*
  store { i32, i32, i8* }* %ary.malloc, { i32, i32, i8* }** %class
  store i32 0, i32* %i
  store i32 0, i32* %i
  br label %while

while:                                            ; preds = %while_body, %entry
  %i9 = load i32, i32* %i
  %size10 = load i32, i32* @size
  %relational_expr = icmp slt i32 %i9, %size10
  br i1 %relational_expr, label %while_body, label %merge

while_body:                                       ; preds = %while
  %size1 = load i32, i32* @size
  %i2 = load i32, i32* %i
  %mult_expr = mul i32 2, %i2
  %add_expr = add i32 %mult_expr, 3
  %mult_expr3 = srem i32 %size1, %add_expr
  %struct.ptr = getelementptr inbounds { i32, i32, i8* }, { i32, i32, i8* }* %tmp, i32 0, i32 1
  store i32 %mult_expr3, i32* %struct.ptr
  %tmp4 = load { i32, i32, i8* }, { i32, i32, i8* }* %tmp
  %i5 = load i32, i32* %i
  %ary.cast = zext i32 %i5 to i64
  %class6 = load { i32, i32, i8* }*, { i32, i32, i8* }** %class
  %ary.ptr = getelementptr { i32, i32, i8* }, { i32, i32, i8* }* %class6, i64 %ary.cast
  store { i32, i32, i8* } %tmp4, { i32, i32, i8* }* %ary.ptr
  %i7 = load i32, i32* %i
  %add_expr8 = add i32 %i7, 1
  store i32 %add_expr8, i32* %i
  br label %while

merge:                                            ; preds = %while
  %class11 = load { i32, i32, i8* }*, { i32, i32, i8* }** %class
  %ary.ptr12 = getelementptr { i32, i32, i8* }, { i32, i32, i8* }* %class11, i64 7
  %ary.val = load { i32, i32, i8* }, { i32, i32, i8* }* %ary.ptr12
  store { i32, i32, i8* } %ary.val, { i32, i32, i8* }* %tmp
  %struct.ptr13 = getelementptr inbounds { i32, i32, i8* }, { i32, i32, i8* }* %tmp, i32 0, i32 1
  %struct.val.student_id = load i32, i32* %struct.ptr13
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt, i32 0,
i32 0), i32 %struct.val.student_id)
  ret i32 0
}

declare noalias i8* @malloc(i32)
```

test-pointerderef.cp

This test is much more representative of the ~one-hundred others, in that it very simply tests a single feature or two. This particular test demonstrates pointer declarations, pointer address retrieval, and pointer dereference capabilities.

```
int main()
{
    int x;
    int * p;
    int y;
    x = 5;
    p = &x;
    y = *p;
    print(y);
    return 0;s
```

test-pointerderef.ll

```
; ModuleID = 'CPlus'
@fmt = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.1 = private unnamed_addr constant [4 x i8] c"%s\0A\00"

declare i32 @printf(i8*, ...)
declare i8* @atoi(i8*)
declare i8* @strdup(i8*)
declare i32 @printbig(i32)

define i32 @main() {
entry:
  %x = alloca i32
  %p = alloca i32*
  %y = alloca i32
  store i32 5, i32* %x
  store i32* %x, i32** %p
  %p1 = load i32*, i32** %p
  %deref = load i32, i32* %p1
  store i32 %deref, i32* %y
  %y2 = load i32, i32* %y
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt, i32 0,
i32 0), i32 %y2)
  ret i32 0
}
```

test-nodeEdgeHash.cp

This is another more complex test which demonstrates the use of functions, and some complex global structure declarations. It doesn't do much, but the reason it is included here is to demonstrate some of the limitations of the language. We have been unable to get arrays declared in global scope to properly initialize. The original strategy deployed to initialize arrays (and any initializer declaration for that matter) in code generation was to delay the actual calculation and assignment until the program entered the first basic block. This works fine local variable initializers and arrays, but not for globals. See in the LLVM file that the "hashArray[100]" is initialized as a null pointer when it is encountered, but no memory is ever allocated for it.

```
/* Hashable Code for Node/Edge ID */
int hashCode(int key, int size) {
   return key % size;
}

/* Node data structure */
struct Node {
        /* node_id is hash int id, node_label is string id, prop are properties */
        int node_id;
        char* node_label;
        char* prop;
};

/* Edge data structure */
struct Edge {
        /* edge_id is has int id, edge_label is string id, prop are properties */
        int edge_id;
        int source_id;
        int sink_id;
        char* edge_label;
        char* prop;

};

int size = 100;
Node* hashArray[100];
Node* dummyItem;
Node* item;

int main()
{
    Node n1, n2, n3;
    Edge e1, e2, e3;
    int h = hashCode(1005, size);
    print(h);
    return 0;
}
```

test-nodeEdgeHash.ll

```
; ModuleID = 'CPlus'

@item = global i32 0
@dummyItem = global i32 0
@hashArray = global { i8*, i8*, i32 }* null
@size = global i32 100
@fmt = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.1 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@fmt.2 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.3 = private unnamed_addr constant [4 x i8] c"%s\0A\00"

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

declare i8* @atoi(i8*)

declare i8* @strdup(i8*)

declare i32 @printbig(i32)

define i32 @main() {
entry:
  %n1 = alloca { i8*, i8*, i32 }
  %n2 = alloca { i8*, i8*, i32 }
  %n3 = alloca { i8*, i8*, i32 }
  %e1 = alloca { i8*, i8*, i32, i32, i32 }
  %e2 = alloca { i8*, i8*, i32, i32, i32 }
  %e3 = alloca { i8*, i8*, i32, i32, i32 }
  %h = alloca i32
  %size = load i32, i32* @size
  %hashCode_result = call i32 @hashCode(i32 1005, i32 %size)
  store i32 %hashCode_result, i32* %h
  %h1 = load i32, i32* %h
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt, i32 0,
i32 0), i32 %h1)
  ret i32 0
}

define i32 @hashCode(i32 %key, i32 %size) {
entry:
  %key1 = alloca i32
  store i32 %key, i32* %key1
  %size2 = alloca i32
  store i32 %size, i32* %size2
  %key3 = load i32, i32* %key1
  %size4 = load i32, i32* %size2
  %mult_expr = srem i32 %key3, %size4
  ret i32 %mult_expr
}
```

Project Planning

Planning

As a two-member CVN team working together remotely across a time-zone boundary, and both working full-time, planning was difficult. We tried to meet via Skype once a week with each-other, and once every other week with Dr. Edwards. Most of our communications, however, took place via instant messenger sporadically throughout the weeks.

Timeline

Please note that we did not keep a formal time-log, so the sessions mentioned below are really just milestones that were reached. There were many, many more nights and weekends than just 8 devoted to writing the actual code on our own. Also, tests were added throughout the process.

Session 1 – General concepts discussed, ideas presented coding the entire C language, adding inheritance, using this to create and sort a graph database

Session 2 – Language Reference Manual: working marathon over the course of two days where we poured through the Kernighan and Ritchie "C Programming Language" manual and selected salient features for our own language.

Session 3 – Spoke with Dr. Edwards to review our Language Reference Manual and to set expectations for the project. Discussed methods of graphics visualization, inheritance implementation, and whether to use inheritance at all.

Session 4 – Set up github repo, AWS instance. Installed OCaml, MicroC, and started working towards "Hello World" milestone. Test-suite implemented

Session 5 – All "primitive" features implemented, started moving on to complex ones beginning with pointers. Language implementation of pointers hits a huge snag. At this point the entire Context-Free Grammar has to be re-worked…

Session 6 – Application in C 50% complete; CFG is fixed, major hurdle at this point is figuring out the GEP instruction and differentiating between LHS pointer deref operator and RHS one. Spoke again with Dr. Edwards to get specific advice about implementation.

Session 7 – Application in C 80% complete; Pointer LHS/RHS deref is working, moving on to arrays and structures.

Session 8 – Application in C 100% complete; Structures, Pointers, Arrays working. Attempting to get application working in the language. Realization of the huge gap between what we were able to implement and the C features required by the Application.

Roles

Alexander Stein – Language Designer, Test-bench Designer, Manager, Report Writer
Eric Johnson – Sample Application Developer, Report Writer

Development Environment

Github was used to version control all of our code, and the repository can be found here, our language folder is called “cplus-llvm”. There were 3 branches tracked: alex-dev, master, eric-dev. The process was for each person to upload their changes to their own branch, and then pull-requests would be issued on a regular basis, after which we would meet to discuss merge conflicts:

https://github.com/tomodachi21/plt_c-graph

We also used an AWS server with Ubuntu 16.04 LTS installed on it to collaborate in real time when needed. If either of us was having a problem pulling from the repo or debugging some test, the other could log in and help fix the exact same files.

For the most part though, the language was developed and compiled onto LLVM and then x86 for MAC OSX Sierra.

Project Log

```
f648dfb  aistein    Thu May 11 12:00:13 2017 Merge pull request #5 from tomodachi21/alex-dev
79f202e  Alex Stein Thu May 11 11:59:25 2017 .
b2cdcac  aistein    Thu May 11 11:56:48 2017 Merge pull request #4 from tomodachi21/alex-dev
280b2e1  Alex Stein Thu May 11 03:44:33 2017 some cleanup
dd0d801  Alex Stein Wed May 10 19:24:18 2017          got array of pointers to struct dereference working
7106981  Alex Stein Wed May 10 17:13:31 2017          abandoning ship on the graph thing, arrays of struct pointers not working
5b5ebc5  Alex Stein Wed May 10 15:53:53 2017          basic version of insert_node sort of working
b3e4291  Alex Stein Wed May 10 13:19:39 2017          added some more charstar functionality
c076630  aistein    Wed May 10 12:11:42 2017          Merge pull request #3 from tomodachi21/alex-dev
9df5b2f  Alex Stein Wed May 10 12:10:44 2017          holy crap, got struct pointer dereference working
af4f21d  aistein    Tue May 9 21:50:01 2017  Merge pull request #2 from tomodachi21/eric-dev
83e668d  Alex Stein Tue May 9 21:48:56 2017  fixed merge
8e07960  aistein    Tue May 9 21:36:06 2017  Merge pull request #1 from tomodachi21/eric-dev
c534558  Ubuntu     Tue May 9 21:29:07 2017  merging
38f3c6e  Ubuntu     Tue May 9 21:21:29 2017  Merge branch 'master' of github.com:tomodachi21/plt_c-graph into eric-dev
5db739e  Alex Stein Tue May 9 21:13:14 2017  started including some more built-in functions
33c0fcc  Ubuntu     Tue May 9 21:10:20 2017          modified:   sample/c+graph.c
a841abf  Alex Stein Tue May 9 20:16:22 2017  added struct arrow operator, modified node test
099411a  Alex Stein Tue May 9 19:31:13 2017  basic char star working
31f848e  Alex Stein Tue May 9 18:56:43 2017  enabled basic Char, moving forward
f7a5ecd  Alex Stein Tue May 9 15:11:42 2017  struct semantic checking and access working correctly
a6be4c7  Alex Stein Mon May 8 23:50:57 2017  almost have struct element assignment working, now just need to semantically enable it
d0bf2a6  Alex Stein Mon May 8 23:33:45 2017  struct dot operator standalone working, though it seems struct items are allocated backwards
8253d8b  Alex Stein Mon May 8 19:42:57 2017  capable of allocating structs on the stack, currently can only contain primitive types and their pointers
1cce0bb  Alex Stein Mon May 8 17:53:49 2017  update
1debf89  Alex Stein Mon May 8 17:52:17 2017  merged
8a5d1cc  Alex Stein Mon May 8 17:48:01 2017  changed test postfix to cp from mc
a50fd70  Alex Stein Mon May 8 02:04:23 2017  fixed the lazy booboo
2aded19  Alex Stein Mon May 8 00:08:57 2017  added structs to ast, scanner, and parser, codegen and semant next
63b1791  Alex Stein Sun May 7 23:00:46 2017  rid of some warnings, man our semantic checking is severly under-developed
5329b1a  Ubuntu     Sun May 7 22:20:19 2017  fixed server error by changin anonymous functions
79de443  Ubuntu     Sun May 7 21:47:49 2017  Merge branch 'master' of github.com:tomodachi21/plt_c-graph
4bdfb56  Ubuntu     Sun May 7 21:47:33 2017  edits to c+graph.c
1a6b971  Alex Stein Sun May 7 21:33:18 2017  array access works, but have to hack your way around this laziness error
c1d7ad3  Alex Stein Sun May 7 16:00:21 2017  array access sort of working, but values incorrect
44026e0  Alex Stein Sun May 7 14:27:42 2017  need to refactor the way declarators, etc work in order to do arrays properly
4f0ac87  Alex Stein Sun May 7 13:06:34 2017  added postfix and prefix increment decrement, as well as mod-equals
4329ba1  Alex Stein Sun May 7 12:01:04 2017  pointer ADDITION ONLY working, its a start
00177a8  Alex Stein Sun May 7 10:31:54 2017  merging with master
```

```
0c42458 Ubuntu      Sun May 7 10:28:36 2017                 new file:  sample/c+graph new file:  sample/c+graph.c              new file:  sample/t_node.csv
        new file:  sample/test      new file:  sample/test.c    new file:  sample/test2     new file:  sample/test2.c
29e9d13 Alex Stein  Sun May 7 10:26:05 2017 still trying to enable pointer addition, not working
a8450f9 Alex Stein  Sat May 6 18:39:02 2017 removed lots of warnings
c1229e7 Alex Stein  Sat May 6 18:27:58 2017 mallocassign is working, time to move on to arrays
f98e477 Alex Stein  Sat May 6 17:34:32 2017 major breakthrough towards mallocassign, essentially implemented lazy intialization
e43b5d5 Alex Stein  Sat May 6 17:03:03 2017 got array declarations working, still cant declare and assign a pointer on the same line
aa876d7 Alex Stein  Sat May 6 12:46:36 2017 added semantic checking for malloc
c1a11cc Alex Stein  Sat May 6 12:08:22 2017 ...
70d2dbf Alex Stein  Sat May 6 12:04:44 2017 added capability to malloc sizes larger than one, still need to test further
af91f4a Alex Stein  Tue May 2 11:24:26 2017 going to branch before trying to use objects to solve mutual recursion problem
9bfbef1 Alex Stein  Sat Apr 29 21:53:55 2017 declaring initialization lists now works, check out declareinit.mc
0f0ef43 Alex Stein  Thu Apr 27 21:43:01 2017 holy shit, LH pointer assign is working.
8d3d203 Alex Stein  Tue Apr 25 21:06:24 2017 baby steps, turned a program from a tuple into a record of lists
6b7cbc5 Alex Stein  Tue Apr 25 19:05:35 2017 couldn't get LH pointers working, nuclear option of copying CIMPLE initiated
39b41ec Alex Stein  Sat Apr 22 22:52:35 2017 got pointer deref basics working
ce0594e Alex Stein  Sat Apr 22 15:32:40 2017 more target enforcement
5bb7702 Alex Stein  Sat Apr 22 15:30:30 2017 placeholders to enforce target structure, updated operator precedence
d45d64e Alex Stein  Sat Apr 22 15:16:07 2017 pointer addressOf function sort of working, moving on
c9142a9 Ubuntu      Sat Apr 22 14:44:36 2017 target needs to be in repo for this to work
c2381c2 Alex Stein  Sat Apr 22 13:58:50 2017 pushing working version for Eric
86878cc Alex Stein  Mon Apr 17 09:34:03 2017 added ampersand operator, but its not working yet
1bf5e42 Alex Stein  Sat Apr 15 19:14:43 2017 added free function, replaced C import with LLVM calls for build_malloc and build_free
7277434 Alex Stein  Fri Apr 14 20:54:28 2017 malloc tests working, time for next steps
049e769 Alex Stein  Fri Apr 14 20:07:52 2017 mallocexpr working
3a0900b Alex Stein  Fri Apr 14 20:02:00 2017 got casting working for voidptr to intptr
51764ce Alex Stein  Thu Apr 13 19:33:55 2017 pointer assign is working, just needed to change from physical to structural equality in semanta
9731298 Alex Stein  Thu Apr 13 14:29:07 2017 more tests
e0f9e6b Alex Stein  Thu Apr 13 13:42:35 2017 still don't have pointerassign working, but made some progress with casting etc.
3b80143 Alex Stein  Thu Apr 13 01:06:06 2017 semantic checking not working for pointerasign int*=int*
c328b57 Alex Stein  Thu Apr 13 00:16:15 2017 sizeof operator with some progress, test is passing with minimal checking
2bc5206 Alex Stein  Wed Apr 12 23:28:06 2017 got malloc and sizeof compiling, but no semantic checking, antd the pointer-assign test is failing
a1a6754 Alex Stein  Tue Apr 11 19:27:09 2017 pointer declarations working, no actual assignment yet
8f23ef0 Alex Stein  Sun Apr 2 15:02:27 2017 fixed codegen.ml so that print didn't have garbage, used clang on printf.c to figure it out
2ea3f67 Alex Stein  Thu Mar 30 23:05:08 2017 created target area thru test shell scripts
ffddf15 Alex Stein  Tue Mar 28 00:36:16 2017 fixed makefule tar facility
e9635b3 Alex Stein  Tue Mar 28 00:15:47 2017 crappy but semi-functional hello-world delivery
599a136 Alex Stein  Mon Mar 27 22:32:03 2017                 almost there
3f3eb5d Alex Stein  Mon Mar 27 22:14:06 2017                 llvm is doing the proper ASCII character input, but printf is not reading it out properly
bac9e74 Alex Stein  Mon Mar 27 21:25:07 2017                 got codegen doing something for strlit example
fa442a7 Alex Stein  Mon Mar 27 21:04:29 2017                 codegen building, just not passing strlit test
590f768 Alex Stein  Mon Mar 27 20:42:33 2017                 sharing latest broken code
ca7cead Alex Stein  Mon Mar 27 18:39:02 2017                 it ain't workin totally, but its damn close... on codegen step for StringLit
7feada9 Alex Stein  Mon Mar 27 09:01:05 2017                 started messing with the ocaml code for cplus, moved everything into cplus
55d2cb6 Alex Stein  Mon Mar 27 02:30:09 2017                 merging
c865ac2 Alex Stein  Mon Mar 27 02:29:06 2017                 .
787a2b6 Alex Stein  Mon Mar 27 02:18:35 2017                 merging
6b30133 Alex Stein  Mon Mar 27 02:16:59 2017                 setup test area for cplus, mocked microc automation, still some fixes needed
0bebedf Ubuntu      Sun Mar 26 23:54:23 2017 .
dc4ba2b Alex Stein  Sun Mar 26 23:48:31 2017 fixed problem with microC testing
d612502 Alex Stein  Sun Mar 26 23:26:56 2017 removed the crap
3f2aebb Alex Stein  Sun Mar 26 23:24:51 2017 got it working on my local machine
380e703 Ubuntu      Sun Mar 26 22:55:49 2017 fixing gitignore file
4215fbf Ubuntu      Sun Mar 26 22:53:44 2017 successfully installed llvm.3.7 and tested microC
199db16 Ubuntu      Sun Mar 5 13:32:16 2017                 new file:  test.ml
27aedd3 tomodachi21 Sun Mar 5 13:12:46 2017 Initial commit
```

Learnings

Alex Stein – Language Designer

First and foremost, I should have listened – really, really listened – to Dr. Edwards when he mentioned that developing the entire C language was outside the scope of this class. This was learned the hard way when it was already too late to change course. As a CVN student, time is limited to begin with, and our undertaking was far too large for a team of two people to undertake. The concept for our intended graphing application was exciting and we were swept up in it, but came nowhere close to getting it working. If I had to give advice to future groups about this particular issue, I'd summarize it like this: "This class is not for learning new algorithms, or for writing interesting applications in languages you already know. It's for writing compilers. Pick a small set of features and implement them really well."

As echoed by students in the past about OCaml "never have I spent so much time writing so little that does so much." It was fun to discover the pattern-matching is the solution to almost every problem. Need to do different things when assigning to an array as compared to a pointer, or a dereference? No problem, pattern matching. Need to write a recursive function that returns the indices for your structure-pointer offset? No problem, pattern matching. Need to add special "escape" sequences to stop your semantic checker from keeping your AST from the really hard stuff (the code generation)? Add some pattern-matching to your pattern-matching. I'm a little upset that it doesn't let me put comments wherever I want.

The OCaml LLVM IR is extremely useful, and I'm thankful that there is *some* documentation. That said, the C++ LLVM documentation is far more extensive, I used it quite a bit. If I were to use LLVM professionally, I think I'd prefer to do so with some sort of C++ interface, which allow more control over how exactly to generate the LLVM I'd like.

Eric Johnson – Application Developer

One thing I think the Professor Edwards predicted was the need to reduce the scope of our project. I think at the beginning of the semester a lot of student underestimate the amount of work that is needed to implement a language and all of the complexities that come up. The original scope of our project included both RDF Graphs and Property Graphs but I think we spent so much time actually implementing the language to get it to a usable state that for two people the original proposal was too ambitious. I think that one of our greatest achievements was to get the arrays of struct pointers working as well as dereferencing of structs. Perhaps if we had more people we could have gone on to implement more fancy data structures like matrices but for our purposes I think we achieved a sizable amount – we implemented a fully in-memory graph abstraction in a custom language and showed the ETL (extract transform load) process that can occur when getting RDMS data into a true graph structure.

One general comment I think is true of our group is that we are both working students – which is really challenging. Because of this you end up having to put the majority of your work on weekends or very late night – pulling all nighters and then waking up at 7 am to go into the office the next day. This is just a general struggle working professionals have. That being said I think that my depth of knowledge in the subject compared to before I started the course has progressed significantly. Although we may not have the time to commit to the course as do

some undergraduate students – Professor Edwards was an excellent instructor and did a great job and making the material both approachable and easy to understand.

In terms of our project – I think that we did get started a little bit later than some of our classmates and that made it more challenging. If we had more time I think we could really take this C+Graph language much further and make it a full blown desktop application for users. I look forward to implementing some more complex algorithms and data-structures in the future. Now that the majority of the requirements to implement the working C+Graph

Appendix A: C+Graph Application Tutorial

The c+graph application allows users to import up to 1000 nodes and edges into memory and extrapolates traditional RDMS data into a true graph abstration. As you can see on in the introduction there are three main types of structures in Property Graphs: 1) nodes, 2) edges, and 3) properties. As you can see in the code we have made each of these its own struct that can be invoked in memeory.

```
/* Property Data Structure: Stores a key-value property pair for either a given node or edge */

struct prop {

    int prop_id;

    char* property;

    char* value;

};



/* Node Data Structure: Stores the node object as well as an array of pointers to its property objects */

struct node {

    int node_id;

    int prop_cnt;

    char* node_label;

    struct prop *property[10];

};



/* Edge Data Structure: Stores the edge object as well as pointers to its src (source) and trg (target) node objects */
```

```
struct edge {
    int edge_id;
    char* edge_label;
    struct node *src;
    struct node *trg;
};
```

Here we see that a node is definied by a “node_label” and can have any number of properties associated with it. Each property of a node is stored as a pointer in the array **struct prop *property**. For default we set this to 10 properties per node but this number could be larger. For lazy reference we also keep attributes associated with the node such as prop_cnt (the number of properties the node has) and node_id (the index in our global node array that helps keep track of all nodes in existence). When a user uploads a node.csv file the first list is treated as a descriptor for what the properties being imported are called. For example if we take the first line of t_node.csv we can see that it has: node_id,Age,Height,Phone. This tells us that for each row we have a series of comma separated values – the first being the name/identifier of the node then followed by the persons age, height, and telephone number.

Each time a file is ingested it keeps an index of the property name associated with each value. This key-value pair is created as a **prop** that is then stored in the associated row's **node** object. So anytime I want to get a property associated with a node I retrieve the nodes property array and then can jump to the value of the property that is stored for that node.

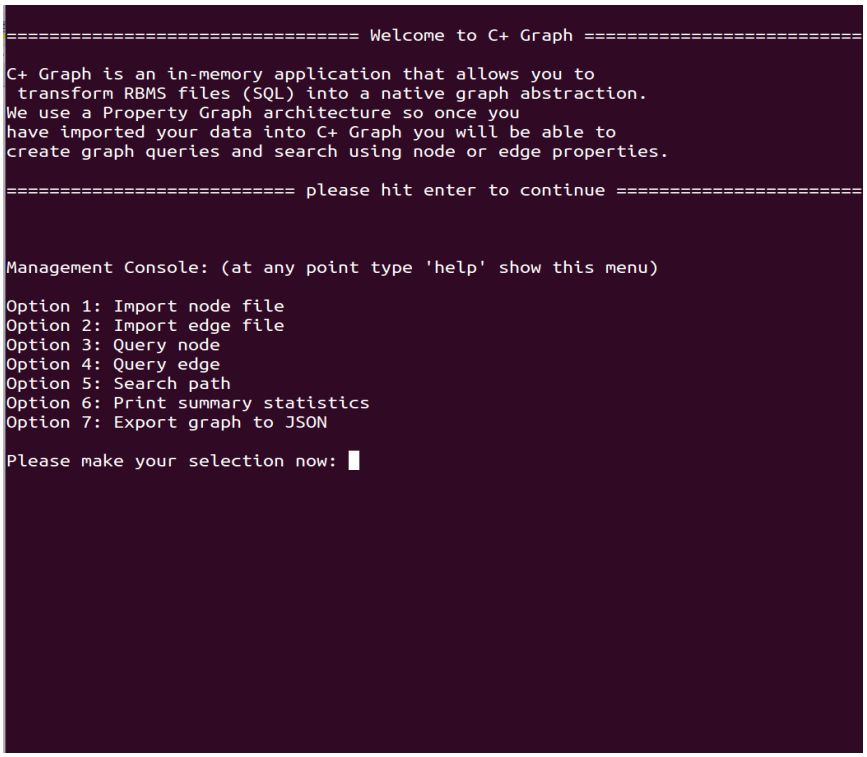
The **edge** struct is the link that exists between each node. When you import an edge file such as t_edge.csv the file will search for the pointer reference to each of the nodes that are listed (first being source and second being target) and will create a new edge object that has these two pointers stored inside of it. There are a lot of tricks that can be done here with regards to indexing and how your create index referencing. For this project I just have a global hash array called edgeIndex that keeps track of all of the edges in existence. Another form of redunancy that could be done is to have two separate arrays stored inside of the **node** object that keep references to **edge** pointers for which the node is both a **source** and a **target** for – these would be two separate arrays. Alternatively if you chose to store the source and target nodes within a two column matrix or a much larger **adjacency matrix** you could do quick traversals using linear algebra. Because our language does not support matrices we chose to go with the array pointer method but I am curious as to the benefits/drawbacks to either.

Here is a quick summary of the code which is currently about 400 lines. I look to keep adding to the project now that a large portion of the implementation is done. A large amount of time was

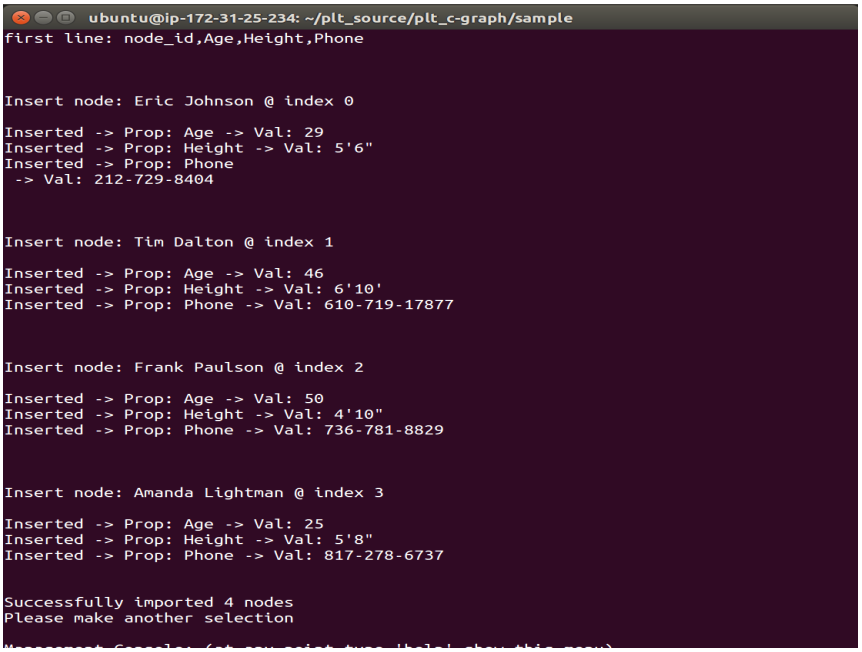
spent getting the language working and the GUI. Because of things like carriage returns in user input or hiding in files people import a lot of debugging was doing to make sure the search algorithms worked correct.

User Interface Guide (with sample data):

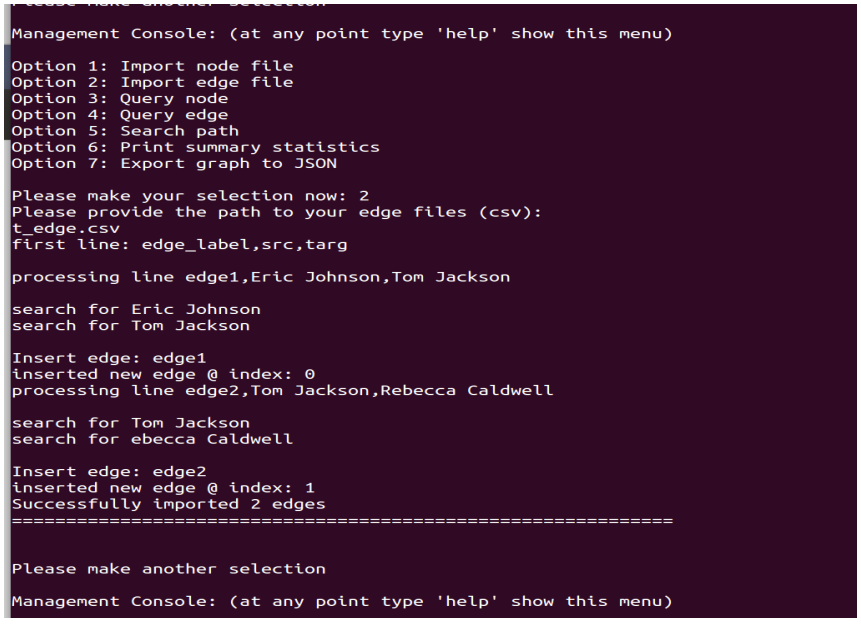
Introduction Screen – User GUI that provides the user access to 7 options through stdin. At any point during the interface the user can type the command “help” to jump back to the main menu.



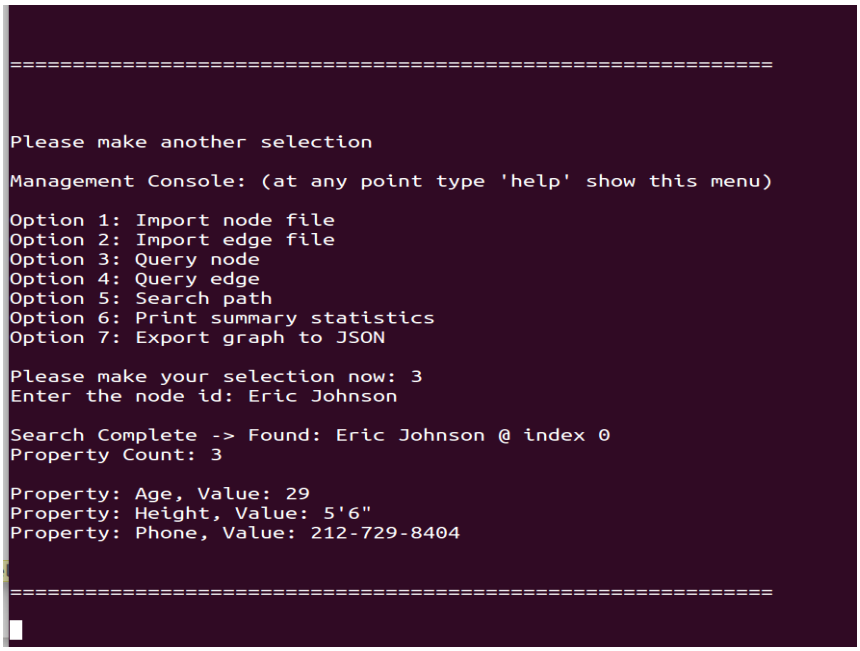
Import Node – A user can provide a CSV such that the first column is the name of the node and the columns that follow are all of the properties associated with that node. Each sheet that is imported will take the column header as the name of the property and the entry as its value. First the properties are created then they are stored in the node's property array.



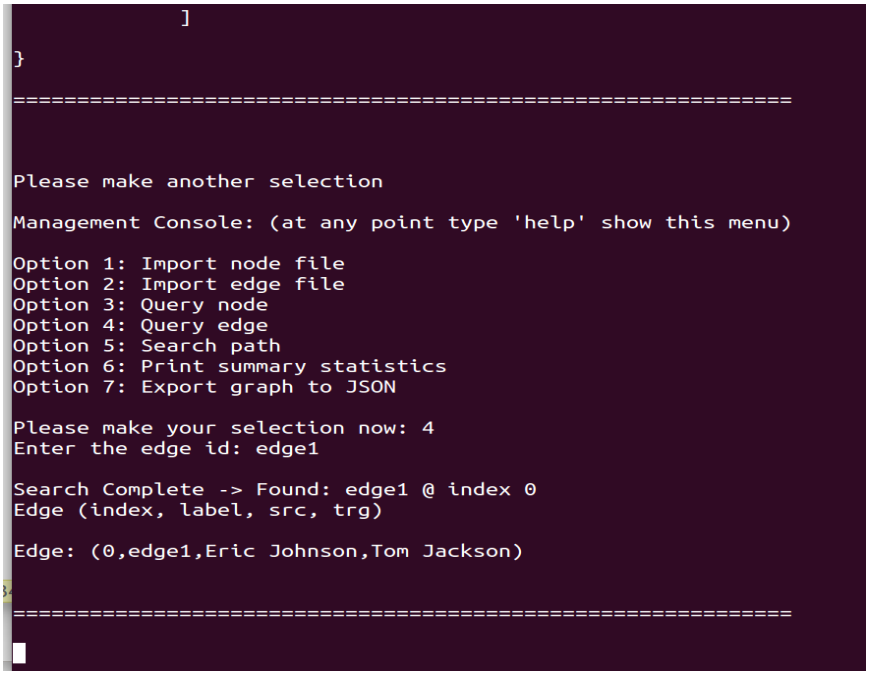
Import Edge – The user can specify a edge file that has the name of the edge followed by the name of each node. When it imports the edges it creates an edge and then stores a reference to the pointer of the nodes that exist in that edge within the property.



Search Node – The user can enter a node label (the name of the node) and the system will return all of the properties associated with it. It does this by accessing the prop pointers that exist on the node and then dereferencing the data that is store at each prop pointer.



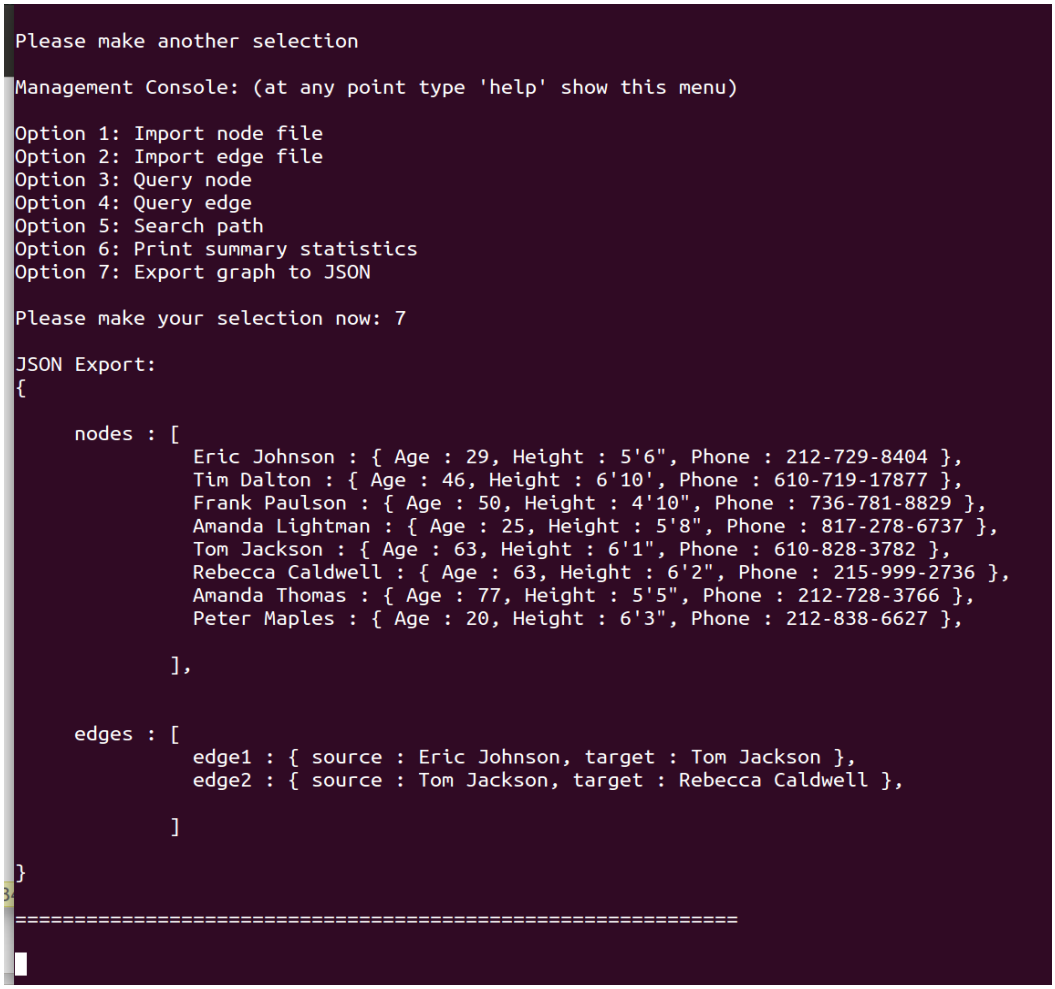
Search Edge – The user can enter a edge label (the name of the edge) and the system will return all of the pointers that associated with it. The names here are shown by dereferencing the node labels that are associated with each of those nodes. In a future iteration I would like to adapt the edge to have properties stored directly on it as well.



Graph Summary – Allows the user to summarize the current number of active nodes and edges that exist in the graph. Here you can see all of the indices that are associated with each element that is stored in the globa hash array for nodes and edges.

```
Node: (7,Peter Maples)

Graph Summary - Edges: 0
Edge (index, label, src, trg)



============================================================



Please make another selection

Management Console: (at any point type 'help' show this menu)

Option 1: Import node file
Option 2: Import edge file
Option 3: Query node
Option 4: Query edge
Option 5: Search path
Option 6: Print summary statistics
Option 7: Export graph to JSON

Please make your selection now: 6

Graph Summary - Nodes: 8
Node (index, node id)

Node: (0,Eric Johnson)
Node: (1,Tim Dalton)
Node: (2,Frank Paulson)
Node: (3,Amanda Lightman)
Node: (4,Tom Jackson)
Node: (5,Rebecca Caldwell)
Node: (6,Amanda Thomas)
Node: (7,Peter Maples)

Graph Summary - Edges: 0
Edge (index, label, src, trg)
```

JSON Output – Once the graph as been fully loaded into memory it can be persisted using the export to JSON option. As you can see below this is a way of storing the graph in plain text. In this current implementation of the tool I do not have a “reimport” option where the user can import the JSON format back into a graph – would like to add that at a later date. As you can see the nested structure of JSON helps retain the key-vlaue store relationship between attributes and value.



Code Overview:

Here you can see the structure of the objects (structs) that we used in the program. There were 3 main object types, the prop – property, the node – node object, and the edge – edge object. Each contains pointers to other objects.

```
@ip-172-31-25-234: ~/plt_source/plt_c-graph/sample
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

/* ============================ Data Structures  ================================= */

char *prop_names[10];
struct node* nodeIndex[1000];
struct node* dummyItem;
struct node* item;
struct edge* edgeIndex[1000];
struct edge* dummyItemEdge;
struct edge* itemEdge;
struct prop* propIndex[1000];
struct prop* dummyItemProp;
struct prop* itemProp;

/* Property Data Structure: Stores a key-value property pair for either a given node or edge */
struct prop {
        int prop_id;
        char* property;
        char* value;
};

/* Node Data Structure: Stores the node object as well as an array of pointers to its property objects */
struct node {
        int node_id;
        int prop_cnt;
        char* node_label;
        struct prop *property[10];
};

/* Edge Data Structure: Stores the edge object as well as pointers to its src (source) and trg (target) node o
struct edge {
        int edge_id;
        char* edge_label;
        struct node *src;
        struct node *trg;
};


/* ============================ Creation Functions ================================= */

/* Creates a Node Object and inserts into Hashable Index for Reference */
int insert_node(char* key) {

   struct node *item = (struct node*) malloc(sizeof(struct node));
   item->node_label = strdup(key);
   item->prop_cnt = 0;

"c+graph.c" 509L, 14633C
```

Here is a quick screen-shot of the logic that we used to make the GUI for the user. Although it is simple text being displayed to the user this is the majority of how the program operates. I wanted to make it so that the user could always return to the menu when needed to help is an escape phrase.

```c
void menu() {

    printf("\nManagement Console: (at any point type 'help' show this menu)\n\n");
    printf("Option 1: Import node file\n");
    printf("Option 2: Import edge file\n");
    printf("Option 3: Query node\n");
    printf("Option 4: Query edge\n");
    printf("Option 5: Search path\n");
    printf("Option 6: Print summary statistics\n");
    printf("Option 7: Export graph to JSON\n\n");
    printf("Please make your selection now: ");
}

void logic(char* input) {

  if(strcmp(input, "help") == 1){
    menu();
  }
  else {
  switch (*input) {
  case '1':
    printf("\nPlease provide the path to your node files (csv):\n");
    read_nodes();
    break;
  case '2':
    printf("Please provide the path to your edge files (csv):\n");
    read_edges();
    printf("\n============================================================\n\n");
    break;
  case '3':
    printf("Enter the node id: ");
    struct node *temp_node = search();
    display_node(temp_node);
    printf("\n============================================================\n\n");
    break;
  case '4':
    printf("Enter the edge id: ");
    struct edge *temp_edge = search_edge();
    display_edge(temp_edge);
    printf("\n============================================================\n\n");
    break;
  case '5':
    printf("Enter node1: ");
    printf("Enter node2: ");
    printf("\n============================================================\n\n");
    break;
  case '6':
    printf("\nGraph Summary - Nodes: ");
    display();
```

Here we can see the code that imports the edges from an edge file. First it strips out the first line which is a header and then it goes line by line searching for the nodes in the edge and returning their pointers. With these pointers it is then able to create a new edge entry and store the pointer to the associated nodes in its **src** and **trg.**

@ip-172-31-25-234: ~/plt_source/plt_c-graph/sample

```
/* Read and Import the file provided for Edges into memory and reference appropriate Node Objects (can use sa
void read_edges() {
   int line_count;
   char buffer[1024];
   char *line;
   char fileparam[80];
   scanf("%s", fileparam);
   FILE *fstream = fopen(fileparam,"r");
   if(fstream == NULL)
   {
      printf("\n file opening failed ");
      read_nodes();
   }

   int i = 0;
   line_count = 0;
   line = fgets(buffer,sizeof(buffer),fstream);
   printf("first line: %s\n", line);

   while((line=fgets(buffer,sizeof(buffer),fstream))!=NULL)
   {
     printf("processing line %s\n", line);

     line_count++;
     int i = 0;
     char *array[10];
     array[i] = strtok(line,",");

     while(array[i]!=NULL)
     {
         array[++i] = strtok(NULL,",");
     }

     printf("search for %s\n", array[1]);
     struct node *temp1 = get_node(array[1]);
     printf("search for %s\n", array[2]);
     struct node *temp2 = get_node(array[2]);
     insert_edge(temp1, temp2, array[0]);
   }
   printf("Successfully imported %i edges",line_count);
}


void menu() {

    printf("\nManagement Console: (at any point type 'help' show this menu)\n\n");
    printf("Option 1: Import node file\n");
    printf("Option 2: Import edge file\n");
    printf("Option 3: Query node\n");
    printf("Option 4: Query edge\n");
```

Here you can see how the insert_edge function actually works. For both the import node and import edge functions there is first a function that is called to read the file and then for each line in the file the “insert” function is called to actually create the objects.

@ip-172-31-25-234: ~/plt_source/plt_c-graph/sample

```c
/* Creates a Edge Object (owns references to src and trg node pointers) and inserts into Global Hashable Index
struct edge *insert_edge(struct node *src, struct node *trg, char* label) {

   struct edge *itemEdge = (struct edge*) malloc(sizeof(struct edge));
   itemEdge->src = src;
   itemEdge->trg = trg;
   itemEdge->edge_label = strdup(label);

   int hashIndex = 0;
   printf("Insert edge: %s\n", label);

   while(edgeIndex[hashIndex] != NULL && edgeIndex[hashIndex]->edge_label != label) {
      ++hashIndex;
   }

   itemEdge->edge_id = hashIndex;
   edgeIndex[hashIndex] = itemEdge;
   printf("inserted new edge @ index: %i\n", hashIndex);
   return itemEdge;
}


/* Count the number of occurance of a character when passed a line of text and the character */
int countChars( char* s, char c )
{
    return *s == '\0'
              ? 0
              : countChars( s + 1, c ) + (*s == c);
}

/* Accepts each line of text from RDMS File (CSV) and processes into Node Objects */
char* create_node(char* line)
{
        int i=0;
        char *array[10];

        int c = countChars(line, ',');
        array[i] = strtok(line,",");

        while(array[i]!=NULL)
        {
                array[++i] = strtok(NULL,",");
        }

        int node_index = insert_node(array[0]);
        int prop_cnt = nodeIndex[node_index]->prop_cnt;
        for (i = 1; i <= c; ++i)
        {
                printf("Inserted -> Prop: %s -> Val: %s\n", prop_names[i], array[i]);
```

Here you can see the search function which will either prompt the user to enter a string and then search for the matching node with the correct node label. The get_node function is a little different because it is meant to return the node based on a parameter that is passed to it – this makes dereferencing in other functions easier.

@ip-172-31-25-234: ~/plt_source/plt_c-graph/sample

```
/* ================================ Graph Functions ============================== */

/* Search for Node based on user input of node_label - uses Hashable Index */
struct node *search() {
   int hashIndex = 0;
   char line[100];
   fgets(line, 100, stdin);

   while(nodeIndex[hashIndex] != NULL) {

      if(strcmp(strtok(nodeIndex[hashIndex]->node_label, "\n"), strtok(line, "\n")) == 0){
         printf("\nSearch Complete -> Found: %s @ index %i", line, hashIndex);
         return nodeIndex[hashIndex];
       }

      printf("Node label %s is compared to %s with value %i\n", nodeIndex[hashIndex]->node_label, line, strcmp
      hashIndex++;
   }
   return NULL;
}

/* Search for Edge based on user input of edge_label - uses Hashable Index */
struct edge *search_edge() {
   int hashIndex = 0;
   char line[100];
   fgets(line, 100, stdin);

   while(edgeIndex[hashIndex] != NULL) {

      if(strcmp(strtok(edgeIndex[hashIndex]->edge_label, "\n"), strtok(line, "\n")) == 0){
         printf("\nSearch Complete -> Found: %s @ index %i", line, hashIndex);
         return edgeIndex[hashIndex];
       }

      printf("Edge label %s is compared to %s with value %i\n", edgeIndex[hashIndex]->edge_label, line, strcmp
      hashIndex++;
   }
   return NULL;
}

/* Return a pointer to the node reference that matches a particular node_label id */
struct node *get_node(char* node_label) {
   int hashIndex = 0;

   while(nodeIndex[hashIndex] != NULL) {

      if(strcmp(strtok(nodeIndex[hashIndex]->node_label, "\n"), strtok(node_label, "\n")) == 0){
         return nodeIndex[hashIndex];
       }
      hashIndex++;
```

Here you can see the display functions that basically handle dereferencing the objects and getting their attributes for display in the console. This was a little tricky at first but was easy once figured out for one object type.

@ip-172-31-25-234: ~/plt_source/plt_c-graph/sample

```c
/* Display contents of a node object by accepting the node pointer as parameter */
void display_node(struct node *node_id) {
   int prop_cnt = node_id->prop_cnt;
   printf("\nProperty Count: %i\n\n", prop_cnt);

   int i;
   for(i = 0; i < prop_cnt; i++) {
         struct prop *temp= node_id->property[i];
         printf("Property: %s, Value: %s\n", temp->property, temp->value);
   }

   printf("\n");
}

/* Display contents of a edge object by accepting the edge pointer as a parameter */
void display_edge(struct edge *edge_input) {

    struct node *temp1 = edge_input->src;
    struct node *temp2 = edge_input->trg;
    printf("\nEdge (index, label, src, trg)\n\n");
    printf("Edge: (%d,%s,%s,%s) \n", edge_input->edge_id, edge_input->edge_label, temp1->node_label, temp2->no
   printf("\n");
}



/* Main script that accepts a user input file and generates all of the Node Objects and Property (Prop) Object
void read_nodes() {
   int line_count;
   char buffer[1024];
   char *line;
   char fileparam[80];
   scanf("%s", fileparam);
   FILE *fstream = fopen(fileparam,"r");
   if(fstream == NULL)
   {
      printf("\n file opening failed ");
      read_nodes();
   }

   int i = 0;
   line_count = 0;
   line = fgets(buffer,sizeof(buffer),fstream);
   printf("first line: %s\n", line);
   prop_names[i] = strtok(line,",");
   while(prop_names[i]!=NULL){
      prop_names[++i] = strtok(NULL,",");
   }
```

Because I wrote my own function to concatenate and create the strings that would supply the JSON I needed to create an append function that uses some built in functions in C.

@ip-172-31-25-234: ~/plt_source/plt_c-graph/sample

```
/* Increases the size of the char* pointer to allocate the new item being added */
char* append(char* str1, char* str2) {
   char * str3 = (char *) malloc(1 + strlen(str1)+ strlen(str2) );
   strcpy(str3, str1);
   strcat(str3, str2);
   return str3;
}

/* Display full contents of a graph - iterating and accessing all elements in node, edge, and property
void display() {
   int i = 0;
   int n_cnt = 0;
   int e_cnt = 0;

   for(i = 0; i<1000; i++) {
      if(nodeIndex[i] != NULL)
         n_cnt++;
   }

   printf("%i \nNode (index, node id)\n\n", n_cnt);
   for(i = 0; i < n_cnt; i++) {
     printf("Node: (%d,%s) \n", nodeIndex[i]->node_id, nodeIndex[i]->node_label);
   }
   printf("\n");

   for(i = 0; i<1000; i++) {
      if(edgeIndex[i] != NULL)
         e_cnt++;
   }

   printf("Graph Summary - Edges: %i \nEdge (index, label, src, trg)\n\n", e_cnt);
   for(i = 0; i < e_cnt; i++) {
     struct node *temp1 = edgeIndex[i]->src;
     struct node *temp2 = edgeIndex[i]->trg;
     printf("Edge: (%d,%s,%s,%s) \n", i, edgeIndex[i]->edge_label, temp1->node_label, temp2->node_label
   }
   printf("\n");

}

/* Converts the contents of the current graph into a JSON readible format for export */
void display_JSON() {
   int i = 0;
   int f = 0;
   int n_cnt = 0;
   int e_cnt = 0;
   char* output;
   char* tab = "     ";
   char* ttab = "                    ";
```

The displayJSON function will dereference all of the objects that exist in the graph and output them to the user in a standard JSON format. This can be piped to a file.

@ip-172-31-25-234: ~/plt_source/plt_c-graph/sample

```
/* Converts the contents of the current graph into a JSON readible format for export */
void display_JSON() {
   int i = 0;
   int f = 0;
   int n_cnt = 0;
   int e_cnt = 0;
   char* output;
   char* tab = "     ";
   char* ttab = "               ";

   char* start = "{\n\n";
   output = start;
   output = append(output, tab);
   output = append(output, "nodes : [\n");
   output = append(output, ttab);

   for(i = 0; i<1000; i++) {
      if(nodeIndex[i] != NULL)
         n_cnt++;
   }

   for(i = 0; i < n_cnt; i++) {
     char* name = nodeIndex[i]->node_label;
     output = append(output, name);
     output = append(output, " : { ");
     int prop_cnt = nodeIndex[i]->prop_cnt;
     int g;
     for(g = 0; g < prop_cnt-1; g++) {
         struct prop *temp= nodeIndex[i]->property[g];
         char* p = temp->property;
         output = append(output, p);
         output = append(output, " : ");
         char* v = temp->value;
         output = append(output, v);
         output = append(output, ", ");
     }
     struct prop *temp= nodeIndex[i]->property[prop_cnt-1];
     char* p = temp->property;
     output = append(output, p);
     output = append(output, " : ");
     char* v = temp->value;
     output = append(output, v);
     output = append(output, " },\n");
     output = append(output, ttab);
   }
   output = append(output, "\n");
   output = append(output, tab);
   output = append(output, "        ],\n");
   output = append(output, "\n\n");
   output = append(output, tab);
```

Appendix B: C+ Compiler Code

c_includes.c

```
/* Here we simply included all the needed C standard libraries */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main()
{
    printf("Including stdio, stdlib, string\n");
    return 0;
}
```

printbig.c

```
/*
 *  A function illustrating how to link C code to code generated from LLVM
 */

#include <stdio.h>
#include <stdlib.h>

/*
 * Font information: one byte per row, 8 rows per character
 * In order, space, 0-9, A-Z
 */
static const char font[] = {
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x1c, 0x3e, 0x61, 0x41, 0x43, 0x3e, 0x1c, 0x00,
  0x00, 0x40, 0x42, 0x7f, 0x7f, 0x40, 0x40, 0x00,
  0x62, 0x73, 0x79, 0x59, 0x5d, 0x4f, 0x46, 0x00,
  0x20, 0x61, 0x49, 0x4d, 0x4f, 0x7b, 0x31, 0x00,
  0x18, 0x1c, 0x16, 0x13, 0x7f, 0x7f, 0x10, 0x00,
  0x27, 0x67, 0x45, 0x45, 0x45, 0x7d, 0x38, 0x00,
  0x3c, 0x7e, 0x4b, 0x49, 0x49, 0x79, 0x30, 0x00,
  0x03, 0x03, 0x71, 0x79, 0x0d, 0x07, 0x03, 0x00,
  0x36, 0x4f, 0x4d, 0x59, 0x59, 0x76, 0x30, 0x00,
  0x06, 0x4f, 0x49, 0x49, 0x69, 0x3f, 0x1e, 0x00,
  0x7c, 0x7e, 0x13, 0x11, 0x13, 0x7e, 0x7c, 0x00,
  0x7f, 0x7f, 0x49, 0x49, 0x49, 0x7f, 0x36, 0x00,
  0x1c, 0x3e, 0x63, 0x41, 0x41, 0x63, 0x22, 0x00,
  0x7f, 0x7f, 0x41, 0x41, 0x63, 0x3e, 0x1c, 0x00,
  0x00, 0x7f, 0x7f, 0x49, 0x49, 0x49, 0x41, 0x00,
  0x7f, 0x7f, 0x09, 0x09, 0x09, 0x09, 0x01, 0x00,
  0x1c, 0x3e, 0x63, 0x41, 0x49, 0x79, 0x79, 0x00,
  0x7f, 0x7f, 0x08, 0x08, 0x08, 0x7f, 0x7f, 0x00,
  0x00, 0x41, 0x41, 0x7f, 0x7f, 0x41, 0x41, 0x00,
  0x20, 0x60, 0x40, 0x40, 0x40, 0x7f, 0x3f, 0x00,
  0x7f, 0x7f, 0x18, 0x3c, 0x76, 0x63, 0x41, 0x00,
  0x00, 0x7f, 0x7f, 0x40, 0x40, 0x40, 0x40, 0x00,
  0x7f, 0x7f, 0x0e, 0x1c, 0x0e, 0x7f, 0x7f, 0x00,
  0x7f, 0x7f, 0x0e, 0x1c, 0x38, 0x7f, 0x7f, 0x00,
  0x3e, 0x7f, 0x41, 0x41, 0x41, 0x7f, 0x3e, 0x00,
  0x7f, 0x7f, 0x11, 0x11, 0x11, 0x1f, 0x0e, 0x00,
  0x3e, 0x7f, 0x41, 0x51, 0x71, 0x3f, 0x5e, 0x00,
  0x7f, 0x7f, 0x11, 0x31, 0x79, 0x6f, 0x4e, 0x00,
  0x26, 0x6f, 0x49, 0x49, 0x4b, 0x7a, 0x30, 0x00,
  0x00, 0x01, 0x01, 0x7f, 0x7f, 0x01, 0x01, 0x00,
  0x3f, 0x7f, 0x40, 0x40, 0x40, 0x7f, 0x3f, 0x00,
  0x0f, 0x1f, 0x38, 0x70, 0x38, 0x1f, 0x0f, 0x00,
  0x1f, 0x7f, 0x38, 0x1c, 0x38, 0x7f, 0x1f, 0x00,
  0x63, 0x77, 0x3e, 0x1c, 0x3e, 0x77, 0x63, 0x00,
  0x00, 0x03, 0x0f, 0x78, 0x78, 0x0f, 0x03, 0x00,
  0x61, 0x71, 0x79, 0x5d, 0x4f, 0x47, 0x43, 0x00
};
```

```
void printbig(int c)
{
  int index = 0;
  int col, data;
  if (c >= '0' && c <= '9') index = 8 + (c - '0') * 8;
  else if (c >= 'A' && c <= 'Z') index = 88 + (c - 'A') * 8;
  do {
    data = font[index++];
    for (col = 0 ; col < 8 ; data <<= 1, col++) {
      char d = data & 0x80 ? 'X' : ' ';
      putchar(d); putchar(d);
    }
    putchar('\n');
  } while (index & 0x7);
}


#ifdef BUILD_TEST
int main()
{
  char s[] = "HELLO WORLD09AZ";
  char *c;
  for ( c = s ; *c ; c++) printbig(*c);
}
#endif
```

cplus.ml

```
(* Top-level of the CPlus compiler: scan & parse the input,
   check the resulting AST, generate LLVM IR, and dump the module *)

type action = Ast | LLVM_IR | Compile

let _ =
  let action = if Array.length Sys.argv > 1 then
    List.assoc Sys.argv.(1) [ ("-a", Ast);      (* Print the AST only *)
                              ("-l", LLVM_IR);  (* Generate LLVM, don't check *)
                              ("-c", Compile) ] (* Generate, check LLVM IR *)
  else Compile in
  let lexbuf = Lexing.from_channel stdin in
  let ast = Parser.program Scanner.token lexbuf in
  Semant.check ast;
  match action with
    Ast -> print_string (Ast.string_of_program ast)
  | LLVM_IR -> print_string (Llvm.string_of_llmodule (Codegen.translate ast))
  | Compile -> let m = Codegen.translate ast in
    Llvm_analysis.assert_valid_module m;
    print_string (Llvm.string_of_llmodule m)
```

ast.ml

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

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

type uop = Neg | Not

type aop = Asn | ModAsn

(* TODO: Remove String, replace with Char, do strings with Charptr *)
type typ = Int | Size_t | Bool | Char | String | Void | Struct of string | Pointer of typ
(* Array of typ * int *)

type expr =
    Literal of int
  | SizeLit of int64
  | StringLit of string
  | CharLit of char
  | BoolLit of bool
```

```
  | Id of string
  | Binop of expr * op * expr
  | Unop of uop * expr
  | Assign of expr * aop * expr
  | Call of expr * expr list
  | ArrayAccess of expr * expr
  | StructAccess of expr * string
  | StructPointerAccess of expr * string
  | BuildArray of string * expr
  | BuiltInCall of string * expr list
  | Cast of typ * expr
  | Address of expr
  | Dereference of expr
  | Sizeof of typ
  | Nullexpr
  | Noexpr

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

(* typ - type
   string - name
   expr - value
   bool - flag for delaying evaluation *)
type bind = typ * string * expr * bool

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

type struct_decl = {
    members: bind list;
    struct_name: string;
  }

type program = {
    globals: bind list;
    functions: func_decl list;
    structs: struct_decl list;
}

(* flag - is this variable lazy?
   eval - have we yet evaluated it?
   expr - the expr to evaluate *)
type lazy_record = {
  name : string;
  flag : bool;
  mutable eval : bool;
  expr : expr;
}

(* Pretty-printing functions *)
(* TODO: add pretty print for strings, pointers, structs *)

let rec string_of_typ = function
    Int -> "int"
  | Size_t -> "size_t"
  | String -> "string"
  | Bool -> "bool"
  | Char -> "char"
  | Void -> "void"
  | Struct(id) -> id
  | Pointer(t) -> string_of_typ(t) ^ "*"

let string_of_op = function
    Add -> "+"
  | Sub -> "-"
```

```
  | Mult -> "*"
  | Div -> "/"
  | Equal -> "=="
  | Neq -> "!="
  | Less -> "<"
  | Leq -> "<="
  | Greater -> ">"
  | Geq -> ">="
  | And -> "&&"
  | Or -> "||"
  | Mod -> "%"

let string_of_uop = function
    Neg -> "-"
  | Not -> "!"

let string_of_aop = function
    Asn -> "="
  | ModAsn -> "%="

let rec string_of_expr = function
    Literal(l) -> string_of_int l
  | SizeLit(n) -> Int64.to_string n
  | StringLit(s) -> s
  | CharLit(c) -> String.make 1 c
  | BoolLit(true) -> "true"
  | BoolLit(false) -> "false"
  | 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, o, e) -> string_of_expr v ^ string_of_aop o ^ string_of_expr e
(*  | DerefAssign(p, e) -> p ^ " = " ^ string_of_expr e *)
  | BuiltInCall(f, el) ->
    f ^ "(" ^ String.concat ", " (List.map string_of_expr el) ^ ")"
  | Call(f, el) ->
    (string_of_expr f) ^ "(" ^ String.concat ", " (List.map string_of_expr el) ^ ")"
  | ArrayAccess(a, i) ->
    (string_of_expr a) ^ "[" ^ string_of_expr i ^ "]"
  | StructAccess(s, n) ->
    (string_of_expr s) ^ "." ^ n
  | StructPointerAccess(s, n) ->
    (string_of_expr s) ^ "->" ^ n
  | BuildArray(a, n) ->
    a ^ "[" ^ string_of_expr n ^ "]"
  | Cast(t, e) -> "<" ^ (string_of_typ t) ^ "> " ^ string_of_expr e
  | Sizeof(t) -> "sizeof(" ^ (string_of_typ t) ^ ")"
  | Address(v) -> "&" ^ string_of_expr v
  | Dereference(p) -> "*" ^ string_of_expr p
  | Noexpr -> ""
  | Nullexpr -> "NULL"

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

let string_of_vdecl (t, id, value, _) = string_of_typ t ^ " " ^ id ^ " " ^ (string_of_expr value) ^
";\n"

let string_of_fdecl fdecl =
  string_of_typ fdecl.typ ^ " " ^
  fdecl.fname ^ "(" ^ String.concat ", " (List.map (fun (_,x,_,_) -> x) fdecl.formals) ^
  ")\n{\n" ^
  String.concat "" (List.map string_of_vdecl fdecl.locals) ^
  String.concat "" (List.map string_of_stmt fdecl.body) ^
  "}\n"
```

```
let string_of_sdecl sdecl =
  "struct" ^ sdecl.struct_name ^ String.concat "{\n" (List.map string_of_vdecl sdecl.members) ^ "\n}\n"

let string_of_program prg =
  String.concat "" (List.map string_of_vdecl prg.globals) ^ "\n" ^
  String.concat "\n" (List.map string_of_fdecl prg.functions) ^
  String.concat "\n" (List.map string_of_sdecl prg.structs)
```

scanner.mll

```
(* Ocamllex scanner for CPlus*)

{ open Parser }

rule token = parse
  [' ' '\t' '\r' '\n'] { token lexbuf } (* Whitespace *)
| "/*"     { comment lexbuf }           (* Comments *)
| '('      { LPAREN }
| ')'      { RPAREN }
| '{'      { LBRACE }
| '}'      { RBRACE }
| '['      { LSQUARE }
| ']'      { RSQUARE }
| ';'      { SEMI }
| ','      { COMMA }
| '+'      { PLUS }
| '-'      { MINUS }
| '*'      { TIMES }
| '/'      { DIVIDE }
| '%'      { MOD }
| '='      { ASSIGN }
| '"'      { read_string (Buffer.create 17) lexbuf }
| '&'      { AMP }
| '.'      { DOT }
| "->"     { ARROW }
| "%="     { MOD_ASSIGN }
| "++"     { INC }
| "--"     { DEC }
| "=="     { EQ }
| "!="     { NEQ }
| '<'      { LT }
| "<="     { LEQ }
| ">"      { GT }
| ">="     { GEQ }
| "&&"     { AND }
| "||"     { OR }
| "!"      { NOT }
| "if"     { IF }
| "else"   { ELSE }
| "for"    { FOR }
| "while"  { WHILE }
| "return" { RETURN }
| "int"    { INT }
| "char"    { CHAR }
| "size_t" { SIZE_T }
| "string" { STRING }
| "char"   { CHAR }
| "bool"   { BOOL }
| "void"   { VOID }
| "true"   { TRUE }
| "false"  { FALSE }
| "sizeof" { SIZEOF }
| "struct" { STRUCT }
| "NULL"   { NULL }
(* BUILT IN FUNCTIONS *)
| "printf" as n { PRINTF(n) }
| "atoi" as n { ATOI(n) }
| "strdup" as n { STRDUP(n) }
| "printb" as n { PRINTB(n) }
| "print" as n  { PRINT(n) }
| "printbig" as n { PRINTBIG(n) }
| "malloc" as n { MALLOC(n) }
```

```
| "free" as n { FREE(n) }
(* LITERALS *)
| ['0'-'9']+ as lxm { LITERAL(int_of_string lxm) }
| ['a'-'z']['a'-'z' 'A'-'Z' '0'-'9' '_']* as lxm { ID(lxm) }
| ['A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']* as structLit { STRUCT_ID(structLit) }
| eof { EOF }
| ['\x00'-'\x7F'] as chr { CHARLIT(chr) }
| _ as char { raise (Failure("illegal character " ^ Char.escaped char)) }

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

(*
This "read_string" function was borrowed directly from this link
https://realworldocaml.org/v1/en/html/parsing-with-ocamllex-and-menhir.html
*)
and read_string buf =
  parse
  | '"'       { STRINGLIT (Buffer.contents buf) }
  | '\\' '/'  { Buffer.add_char buf '/'; read_string buf lexbuf }
  | '\\' '\\' { Buffer.add_char buf '\\'; read_string buf lexbuf }
  | '\\' 'b'  { Buffer.add_char buf '\b'; read_string buf lexbuf }
  | '\\' 'f'  { Buffer.add_char buf '\012'; read_string buf lexbuf }
  | '\\' 'n'  { Buffer.add_char buf '\n'; read_string buf lexbuf }
  | '\\' 'r'  { Buffer.add_char buf '\r'; read_string buf lexbuf }
  | '\\' 't'  { Buffer.add_char buf '\t'; read_string buf lexbuf }
  | [^ '"' '\\']+
    { Buffer.add_string buf (Lexing.lexeme lexbuf);
      read_string buf lexbuf
    }
  | _ { raise (Failure ("Illegal string character: " ^ Lexing.lexeme lexbuf)) }
  | eof { raise (Failure ("String is not terminated")) }
```

parser.mly

```
/* Ocamlyacc parser for CPlus */

%{
open Ast
%}

%token SEMI LPAREN RPAREN LBRACE RBRACE LSQUARE RSQUARE COMMA
%token PLUS MINUS TIMES DIVIDE MOD ASSIGN MOD_ASSIGN NOT
%token DOT ARROW
%token AMP INC DEC
%token EQ NEQ LT LEQ GT GEQ TRUE FALSE AND OR
%token RETURN IF ELSE FOR WHILE SIZEOF
%token INT SIZE_T STRING BOOL CHAR VOID STRUCT NULL
%token <string> PRINTF PRINT PRINTB PRINTBIG MALLOC FREE ATOI STRDUP
%token <int> LITERAL
%token <string> STRINGLIT
%token <char> CHARLIT
%token <string> ID
%token <string> STRUCT_ID
%token EOF

/* Opeartor Precedence closesly modeled on C */
%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 AMP
%right CAST
%right NOT NEG*/
/* %left DOT */
```

```
%start program
%type <Ast.program> program

%%

program:
  decls EOF { $1 }

decls:
   /* nothing */ { {globals=[]; functions=[]; structs=[]} }
 | decls declaration { {globals = ($2 @ $1.globals); functions = $1.functions; structs = $1.structs} }
 | decls func_decl { {globals = $1.globals; functions = ($2 :: $1.functions); structs = $1.structs} }
 | decls struct_decl { {globals = $1.globals; functions = $1.functions; structs = ($2 :: $1.structs)} }

func_decl:
   typ ID LPAREN formals_opt RPAREN LBRACE declaration_list stmt_list RBRACE
     { { typ = $1;
                fname = $2;
                formals = $4;
                locals = List.rev $7;
                body = List.rev $8 } }

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

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

struct_decl:
    STRUCT STRUCT_ID LBRACE declaration_list RBRACE SEMI
    { { members = $4;
        struct_name = $2; } }

typ:
    INT { Int }
  | SIZE_T { Size_t }
  | STRING { String }
  | BOOL { Bool }
  | CHAR { Char }
  | VOID { Void }
  | STRUCT_ID { Struct($1) }
  | typ TIMES { Pointer($1) }

declaration_list:
    /* nothing */    { [] }
  | declaration_list declaration { $2 @ $1 }

/* should be able to declare multiple vars of one type on the same line */
declaration: /* is a list */
   typ init_declarator_list SEMI {
     let append_typ t (a,b,c) = (match b with
          BuildArray(_,_) -> (Pointer t, a, b, c)
       |  _ -> (t, a, b, c)
       ) in
     List.map (append_typ $1) $2 }

/* id to value binding for a declarator */
init_declarator:
    ID { ($1, Noexpr, false) }
  | ID LSQUARE add_expr RSQUARE { ($1, BuildArray($1, $3), true) }
  | ID ASSIGN expr { ($1, $3, true) }

/* list of name-value tuples for declarators */
init_declarator_list:
    init_declarator { [$1] }
  | init_declarator_list COMMA init_declarator { $3 :: $1 }

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

stmt:
    expr SEMI { Expr $1 }
  | selection_stmt { $1 }
```

```
  | iteration_stmt { $1 }
  | RETURN SEMI { Return Noexpr }
  | RETURN expr SEMI { Return $2 }
  | LBRACE stmt_list RBRACE { Block(List.rev $2) }

selection_stmt:
    IF LPAREN expr RPAREN stmt %prec NOELSE { If($3, $5, Block([])) }
    /*IF LPAREN expr RPAREN stmt              { If($3, $5, Block([])) }*/
  | IF LPAREN expr RPAREN stmt ELSE stmt    { If($3, $5, $7) }

iteration_stmt:
    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:
    assignment_expr  { $1 }

assignment_operator:
    ASSIGN                              { Asn }
  | MOD_ASSIGN                          { ModAsn }

assignment_expr:
    logical_or_expr                     { $1 }
  | postfix_expr assignment_operator expr { Assign($1, $2, $3) }

logical_or_expr:
    logical_and_expr                    { $1 }
  | logical_or_expr OR logical_and_expr   { Binop($1, Or, $3) }

logical_and_expr:
    equality_expr                       { $1 }
  | logical_and_expr AND equality_expr    { Binop($1, And, $3) }

equality_expr:
    relational_expr                     { $1 }
  | equality_expr EQ relational_expr      { Binop($1, Equal, $3) }
  | equality_expr NEQ relational_expr     { Binop($1, Neq, $3) }

relational_expr:
    add_expr                            { $1 }
  | relational_expr LT add_expr           { Binop($1, Less, $3) }
  | relational_expr GT add_expr           { Binop($1, Greater, $3) }
  | relational_expr LEQ add_expr          { Binop($1, Leq, $3) }
  | relational_expr GEQ add_expr          { Binop($1, Geq, $3) }

cast_expr:
    unary_expr                          { $1 }
  | LPAREN typ RPAREN cast_expr           { Cast($2, $4) }

unary_operator:
    NOT                                  { Not }
  | MINUS                                { Neg }

unary_expr:
    postfix_expr                        { $1 }
  | unary_operator postfix_expr           { Unop($1, $2)  }
  | INC postfix_expr                      { Assign($2,Asn,Binop($2, Add, Literal(1))) }
  | DEC postfix_expr                      { Assign($2,Asn,Binop($2, Sub, Literal(1))) }

postfix_expr:
    built_in_expr                               { $1 }
  | postfix_expr INC                            { Assign($1,Asn,Binop($1, Add, Literal(1))) }
  | postfix_expr DEC                            { Assign($1,Asn,Binop($1, Sub, Literal(1))) }
  | postfix_expr LPAREN actuals_list_opt RPAREN  { Call($1, $3) }
  | postfix_expr LSQUARE postfix_expr RSQUARE    { ArrayAccess($1, $3) }
  | postfix_expr DOT ID                         { StructAccess($1, $3) }
  | postfix_expr ARROW ID                       { StructPointerAccess($1, $3) }

built_in_expr:
    primary_expr                                { $1 }
  | PRINTF LPAREN actuals_list_opt RPAREN        { BuiltInCall($1, $3) }
```

```
  | PRINT LPAREN actuals_list_opt RPAREN         { BuiltInCall($1, $3) }
  | PRINTB  LPAREN actuals_list_opt RPAREN       { BuiltInCall($1, $3) }
  | PRINTBIG LPAREN actuals_list_opt RPAREN      { BuiltInCall($1, $3) }
  | MALLOC LPAREN actuals_list_opt RPAREN        { BuiltInCall($1, $3) }
  | FREE LPAREN actuals_list_opt RPAREN          { BuiltInCall($1, $3) }
  | ATOI LPAREN actuals_list_opt RPAREN          { BuiltInCall($1, $3) }
  | STRDUP LPAREN actuals_list_opt RPAREN        { BuiltInCall($1, $3) }

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

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

add_expr:
    mult_expr                                 { $1 }
  | add_expr PLUS mult_expr                   { Binop($1, Add, $3) }
  | add_expr MINUS mult_expr                  { Binop($1, Sub, $3) }

mult_expr:
    cast_expr                                 { $1 }
  | mult_expr TIMES cast_expr                 { Binop($1, Mult, $3) }
  | mult_expr DIVIDE cast_expr                { Binop($1, Div, $3) }
  | mult_expr MOD cast_expr                   { Binop($1, Mod, $3) }

primary_expr:
    LPAREN expr RPAREN                        { $2 }
  | LITERAL                                   { Literal($1) }
  | STRINGLIT                                 { StringLit($1) }
  | CHARLIT                                   { CharLit($1) }
  | TRUE                                      { BoolLit(true) }
  | FALSE                                     { BoolLit(false) }
  | ID                                        { Id($1) } /* Identifier */
  | AMP primary_expr                          { Address($2) }
  | TIMES primary_expr                        { Dereference($2) }
  | SIZEOF LPAREN typ RPAREN                  { Sizeof($3) }
  | NULL                                      { Nullexpr }
```

semant.ml

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

open Ast

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

  (* Reveal the internal type of a pointer type, otherwise just return input *)
  let strip p = match p with
      Pointer(typ) -> typ
```

```
  | _ -> p
in

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

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

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

report_duplicate (fun n -> "duplicate global " ^ n) (List.map (fun (_,x,_,_) -> x) program.globals);

(**** Checking program.functions ****)

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

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

(* Function declaration for a named function *)
let built_in_decls =  StringMap.add "print"
   { typ = Void; fname = "print"; formals = [(Int, "x", Noexpr, false)];
     locals = []; body = [] } (StringMap.add "printb"
   { typ = Void; fname = "printb"; formals = [(Bool, "x", Noexpr, false)];
     locals = []; body = [] } (StringMap.add "printf"
   { typ = Void; fname = "printf"; formals = [(String, "x", Noexpr, false)];
     locals = []; body = [] } (StringMap.add "malloc"
   { typ = Pointer(Void); fname = "malloc"; formals = [(Size_t, "x", Noexpr, false)];
     locals = []; body = [] } (StringMap.add "free"
   { typ = Void; fname = "free"; formals = [(Pointer(Void), "x", Noexpr, false)];
     locals = []; body = [] } (StringMap.add "atoi"
   { typ = Pointer(Char); fname = "atoi"; formals = [(Pointer(Char), "x", Noexpr, false)];
     locals = []; body = []; } (StringMap.add "strdup"
   { typ = Pointer(Char); fname = "strdup"; formals = [(Pointer(Char), "x", Noexpr, false)];
     locals = []; body = []; } (StringMap.singleton "printbig"
   { typ = Void; fname = "printbig"; formals = [(Int, "x", Noexpr, false)];
     locals = []; body = [] } )))))))
 in

let struct_decls = List.fold_left (fun m sd -> StringMap.add sd.struct_name sd m)
                     StringMap.empty program.structs
in

let struct_decl s = try StringMap.find s struct_decls
    with Not_found -> raise (Failure ("SEMANT: unrecognized struct " ^ s))
in

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

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

let _ = function_decl "main" in (* Ensure "main" is defined *)

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 (fun (_,x,_,_) -> x) 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 (fun (_,x,_,_) -> x) 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.globals @ 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
        Literal _ -> Int
      | CharLit _ -> Char
      | SizeLit _ -> Size_t
      | StringLit _ -> String
      | BoolLit _ -> Bool
      | Id s -> type_of_identifier s
      | Sizeof _ -> Size_t
      | Binop(e1, op, e2) as e ->
        let t1 = expr e1 and t2 = expr e2 in
        let err = (Failure ("illegal binary operator " ^
                            string_of_typ t1 ^ " " ^ string_of_op op ^ " " ^
                            string_of_typ t2 ^ " in " ^ string_of_expr e)) in
                (match op with
             Add | Sub when t2 = Int ->
                (match t1 with
                   Pointer(t) -> Pointer(t)
                 | Int -> Int
                 | _ -> raise err
              )
            | Mult | Div | Mod  when t1 = Int && t2 = Int -> Int
            | Equal | Neq when ((t1 = t2) || (e2 = Nullexpr)) -> Bool
                  | Less | Leq | Greater | Geq when t1 = Int && t2 = Int -> Bool
                  | And | Or when t1 = Bool && t2 = Bool -> Bool
            | _ -> raise err
            )
      | Unop(op, e) as ex -> let t = expr e in
                 (match op with
                   Neg when t = Int -> Int
              | Not when t = Bool -> Bool
              | _ -> raise (Failure ("illegal unary operator " ^ string_of_uop op ^
                string_of_typ t ^ " in " ^ string_of_expr ex)))
      | Address(e) -> let t = expr e in (match t with
            Int -> Pointer(Int)
          | Bool -> Pointer(Bool)
          | Pointer(Int) -> Pointer(Pointer(Int))
          | Pointer(Bool) -> Pointer(Pointer(Bool))
          | _ -> raise (Failure ("trying to get address of expr " ^ string_of_expr e ^ " but it has
illegal type " ^ string_of_typ t))
        )
      | Dereference(e) -> let t = expr e in (match t with
            Pointer(Int) -> Int
          | Pointer(Bool) -> Bool
          | Pointer(Char) -> Char
          | Pointer(Pointer(t)) -> if t = Char then String else Pointer(t)
          | Pointer(Struct(n)) -> Struct(n)
          | _ -> raise (Failure ("trying to get dereference of non-pointer " ^ string_of_expr e ^ " of
type " ^ string_of_typ t))
        )
      | Noexpr -> Void
      | Nullexpr -> Void
      | Assign(lhs, op, rhs) as ex ->
        let rt = if (expr rhs) = Pointer(Char) then String else (expr rhs)
        and lt = if (expr lhs) = Pointer(Char) then String else (expr lhs) in
        let err a b = (Failure ("illegal assignment " ^ string_of_typ a ^ string_of_aop op ^
string_of_typ b ^ " in " ^ string_of_expr ex)) in
         (match op with
            Asn -> (match lhs with
                ArrayAccess(_,_) -> check_assign (strip lt) rt (err (strip lt) rt)
              | _ -> (match rhs with
                    ArrayAccess(_,_) -> check_assign lt (strip rt) (err lt (strip rt))
                  | _ -> check_assign lt rt (err lt rt)
                )
            )
          | ModAsn -> if (lt = Bool) || (rt = Bool) then raise (err lt rt) else check_assign lt rt (err
lt rt))
```

```
      | BuiltInCall(fname, actuals) as bcall ->
        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 bcall))
        else  (match fname with
              "free" -> (* HACK: For the built-in function "free" we need to coerce the pointers to
void* *)
                  (match expr (List.hd actuals) with
                     Pointer(_) -> Void
                   | t -> raise (Failure ("built-in function free(): attempting to free non-pointer " ^
string_of_typ t ^ "."))
                  ) (* end match expr *)
            | "malloc" -> let err = (Failure ("SEMANT: improper arguments to malloc-call " ^
string_of_expr bcall ^ ".")) in
              (match (List.hd actuals) with
                 Binop(e1, Mult, e2) -> (match (expr e1, expr e2) with
                     ((Int, Size_t) | (Size_t, Int)) -> Pointer(Void)
                   | _ -> raise err) (* FIXME: need to extract actual pointer type *)
               | Sizeof(t) -> Pointer(t)
               | _ -> raise err) (* end match expr *)
            | "atoi" as func ->
              let input = expr (List.hd actuals) in
              if input = Pointer(Char) then Int else raise (Failure ("SEMANT: improper input type " ^
string_of_typ input ^ " to " ^ func))
            | "strdup" as func ->
              let input = expr (List.hd actuals) in
              if input = Pointer(Char) then input else raise (Failure ("SEMANT: improper input type " ^
string_of_typ input ^ " to " ^ func))
        (* FIXME: Add ANY Semantic checking for the rest of the builtin functions *)
            | "print" -> Void
            | "printb" -> Void
            | "printf" -> Void
            | "printbig" -> Void
            | n -> raise (Failure("SEMANT: Illegal built-in name " ^ n ^ "."))
          ) (* end match fname *)
      | Call(fname, actuals) as call ->
        (* FIXME: A.string_of_expr will only work for non-compound functions, will break for function
pointers *)
         let fname' = string_of_expr fname in
         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
      | ArrayAccess(arr, _) ->
            (* This is C, we are fortunate not to have to do boundary checking *)
            (* FIXME: Add some actual semantic checking here *)
           expr arr
      | StructAccess(s, m) as sacc ->
        let sd = struct_decl (string_of_typ (strip (strip (expr s))))in
          let members = List.fold_left (fun m (t,n,_,_) -> StringMap.add n t m) StringMap.empty
sd.members in
          (* Iterate through the members of the struct; if name found, return its type, else fail *)
          (try StringMap.find m members
           with Not_found -> raise (Failure ("illegal member " ^ m ^ " of struct " ^ string_of_expr
sacc)))
      | StructPointerAccess(s, m) as spacc ->
        let sd = (match s with
            | ArrayAccess(a,_) -> struct_decl (string_of_typ (strip (expr (Dereference a))))
            | _ -> struct_decl (string_of_typ (strip (expr s)))
          ) in
          let members = List.fold_left (fun m (t,n,_,_) -> StringMap.add n t m) StringMap.empty
sd.members in
          (* Iterate through the members of the struct; if name found, return its type, else fail *)
          (try StringMap.find m members
           with Not_found -> raise (Failure ("illegal member " ^ m ^ " of struct " ^ string_of_expr
spacc)))
      | BuildArray(_,_) -> Pointer(Void)
```

```
      | Cast(t, e) -> ignore( expr e ) ; t
      (* | _ as ex -> raise (Failure ("unknown expression " ^ (string_of_expr ex) ^ ".")) *)

    in (* END expr CHECKER *)

    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.functions
```

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/

*)

open Ast

module L = Llvm
module A = Ast

module StringMap = Map.Make(String)

let translate program =
  let context = L.global_context () in
  let the_module = L.create_module context "CPlus"
  and i64_t  = L.i64_type  context
  and i32_t  = L.i32_type  context
  and i8_t   = L.i8_type   context
  and i1_t   = L.i1_type   context
  and str_t  = L.pointer_type (L.i8_type context)
  and void_t = L.void_type context
  and ptr_t t = L.pointer_type t
  and struct_t n = L.named_struct_type context n in

  let rec ltype_of_primitive = function
      A.Int -> i32_t
```

```
    | A.Size_t -> i64_t
    | A.String -> str_t
    | A.Bool -> i1_t
    | A.Char -> i8_t
    | A.Void -> void_t
    | A.Pointer t ->
      if (t == A.Void) then ptr_t i8_t else ptr_t (ltype_of_primitive t)
    | A.Struct n -> struct_t n (* unitialized struct type *)
  in (* END FUNCTION ltype_of_primitive *)

  let structs =
    let struct_decl m sdecl =
      let name = sdecl.A.struct_name
      and member_types = Array.of_list (List.map (fun (t,_,_,_) -> ltype_of_primitive t)
sdecl.A.members)
      in let stype = L.struct_type context member_types in
      StringMap.add name (stype, sdecl.A.members) m in
    List.fold_left struct_decl StringMap.empty program.structs
  in

  let struct_lookup n = try StringMap.find n structs
    with Not_found -> raise (Failure ("CODEGEN: struct " ^ n ^ " not found")) in
  (* END structs *)

  let rec ltype_of_typ = function
      A.Struct n -> fst (struct_lookup n) (* reveals "stype" *)
    | A.Pointer t ->
       if (t == A.Void) then ptr_t i8_t else ptr_t (ltype_of_typ t)
    | t -> ltype_of_primitive t
  in

  (* sometimes we need to get at the subtype *)
  let strip = function
      A.Pointer t ->  t
    | t -> t
  in

  let initialize_value typ v = match (typ,v) with
      (A.Int, A.Literal i) -> L.const_int i32_t i
    | (A.Size_t, A.Literal i) -> L.const_int i64_t i
    | (A.Size_t, A.SizeLit s) -> L.const_of_int64 i64_t s true
    | (A.String, A.StringLit s) -> L.const_string context (s^"\x00")
    | (A.Bool, A.BoolLit b) -> L.const_int i1_t (if b then 1 else 0)
    | (A.Char, A.CharLit c) -> L.const_int i8_t (Char.code c)
    | (_,A.Noexpr) -> L.const_int i32_t 0
    | (A.Pointer(t), _) -> L.const_pointer_null (ltype_of_typ t)
    | _ -> raise (Failure("CODEGEN: attempting to initialize global to non-primitive type"))
    (* | _ -> L.const_int (ltype_of_typ typ) 0 *)
  in

  (* Declare each global variable; remember its value in a map *)
  (* FIXME: currently cannot enable the initialization of global pointers *)
  let global_vars =
    let global_var map (typ, name, value, _) =
      (* let init = L.const_int (ltype_of_typ typ) 0 *)
      let init = initialize_value typ value
      in StringMap.add name (L.define_global name init the_module) map in
    List.fold_left global_var StringMap.empty program.globals
  in

  (* as with global_vars, stores their types in a separate map *)
  let global_var_types =
    let global_var map (typ, name, _, _) =
      StringMap.add name typ map in
    List.fold_left global_var StringMap.empty program.globals
  in

  (* Make also a map of names to whether we need to lazily eval their intial values *)
  (* let global_lazy_vars =
    let global_var m (_, n, v, f) =
      (* the eval field is set false once evaluated *)
      StringMap.add n { name = n; flag = f; eval = true; expr = v } m in
    List.fold_left global_var StringMap.empty program.globals
  in *)

  (* Declare printf(), which the print built-in function will call (only strings for now) *)
```

```
  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

  (* Declare atoi(str), which returns the int of char* string *)
  let atoi_t = L.function_type (ptr_t i8_t) [| (ptr_t i8_t) |] in
  let atoi_func = L.declare_function "atoi" atoi_t the_module in

  (* Declare strdup(str), which returns a char* with space for a copy of str *)
  let strdup_t = L.function_type (ptr_t i8_t) [| (ptr_t i8_t) |] in
  let strdup_func = L.declare_function "strdup" strdup_t the_module in

  (* Declare the built-in printbig() function *)
  let printbig_t = L.function_type i32_t [| i32_t |] in
  let printbig_func = L.declare_function "printbig" printbig_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.A.fname
      and formal_types = Array.of_list (List.map (fun (t,_,_,_) -> ltype_of_typ t) fdecl.A.formals)
      in let ftype = L.function_type (ltype_of_typ fdecl.A.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.functions in

  (* Fill in the body of the given function *)
  let build_function_body fdecl =
    let (the_function, _) = StringMap.find fdecl.A.fname function_decls in
    let builder = L.builder_at_end context (L.entry_block the_function) in

    let int_format_str = L.build_global_stringptr "%d\n" "fmt" builder in
    let str_format_str = L.build_global_stringptr "%s\n" "fmt" 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 =

      (* m = map, t = type, n = name, v = value, p = parameters *)
      let add_formal m (t, n, v, _) p =
         L.set_value_name n p;
           let formal = L.build_alloca (ltype_of_typ t) n builder in
         (* ignore (L.build_store p formal builder); *)
         ignore(
           (* if v = A.Noexpr then (L.build_store p formal builder) else *)
           match (t,v) with
                  ((_,A.Noexpr) | (A.Pointer(Char),_)) ->
                    ignore(L.build_store p formal builder)
                | ((_,A.Literal _) | (_,A.SizeLit _ )| (_,A.StringLit _) | (_,A.BoolLit _)) ->
                    ignore(L.build_store (initialize_value t v) formal builder)
                | _ -> ignore(None) (* IDEA: can keep these things lazy, but eval them earlier than
when the ID is called!!! *)
           (* (L.build_store (initialize_value t v) formal builder) *)
         );

         StringMap.add n formal m (* meat of "add_formal" *)
      in (* end function def of "add_formal" *)

      let add_local m (t, n, v, _) =
        let local_var = L.build_alloca (ltype_of_typ t) n builder in
        (* ignore(if v != A.Noexpr then ignore(L.build_store (initialize_value t v) local_var
builder)); *)
        ignore(
          match (t,v) with
            (A.Pointer(Char),_) -> ignore(None)
          | (_,A.Literal _) | (_,A.SizeLit _) | (_,A.StringLit _)| (_,A.BoolLit _) ->
              ignore(L.build_store (initialize_value t v) local_var builder)
          | _ -> ignore(None) (* Do Nothing *)
        );
        StringMap.add n local_var m (* meat of function def of "add_local" *)
      in (* end function def of "add_local" *)

      (* fold in the list of formals using "add_formal" *)
      let formals = List.fold_left2 add_formal StringMap.empty fdecl.A.formals
          (Array.to_list (L.params the_function)) in

      (* fold in the list of locals using "add_local" *)
```

```
    List.fold_left add_local formals fdecl.A.locals
  in
  (* END DEFINITION of LOCAL_VARS *)

  (* As with local_vars, make a map of types for each name *)
  let local_var_types =

    (* m = map, t = type, n = name, v = value *)
    let add_item m (t, n, _, _) =
      StringMap.add n t m
    in (* end function def of "add_item" *)

    (* fold in the list of formals using "add_formal" *)
    let formals = List.fold_left add_item StringMap.empty fdecl.A.formals in

    (* fold in the list of locals using "add_local" *)
    List.fold_left add_item formals fdecl.A.locals
  in
  (* END DEFINITION of LOCAL_VARS *)

  (* Make also a map of names to whether we need to lazily eval their intial values *)
  let local_lazy_vars =
    let add_item m (_, n, v, f) =
      (* the eval field is set false once evaluated *)
      StringMap.add n { name = n; flag = f; eval = true; expr = v } m
    in
    let formals = List.fold_left add_item StringMap.empty fdecl.A.formals in
    List.fold_left add_item formals fdecl.A.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 -> raise (Failure("CODEGEN: can't find var " ^ n))
  in

  (* Return the type for a variable or formal argument *)
  let lookup_type n =
    (* let is_int s =
      try ignore (int_of_string s); true
        with _ -> false
    in
    if is_int n then A.Int else *)
      try StringMap.find n local_var_types
      with Not_found ->
        (* SUPER HACK: must not fail! So default to void *)
        try StringMap.find n global_var_types
            (* with Not_found -> raise (Failure("CODEGEN: can't find type of " ^ n)) *)
          with Not_found -> A.Void
  in

  (* Did we decide to evaluate this lazily? *)
  (* let lookup_lazy n = try StringMap.find n local_lazy_vars
    with Not_found -> try StringMap.find n global_lazy_vars
      with Not_found -> raise (Failure ("CODEGEN: can't find lazy var"))
  in *)

  let failsafe_flag = 0 in

  (* Construct code for a primary expression; return its value *)
  (* let rec primary_expr builder = function *)
  let rec primary_expr builder = function
      A.Literal i -> L.const_int i32_t i
    | A.SizeLit s -> L.const_of_int64 i64_t s true
    | A.StringLit s -> L.build_global_stringptr (s^"\x00") "strptr" builder
    | A.BoolLit b -> L.const_int i1_t (if b then 1 else 0)
    | A.CharLit c -> L.const_int i8_t (Char.code c)
    | A.Noexpr -> L.const_int i32_t 0
    | A.Nullexpr -> L.const_null (void_t)
    | A.Id s -> L.build_load (lookup s) s builder
    (* Address - just use the lookup function, which in fact will give us
       the 'stack-pointer' returned by alloca when this var was created *)
    | A.Address e -> lookup (A.string_of_expr e)
    | A.Dereference e -> L.build_load (primary_expr builder e) "deref" builder
    | A.Sizeof t ->
        L.size_of (ltype_of_typ t) (* returns a 64-bit (i64_t) integer *)
```

```
      | e -> (* for the case with the parentheses *)
        if failsafe_flag = 1 then
          raise (Failure("CODEGEN: expression reduction failed!!!"))
        else expr builder e

    (* Construct code for mult_expr; return its value *)
    and mult_expr builder = function
        A.Binop (e1, ((A.Mult | A.Div | A.Mod) as op), e2) ->
              let e1' = mult_expr builder e1
              and e2' = cast_expr builder e2 in
              (match op with
                       A.Mult -> L.build_mul
                     | A.Div -> L.build_sdiv
                     | A.Mod -> L.build_srem
                     | _ -> raise (Failure("CODEGEN: improper mult expr!!!"))
              ) e1' e2' "mult_expr" builder
      | e -> cast_expr builder e

    (* Construct code for add_expr; return its value *)
    and add_expr builder = function
        A.Binop (e1, ((A.Add | A.Sub) as op), e2) ->
            (* HACK: string_of_expr will not work for much!! *)
            let t1 = lookup_type (string_of_expr e1)
            and t2 = lookup_type (string_of_expr e2) in
            (* FIXME: only pointer addition works currently, not subtraction *)
              (match (op, t1, t2) with
                  (A.Add, A.Pointer(_), _) ->
                      L.build_gep (add_expr builder e1) [| (mult_expr builder e2) |] "add.ptr" builder
                | (A.Add, _, A.Pointer(_)) ->
                      L.build_gep (mult_expr builder e2) [| (add_expr builder e1) |] "add.ptr" builder
                | (A.Add, _, _) -> L.build_add (add_expr builder e1) (mult_expr builder e2) "add_expr"
builder
                | (A.Sub, _, _) -> L.build_sub (add_expr builder e1) (mult_expr builder e2) "sub_expr"
builder
                | _ -> raise (Failure("CODEGEN: improper add/sub expr!!!"))
              )
      | e -> mult_expr builder e

    (* Construct code for a built_in expression; return its value  *)
    and built_in_expr builder = function
        A.BuiltInCall (n, [e]) -> (match n with
            "printf" -> L.build_call printf_func [| str_format_str ; (expr builder e) |] "printf"
builder
          | "print" | "printb" -> L.build_call printf_func [| int_format_str ; (expr builder e) |]
"printf" builder
          | "printbig" -> L.build_call printbig_func [| (expr builder e) |] "printbig" builder
          (* | "malloc" -> let e' = primary_expr builder e in
            L.build_array_malloc (L.type_of e') "malloc" builder *)
          | "malloc" -> (match e with
               A.Binop(e1, A.Mult, e2) ->
                 let e1' = add_expr builder e1
                 and e2' = add_expr builder e2 in
                   if L.type_of e1' = i32_t then L.build_array_malloc (L.type_of e2') e1' "arr_malloc"
builder
                   else L.build_array_malloc (L.type_of e1') e2' "malloc" builder
             | _ -> let e' = primary_expr builder e in L.build_malloc (L.type_of e') "malloc" builder)
          | "free" -> L.build_free (expr builder e) builder (* this had better be a void* pointer! *)
          | "atoi" -> L.build_call atoi_func [| (expr builder e) |] "atoi" builder
          | "strdup" -> L.build_call strdup_func [| (expr builder e) |] "strdup" builder
          | _ -> raise (Failure("CODEGEN: improper built-in function call!!!"))
        ) (* end BuiltInCall match statement *)
      | A.BuildArray (a, n) ->
            let ptr = lookup_type a in
            let t = strip ptr
            and n' = primary_expr builder n in
            (* malloc returns in i8_t ptr *)
            let tmp = L.build_array_malloc (ltype_of_typ t) n' "ary.malloc" builder in
            (* need to cast it to a ptr_t *)
            L.build_bitcast tmp (ltype_of_typ ptr) "ary.cast" builder
      | e -> primary_expr builder e

    (* Construct code for a postfix expression; return its value *)
    and postfix_expr builder = function
        A.Call (f, act) ->
            (* FIXME: using string_of_expr for now, but need a comprehensive solution to compound LHS
*)
```

```
            let f' = A.string_of_expr f in
            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.A.typ with A.Void -> "" | _ -> f' ^ "_result") in
            L.build_call fdef (Array.of_list actuals) result builder
      | A.ArrayAccess (a, i) ->
          let n = L.build_zext_or_bitcast (postfix_expr builder i) i64_t "ary.cast" builder in
          (* SOLUTION: need to use i64_t array constants, still figuring out how to do this *)
          let idx = L.build_gep (expr builder a) [| n |] "ary.ptr" builder in
          L.build_load idx "ary.val" builder
      | A.StructAccess (s, m) ->
          let (stype, location) = (match s with
            | A.Id id -> (lookup_type id, lookup id)
            | A.Dereference sp ->
                (strip(lookup_type (A.string_of_expr sp)), expr builder s)
            | A.ArrayAccess (a, _) ->
              (strip(strip(lookup_type (A.string_of_expr a))) , expr builder s)
            | _ -> raise (Failure("CODEGEN: illegal struct-type " ^ A.string_of_expr s))
          ) in
            let members = snd (struct_lookup (A.string_of_typ stype)) in
              let rec get_idx n lst i = match lst with
                | [] -> raise (Failure("CODEGEN: id " ^ m ^ " is not a member of struct " ^
A.string_of_expr s))
                | hd::tl -> if (hd=n) then i else get_idx n tl (i+1)
              in let idx = (get_idx m (List.map (fun (_,nm,_,_) -> nm) members) 0) in
              let ptr = L.build_struct_gep location idx ("struct.ptr") builder in
          L.build_load ptr ("struct.val."^m) builder
      | A.StructPointerAccess (s, m) ->
        let (stype, location) = (match s with
            | ArrayAccess(a,_) ->
              (match lookup_type (A.string_of_expr a) with
               | Pointer(Pointer(t)) -> (t, expr builder s)
               | Pointer(t) -> (t, expr builder s)
               | _ -> raise (Failure("CODGEN: something wrong with arrays D:"))
              )
            | _ -> ((strip(lookup_type (A.string_of_expr s))), expr builder s)
          ) in
          (* let stype = (strip(lookup_type (A.string_of_expr s)))
          and location = expr builder s in *)
            let members = snd (struct_lookup (A.string_of_typ stype)) in
              let rec get_idx n lst i = match lst with
                | [] -> raise (Failure("CODEGEN: id " ^ m ^ " is not a member of struct " ^
A.string_of_expr s))
                | hd::tl -> if (hd=n) then i else get_idx n tl (i+1)
        in let idx = (get_idx m (List.map (fun (_,nm,_,_) -> nm) members) 0) in
        let ptr = L.build_struct_gep location idx ("struct.ptr") builder in
        L.build_load ptr ("struct.val."^m) builder
      | e -> built_in_expr builder e

    (* Construct code for a unary expression; return its value *)
    and unary_expr builder = function
        A.Unop(((A.Neg | A.Not) as op), e) ->
          let e' = postfix_expr builder e in
          (match op with
             A.Neg     -> L.build_neg e' "tmp" builder
           | A.Not     -> L.build_not e' "tmp" builder)
      | e -> postfix_expr builder e

    (* Construct code for casting the expression to a given type *)
    and cast_expr builder = function
        A.Cast (t, e) ->
        L.build_bitcast (cast_expr builder e) (ltype_of_typ t) "tcast" builder
      | e -> unary_expr builder e

    (* Construct code for a relational expression; return its value *)
    and relational_expr builder = function
        A.Binop(e1, ((A.Less|A.Greater|A.Leq|A.Geq) as op), e2) ->
          let e1' = relational_expr builder e1
          and e2' = add_expr builder e2 in
          (match op with
             A.Less -> L.build_icmp L.Icmp.Slt
           | A.Greater -> L.build_icmp L.Icmp.Sgt
           | A.Leq -> L.build_icmp L.Icmp.Sle
           | A.Geq -> L.build_icmp L.Icmp.Sge
           | _ -> raise (Failure("CODEGEN: improper relational expr!!!"))
          ) e1' e2' "relational_expr" builder
```

```
      | e -> add_expr builder e

    (* Construct code for an equality expression; return its value *)
    and equality_expr builder = function
        A.Binop(e1, ((A.Equal | A.Neq) as op) , e2)->
          let e1' = equality_expr builder e1 in
          let e2' = if e2 = A.Nullexpr then L.const_null (L.type_of e1')
                else relational_expr builder e2 in
          if e2 != A.Nullexpr then
            (match op with
                 A.Equal -> L.build_icmp L.Icmp.Eq
               | A.Neq -> L.build_icmp L.Icmp.Ne
               | _ -> raise (Failure("CODEGEN: improper equality expr!!!"))
            ) e1' e2' "equality_expr" builder
          else if (L.is_null e1') || (L.is_undef e1') then L.const_int i1_t 1
          else L.const_int i1_t 0
      | e -> relational_expr builder e

    (* Construct code for a logical and expression; return its value *)
    and logical_and_expr builder = function
        A.Binop(e1, A.And, e2) ->
          let e1' = logical_and_expr builder e1
          and e2' = equality_expr builder e2 in
          L.build_and e1' e2' "logical_and_expr" builder
      | e -> equality_expr builder e

(* Construct code for a logical or expression; return its value *)
    and logical_or_expr builder = function
        A.Binop(e1, A.Or, e2) ->
          let e1' = logical_or_expr builder e1
          and e2' = logical_and_expr builder e2 in
          L.build_or e1' e2' "logical_or_expr" builder
      | e -> logical_and_expr builder e

    (* Construct code for an assignment expression; return its value *)
    and assignment_expr builder = function
        A.Assign (A.Dereference(p), _, r) ->
          let r' = expr builder r
          and p' = postfix_expr builder p in
          ignore (L.build_store r' p' builder); r'
      | A.Assign (A.ArrayAccess(a,i), _, r) ->
          let r' = expr builder r in
          let n = L.build_zext_or_bitcast (postfix_expr builder i) i64_t "ary.cast" builder in
          let idx = L.build_gep (expr builder a) [| n |] "ary.ptr" builder in
          ignore(L.build_store r' idx builder); r'
      | A.Assign (A.StructAccess(s,m), _, r) ->
          let r' = expr builder r in
          let name = lookup_type (A.string_of_expr s) in
            let members = snd (struct_lookup (A.string_of_typ name)) in
              let rec get_idx n lst i = match lst with
                | [] -> raise (Failure("CODEGEN: id " ^ m ^ " is not a member of struct " ^
A.string_of_expr s))
                | hd::tl -> if (hd=n) then i else get_idx n tl (i+1)
              in
              let idx = get_idx m (List.map (fun (_,s,_,_) -> s) members) 0 in
            let ptr = L.build_struct_gep (lookup (A.string_of_expr s)) idx "struct.ptr" builder in
          ignore(L.build_store r' ptr builder); r'
      | A.Assign (A.StructPointerAccess(s,m), _, r) ->
          let r' = expr builder r
          and name = strip ( lookup_type (A.string_of_expr s) )
          and location = expr builder s in
            let members = snd (struct_lookup (A.string_of_typ name)) in
              let rec get_idx n lst i = match lst with
                | [] -> raise (Failure("CODEGEN: id " ^ m ^ " is not a member of struct " ^
A.string_of_expr s))
                | hd::tl -> if (hd=n) then i else get_idx n tl (i+1)
              in
              let idx = get_idx m (List.map (fun (_,sn,_,_) -> sn) members) 0 in
            let ptr = L.build_struct_gep location idx "struct.ptr" builder in
          ignore(L.build_store r' ptr builder); r'
      | A.Assign (l, op, r) ->
        let r' = expr builder r in ignore (
        (match op with
            A.Asn -> L.build_store r' (lookup (A.string_of_expr l)) builder
          | A.ModAsn ->
            let reg = lookup (A.string_of_expr l) in
```

```
            let tmp = L.build_srem (L.build_load reg "tmp" builder) r' "srem" builder in
            L.build_store tmp reg builder
        )); r' (* QUESTION: what is supposed to be returned here ? *)
      | e -> logical_or_expr builder e

    (* Construct code for an expression; return its value *)
    and expr builder = function
      | e -> assignment_expr builder e

    in (* this in, it is the end, man *)
(*********************************************************)
(* END of mutually recursive chain of expression functions *)
(*********************************************************)

    (* 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
        A.Block sl ->
          (* Go through all global and local vars, initialize them from lazy records *)
          let eval_lazy _ id_val =
            if (id_val.flag && id_val.eval) then
              ignore( id_val.eval <- false; (* the only way to overwrite mutables in OCaml *)
                      L.build_store (expr builder id_val.expr) (lookup id_val.name) builder); in
              ignore(StringMap.iter eval_lazy local_lazy_vars;);
                    (*StringMap.iter eval_lazy global_lazy_vars); *)
          List.fold_left stmt builder sl
      | A.Expr e -> ignore (expr builder e); builder
      | A.Return e -> ignore (match fdecl.A.typ with
            A.Void -> L.build_ret_void builder
          | _ -> L.build_ret (expr builder e) builder); builder
      | A.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

      | A.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

      | A.For (e1, e2, e3, body) -> stmt builder
            ( A.Block [A.Expr e1 ; A.While (e2, A.Block [body ; A.Expr e3]) ] )
    in

    (* Build the code for each statement in the function *)
    let builder = stmt builder (A.Block fdecl.A.body) in

    (* Add a return if the last block falls off the end *)
    add_terminal builder (match fdecl.A.typ with
```

```
      A.Void -> L.build_ret_void
    | t -> L.build_ret (L.const_int (ltype_of_typ t) 0))
  in

  List.iter build_function_body program.functions;
  the_module
```

Appendix C: All C+ Tests

```
/* fail-assign1.cp */

int main()
{
  int i;
  bool b;
```

```
  i = 42;
  i = 10;
  b = true;
  b = false;
  i = false; /* Fail: assigning a bool to an integer */
}
/* fail-assign2.cp */


int main()
{
  int i;
  bool b;

  b = 48; /* Fail: assigning an integer to a bool */
}
/* fail-assign3.cp */

void myvoid()
{
  return;
}

int main()
{
  int i;

  i = myvoid(); /* Fail: assigning a void to an integer */
}
/* fail-chainassigndecl.cp */

int main()
{
    int x = y = z = 1;
    return 0;
}
/* fail-dead1.cp */

int main()
{
  int i;

  i = 15;
  return i;
```

```
  i = 32; /* Error: code after a return */
}
/* fail-dead2.cp */

int main()
{
  int i;

  {
    i = 15;
    return i;
  }
  i = 32; /* Error: code after a return */
}
/* fail-expr1.cp */

int a;
bool b;

void foo(int c, bool d)
{
  int dd;
  bool e;
  a + c;
  c - a;
  a * 3;
  c / 2;
  d + a; /* Error: bool + int */
}

int main()
{
  return 0;
}
/* fail-expr2.cp */

int a;
bool b;

void foo(int c, bool d)
{
  int d;
  bool e;
  b + a; /* Error: bool + int */
```

```
}

int main()
{
  return 0;
}
/* fail-for1.cp */

int main()
{
  int i;
  for ( ; true ; ) {} /* OK: Forever */

  for (i = 0 ; i < 10 ; i = i + 1) {
    if (i == 3) return 42;
  }

  for (j = 0; i < 10 ; i = i + 1) {} /* j undefined */

  return 0;
}
/* fail-for2.cp */

int main()
{
  int i;

  for (i = 0; j < 10 ; i = i + 1) {} /* j undefined */

  return 0;
}
/* fail-for3.cp */

int main()
{
  int i;

  for (i = 0; i ; i = i + 1) {} /* i is an integer, not Boolean */

  return 0;
}
/* fail-for4.cp */

int main()
```

```
{
  int i;

  for (i = 0; i < 10 ; i = j + 1) {} /* j undefined */

  return 0;
}
/* fail-for5.cp */

int main()
{
  int i;

  for (i = 0; i < 10 ; i = i + 1) {
    foo(); /* Error: no function foo */
  }

  return 0;
}
/* fail-func1.cp */

int foo() {}

int bar() {}

int baz() {}

void bar() {} /* Error: duplicate function bar */

int main()
{
  return 0;
}
/* fail-func2.cp */

int foo(int a, bool b, int c) { }

void bar(int a, bool b, int a) {} /* Error: duplicate formal a in bar */

int main()
{
  return 0;
}
/* fail-func3.cp */
```

```
int foo(int a, bool b, int c) { }

void bar(int a, void b, int c) {} /* Error: illegal void formal b */

int main()
{
  return 0;
}
/* fail-func4.cp */

int foo() {}

void bar() {}

int print() {} /* Should not be able to define print */

void baz() {}

int main()
{
  return 0;
}
/* fail-func5.cp */

int foo() {}

int bar() {
  int a;
  void b; /* Error: illegal void local b */
  bool c;

  return 0;
}

int main()
{
  return 0;
}
/* fail-func6.cp */

void foo(int a, bool b)
{
}
```

```
int main()
{
  foo(42, true);
  foo(42); /* Wrong number of arguments */
}
/* fail-func7.cp */

void foo(int a, bool b)
{
}

int main()
{
  foo(42, true);
  foo(42, true, false); /* Wrong number of arguments */
}
/* fail-func8.cp */

void foo(int a, bool b)
{
}

void bar()
{
}

int main()
{
  foo(42, true);
  foo(42, bar()); /* int and void, not int and bool */
}
/* fail-func9.cp */

void foo(int a, bool b)
{
}

int main()
{
  foo(42, true);
  foo(42, 42); /* Fail: int, not bool */
}
/* fail-global1.cp */
```

```
int c;
bool b;
void a; /* global variables should not be void */


int main()
{
  return 0;
}
/* fail-global2.cp */

int b;
bool c;
int a;
int b; /* Duplicate global variable */

int main()
{
  return 0;
}
/* fail-if1.cp */

int main()
{
  if (true) {}
  if (false) {} else {}
  if (42) {} /* Error: non-bool predicate */
}

/* fail-if2.cp */

int main()
{
  if (true) {
    foo; /* Error: undeclared variable */
  }
}
/* fail-if3.cp */

int main()
{
  if (true) {
    42;
```

```
  } else {
    bar; /* Error: undeclared variable */
  }
}
/* fail-malloc.cp */

int main()
{
   malloc(3*3);
   return 0;
}
/* fail-mallocbool.cp */

int main()
{
   malloc(true*sizeof(int));
   return 0;
}
/* fail-nomain.cp */
/* fail-pointerassign.cp */

int main()
{
   int* i;
   bool* b;
   i = b; /* Error: illegal pointer assignment */
}
/* fail-return1.cp */

int main()
{
  return true; /* Should return int */
}
/* fail-return2.cp */

void foo()
{
  if (true) return 42; /* Should return void */
  else return;
}

int main()
{
  return 42;
```

```
}
/* fail-while1.cp */

int main()
{
  int i;

  while (true) {
    i = i + 1;
  }

  while (42) { /* Should be boolean */
    i = i + 1;
  }

}

/* fail-while2.cp */

int main()
{
  int i;

  while (true) {
    i = i + 1;
  }

  while (true) {
    foo(); /* foo undefined */
  }

}
/* test-add1.cp */

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

int main()
{
  print( add(17, 25) );
  return 0;
}
```

```
/* test-addinit.cp */

int main()
{
   int x = 4 + 2;
   print(x);
   return 0;
}
/* test-app_insert.cp */

/* ============================== Data Structures
================================== */

/* Stores a key-value property pair for a given node */
struct Prop {
  /* memory struct to retain properties */
  int prop_id;
  char* property;
  char* value;
};

/* Node data structure */
struct Node {
  /* node_id is hash int id, node_label is string id, prop are properties */
  int node_id;
  int prop_cnt;
  char* node_label;
  Prop *property[10];
};

int node_index_avail = 1000;

/* Edge data structure */
struct Edge {
  /* edge_id is has int id, edge_label is string id, prop are properties */
  int edge_id;
          int num_pairs;
          int prop_cnt;
  char* edge_label;
          Node *src;
          Node *trg;
  Prop *property;
};
```

```
int nodes = 0;
int edges = 0;
int properties = 0;
/* int node_index_avail = 1000; */
/* Node* nodeIndex[node_index_avail]; */
Node* dummyItem;
Node* item;
/* struct edge* edgeIndex[1000]; */
/* struct edge* dummyItemEdge; */
/* struct edge* itemEdge; */
Prop* propIndex[1000];
Prop* dummyItemProp;
Prop* itemProp;

/* ============================== Creation Functions ========================== */

/* Insert a Node into Hashable Index for Reference */
int insert_node(char* key) {

  /* have to declare all variables at the beginning */
  Node *item = (Node*) malloc(sizeof(Node));
  int hashIndex = 0, i = 0; /* have to declare are variables at the top of a function */

  /* FIXME this should be globals!!! */
  Node* nodeIndex[node_index_avail];

  /* make a tmp node to initialize the "global" nodeIndex array */
  Node* tmpNode = (Node*) malloc(sizeof(Node));
  tmpNode->node_label = "init";
  tmpNode->prop_cnt = 0;
  tmpNode->node_id = 0;


  /* FIXME: strdup is broken, might be returning null */
  /* item->node_label = strdup(key); */
  item->node_label = key;
  item->prop_cnt = 0;

  /* get the hash */
  /* printf is not that fancy yet */
  printf("Insert node: ");
  printf(key);
  printf("\n");
```

```
 for (i = 0; i < node_index_avail; i++){
  nodeIndex[i] = tmpNode;
 }

 /* move in array until an empty or deleted cell */
 while(hashIndex < node_index_avail && nodeIndex[hashIndex]->node_id != -1) {
   /* go to next cell */
   ++hashIndex;
 }
 node_index_avail--;

 item->node_id = hashIndex;
 nodeIndex[hashIndex] = item;
 printf("inserted at index: "); print(hashIndex); printf("\n");
 return hashIndex;

}

/* ============================== MAIN ========================== */
int main()
{
 int main_node_id = insert_node("main_node");
 printf("Main node id: "); print(main_node_id); printf("\n");
 return 0;
}
/* test-application.cp */

int size = 1000;

struct Record {
   char* name;
   int student_id;
   int grade;
};

int main()
{
   Record class[size];
   Record tmp;
   int i = 0;

   for (i = 0; i < size; i++){
      tmp.student_id = size % (2 * i + 3);
      class[i] = tmp;
```

```
    }

    tmp = class[7];
    print(tmp.student_id);

    return 0;
}
/* test-arith1.cp */

int main()
{
  print(39 + 3);
  return 0;
}
/* test-arith2.cp */

int main()
{
  print(1 + 2 * 3 + 4);
  return 0;
}
/* test-arith3.cp */

int foo(int a)
{
  return a;
}

int main()
{
  int a;
  a = 42;
  a = a + 5;
  print(a);
  return 0;
}
/* test-arrayaccess.cp */

int main()
{
    int a[10], i;
    for(i = 0; i < 9; i++){
       a[i] = i;
    }
```

```
   print(a[3]);
   print(a[4]);
   print(a[5]);
   print(a[6]);
   return 0;
}
/* test-arraydecl.cp */

int main()
{
   int a[10], i;
   a;
}
/*
 * test-castvoidptr_to_intptr.cp
 *
 * Test that the cast functionality works as an expression
 *
 * Date 4-12-2017
 *
 */

int main()
{
  void* x;
  int* y;
  y = (int*) x;
  return 0;
}
/* test chain of assignment operators */

int main()
{
   int a, b, c, d;
   a = b = c = d;
   return 0;
}
/* test-charstar.cp */

int main()
{
   char* c = "this is a char star.\n";
   printf(c);
   return 0;
```

```
}
/* test initialization lists */

int main()
{
   int a = 1, b = 2, c = 3, d = 4;
   print(d);
   return 0;
}
/* test-fib.cp */

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

int main()
{
 print(fib(0));
 print(fib(1));
 print(fib(2));
 print(fib(3));
 print(fib(4));
 print(fib(5));
 return 0;
}
/* test-for1.cp */

int main()
{
 int i;
 for (i = 0 ; i < 5 ; i = i + 1) {
   print(i);
 }
 print(42);
 return 0;
}
/* test-for2.cp */

int main()
{
 int i;
 i = 0;
```

```
  for ( ; i < 5; ) {
    print(i);
    i = i + 1;
  }
  print(42);
  return 0;
}
/* test-func1.cp */

int add(int a, int b)
{
  return a + b;
}

int main()
{
  int a;
  a = add(39, 3);
  print(a);
  return 0;
}
/* test-func2.cp */

/* Bug noticed by Pin-Chin Huang */

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

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

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

  print(i);
  return 0;
}

/* test-func3.cp */

void printem(int a, int b, int c, int d)
```

```
{
  print(a);
  print(b);
  print(c);
  print(d);
}

int main()
{
  printem(42,17,192,8);
  return 0;
}
/* test-func4.cp */

int add(int a, int b)
{
  int c;
  c = a + b;
  return c;
}

int main()
{
  int d;
  d = add(52, 10);
  print(d);
  return 0;
}
/* test-func5.cp */

int foo(int a)
{
  return a;
}

int main()
{
  return 0;
}
/* test-func6.cp */

void foo() {}

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

```
int main()
{
  print(bar(17, false, 25));
  return 0;
}
/* test-func7.cp */

int a;

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

int main()
{
  foo(73);
  print(a);
  return 0;
}
/* test-func8.cp */

void foo(int a)
{
  print(a + 3);
}

int main()
{
  foo(40);
  return 0;
}
/* test-gcd.cp */

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

int main()
```

```
{
  print(gcd(2,14));
  print(gcd(3,15));
  print(gcd(99,121));
  return 0;
}
/* test-gcd2.cp */

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

int main()
{
  print(gcd(14,21));
  print(gcd(8,36));
  print(gcd(99,121));
  return 0;
}
/* test-gobainit.cp */

int size = 100;

int main()
{
    print(size);
    return 0;
}
/* test-global1.cp */

int a;
int b;

void print_a()
{
  print(a);
}

void print_b()
{
  print(b);
```

```
}

void incab()
{
  a = a + 1;
  b = b + 1;
}

int main()
{
  a = 42;
  b = 21;
  print_a();
  print_b();
  incab();
  print_a();
  print_b();
  return 0;
}
/* test-global2.cp */

bool i;

int main()
{
  int i; /* Should hide the global i */

  i = 42;
  print(i + i);
  return 0;
}
/* test-global3.cp */

int i;
bool b;
int j;

int main()
{
  i = 42;
  j = 10;
  print(i + j);
  return 0;
}
```

```
/* test-hello.cp */

int main()
{
  print(42);
  print(71);
  print(1);
  return 0;
}
/* test-if1.cp */

int main()
{
  if (true) print(42);
  print(17);
  return 0;
}
/* test-if2.cp */

int main()
{
  if (true) print(42); else print(8);
  print(17);
  return 0;
}
/* test-if3.cp */

int main()
{
  if (false) print(42);
  print(17);
  return 0;
}
/* test-if4.cp */

int main()
{
  if (false) print(42); else print(8);
  print(17);
  return 0;
}
/* test-if5.cp */

int cond(bool b)
```

```
{
  int x;
  if (b)
    x = 42;
  else
    x = 17;
  return x;
}

int main()
{
 print(cond(true));
 print(cond(false));
 return 0;
}
/* test-inlinefor.cp */

int main()
{
   int i;
   for (i=0;i < 10;i++){}
   return 0;
}
/* test-lazybooboo.cp */

int main()
{
   int x = 1;
   int y = x + 1;
   x = 5;
   print(x);
   print(y);
}
/* test-local1.cp */

void foo(bool i)
{
  int i; /* Should hide the formal i */

  i = 42;
  print(i + i);
}

int main()
```

```
{
  foo(true);
  return 0;
}
/* test-local2.cp */

int foo(int a, bool b)
{
  int c;
  bool d;

  c = a;

  return c + 10;
}

int main() {
 print(foo(37, false));
 return 0;
}
/* test-mallocassign.cp */

int main()
{
   int* array = (int*) malloc(sizeof(int));
   array;
   return 0;
}
/* test-mallocassign2.cp */

int main()
{
   int x;
   int* ptr = (int*) malloc(sizeof(int));
   *ptr = 5;
   print(*ptr);
   return 0;
}
/* test-mallocexpr.cp */
/*
 * Test for assigning a value to a pointer, via the
 * linked in C malloc function
 *
 * Date 4-11-2017
```

```
 *
 */

int main()
{
  malloc(sizeof(int));
  return 0;
}
/* test-modequals.cp */

int main()
{
   int i;
   i = 11;
   i %= 2;
   print(i);
   return 0;
}
/* test-multimalloc.cp */

/* test for allocating multiple entries */

int main()
{
   malloc(10*sizeof(int));
   malloc(sizeof(bool)*300);
   return 0;
}
/* test-multiptraddr.cp */

int main()
{
   int a;
   int *b = &a;
   int *c = &a;

   *b = 5;
   *c = 42;

   print(a);
   print(*b);
   print(*c);

   return 0;
```

```
}
/* test-nestedcall.cp */

int a (int x)
{
   return x + 1;
}

int b (int y)
{
   return y + 2;
}

int main()
{
   int z = 5;
   print(b(a(z)));
   return 0;
}
/* test-nodeEdgeHash.cp */

/* Hashable Code for Node/Edge ID */
int hashCode(int key, int size) {
  return key % size;
}

/* Node data structure */
struct Node {
      /* node_id is hash int id, node_label is string id, prop are properties */
      int node_id;
      char* node_label;
      char* prop;
};

/* Edge data structure */
struct Edge {
      /* edge_id is has int id, edge_label is string id, prop are properties */
      int edge_id;
      int source_id;
      int sink_id;
      char* edge_label;
      char* prop;

};
```

```
int size = 100;
Node* hashArray[100];
Node* dummyItem;
Node* item;

int main()
{
   Node n1, n2, n3;
   Edge e1, e2, e3;
   int h = hashCode(1005, size);
   print(h);
   return 0;
}
/* test-ops1.cp */

int main()
{
  print(1 + 2);
  print(1 - 2);
  print(1 * 2);
  print(100 / 2);
  print(99);
  printb(1 == 2);
  printb(1 == 1);
  print(99);
  printb(1 != 2);
  printb(1 != 1);
  print(99);
  printb(1 < 2);
  printb(2 < 1);
  print(99);
  printb(1 <= 2);
  printb(1 <= 1);
  printb(2 <= 1);
  print(99);
  printb(1 > 2);
  printb(2 > 1);
  print(99);
  printb(1 >= 2);
  printb(1 >= 1);
  printb(2 >= 1);
  return 0;
}
```

```
/* test-ops2.cp */

int main()
{
  printb(true);
  printb(false);
  printb(true && true);
  printb(true && false);
  printb(false && true);
  printb(false && false);
  printb(true || true);
  printb(true || false);
  printb(false || true);
  printb(false || false);
  printb(!false);
  printb(!true);
  print(-10);
  print(--42); /* WHY DOESNT THIS WORK */
}
/* test-pluspluspostfix.cp */

int main()
{
   int i = 1;
   i++;
   print(i);
   return 0;
}
/* test-plusplusprefix.cp */

int main()
{
   int i = 1;
   ++i;
   print(i);
   return 0;
}
/* test-pointeradd.cp */

/* test for enabling pointer addition */

int main()
{
   int* x;
```

```
   x = (int*) malloc(2*sizeof(int));
   *x = 5;
   x = x + 1;
   *x = 10;
   print(*x);
   return 0;
}
/* test-pointeraddress.cp */

/*
 * Test for getting the address of a pointer, via the
 * via the '&' operator
 *
 * Date 4-11-2017
 *
 */

int main()
{
  int x;
  int * ptr;
  int ** ptr1;
  x = 5;
  ptr = &x;
  ptr1 = &ptr;
  return 0;
}
/* test-pointerassign.cp */

/*
 * Test for assigning a value to a pointer, via the
 * linked in C malloc function
 *
 * Date 4-11-2017
 *
 */

int main()
{
  int * ptr1;
  int * ptr2;
  ptr1 = ptr2;
  return 0;
}
```

```
/* test-pointerdecl.cp */

/*
 * Test for declaring a "pointer variable"
 *
 * This simple test just needs not to fail when the variable is declared.
 * Does NOT test assignment to the pointer, which will be another test.
 *
 * Date: 4-11-2017
 *
 */

int main()
{
  int * ptr;
  return 0;
}
/* test-pointerderef.cp */

/* This test uses the pointer dereference operator
 *
 * Date: 4-22-2017
 *
 */

int main()
{
   int x;
   int * p;
   int y;
   x = 5;
   p = &x;
   y = *p;
   print(y);
   return 0;
}
/* test-pointerexpr.cp */

/*
 * Test for proving that a pointer is an acceptable statement
 *
 * Date 4-12-2017
 *
 */
```

```
int main()
{
  int * ptr;
  ptr;
  return 0;
}
/* test-pointerfree.cp */

/*
 * Test for freeing allocated memory with linked-in C free() function
 *
 * Date 4-13-2017
 *
 */

int main()
{
  int * ptr;
  ptr = (int*) malloc(sizeof(int));
  free(ptr);
  return 0;
}
/* test-pointerlhs.cp */

/* Tests whether a pointer can be used on the LHS without a declaration */

int main()
{
   int * p;
   p = (int*) malloc(sizeof(int));
   *p = 5;
   print(*p);
   return 0;
}
/* test-pointermalloc.cp */

/*
 * Test for assigning a value to a pointer, via the
 * linked in C malloc function
 *
 * Date 4-11-2017
 *
 */
```

```
int main()
{
  int * ptr;
  ptr = (int*) malloc(sizeof(int));
  return 0;
}
/* test-printbig.cp */

/*
 * Test for linking external C functions to LLVM-generated code
 *
 * printbig is defined as an external function, much like printf
 * The C compiler generates printbig.o
 * The LLVM compiler, llc, translates the .ll to an assembly .s file
 * The C compiler assembles the .s file and links the .o file to generate
 * an executable
 */

int main()
{
 printbig(72); /* H */
 printbig(69); /* E */
 printbig(76); /* L */
 printbig(76); /* L */
 printbig(79); /* O */
 printbig(32); /*   */
 printbig(87); /* W */
 printbig(79); /* O */
 printbig(82); /* R */
 printbig(76); /* L */
 printbig(68); /* D */
 return 0;
}
/* test-printf.cp */

/*
 * Test for linking "hello world" enablement in CPLUS
 *
 * printf is defined as an external function
 * The C compiler generates printbig.o, which includes the C standard IO library
 * which includes printf.
 * The LLVM compiler, llc, translates the .ll to an assembly .s file
 * The C compiler assembles the .s file and links the .o file to generate
```

```
 * an executable
 *
 * Date: 3-20-2017
 *
 */

int main()
{
  printf("Hello, My name is Alex\n");
  return 0;
}
/* test-sizeof.cp */

/*
 * test-sizeof.cp
 * Test for using the sizeof operator
 *
 * Date 4-12-2017
 *
 */

int main()
{
   size_t x;
   x = sizeof(int);
   return 0;
}
/* test-strlit.cp */

void accept(string s)
{
    return;
}

int main()
{
    string s;
    s = "hello world";
    accept(s);
    return 0;
}
/* test-structaccess.cp */

struct Test {
```

```
   int x;
   bool y;
   int a[10];
};

int main()
{
   Test t;
   t.x = 8;
   print(t.x);
   return 0;
}
/* test-structarrow.cp */

struct Test {
   int x;
   bool y;
   int a[10];
};

int main()
{
   Test* t = (Test*) malloc(sizeof(Test));
   t->x = 8;
   print(t->x);
   return 0;
}
/* test-structdecl.cp */

struct Test {
   int x;
   bool y;
   int a[10];
};

int main()
{
   Test t;
   return 0;
}
/* test-var2.cp */

int main()
{
```

```
  int a;
  a = 42;
  print(a);
  return 0;
}
/* test-var2.cp */

int a;

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

int main()
{
  foo(73);
  print(a);
  return 0;
}
/* test-while1.cp */

int main()
{
  int i;
  i = 5;
  while (i > 0) {
    print(i);
    i = i - 1;
  }
  print(42);
  return 0;
}
/* test-while2.cp */

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

```
}

int main()
{
  print(foo(7));
  return 0;
}
```