ConLangLang (CLL)

Final Report

Annalise Mariottini

(aim2120)

Spring 2021

Contents

1 Introduction **4**

2 Language Tutorial **5**
2.1 Getting Started . . . 5
2.2 Data Types . . . 5
2.2.1 Operators . . . 5
2.2.2 Regexes . . . 6
2.2.3 Dicts and Lists . . . 6
2.3 Functions . . . 7
2.4 User-Defined Types and TypeDefs . . . 7
2.5 Control Flow . . . 8
2.5.1 Match . . . 8
2.5.2 If-Else . . . 8
2.5.3 Do-While . . . 9

3 Language Reference Manual **10**
3.1 Introduction . . . 10
3.2 LRM Syntax . . . 10
3.3 Lexical Conventions . . . 11
3.3.1 Comments . . . 11
3.3.2 Separators . . . 11
3.4 Identifiers . . . 11
3.5 Keywords . . . 11
3.5.1 Stdin . . . 12
3.6 Literals . . . 12
3.6.1 Integer Literals . . . 12
3.6.2 Float Literals . . . 12
3.6.3 Boolean Literals . . . 13
3.6.4 String Literals . . . 13
3.6.5 Regex Literals . . . 13
3.6.6 List Literals . . . 13
3.6.7 Dictionary Literals . . . 14
3.6.8 Function Literals . . . 14
3.7 Operators . . . 14
3.7.1 Type-Grouped Operators . . . 14
3.7.2 Untyped Operators . . . 15
3.8 Data Variables . . . 16
3.8.1 Primitive Data Types . . . 16
3.8.2 Complex Data Types . . . 16
3.9 Function Variables . . . 17
3.9.1 Function Assignment . . . 17
3.9.2 Function Call . . . 17
3.10 Type Variables . . . 17
3.10.1 Type . . . 17

3.10.2 TypeDef . . . 18
3.11 Control Flow Expressions . . . 19
3.11.1 Variable Scope . . . 19
3.11.2 Expression Value . . . 19
3.11.3 Function Block . . . 19
3.11.4 Match . . . 19
3.11.5 If-Else . . . 20
3.11.6 Do-While . . . 20
3.12 Program Structure . . . 20
3.12.1 Production Rules . . . 21
3.13 Standard Library . . . 21
3.13.1 String . . . 21
3.13.2 Regex . . . 21
3.13.3 List . . . 22
3.13.4 Dictionary . . . 22
3.14 Example Program . . . 23
3.14.1 Input . . . 25
3.14.2 Output . . . 25

4 Project Plan **27**
4.1 Development Process . . . 27
4.1.1 Initial Stages . . . 27
4.1.2 Writing the LRM . . . 27
4.1.3 AST and SAST . . . 27
4.1.4 LLVM Code Generation . . . 27
4.1.5 Automation . . . 28
4.2 Style Conventions . . . 28
4.3 Project Timeline . . . 29
4.4 Roles and Responsibilities . . . 29
4.5 Software Development Environment . . . 29
4.6 Project Log . . . 29

5 Architectural Design **45**
5.1 Compiler Pipeline Diagram . . . 45
5.2 Component Descriptions . . . 45
5.2.1 Scanner . . . 45
5.2.2 Parser . . . 45
5.2.3 Semantic Check . . . 46
5.2.4 Code Generation . . . 46
5.2.5 C Library Files . . . 46
5.2.6 Binary Executable . . . 46

6 Test Plan **47**
6.1 Example Tests and LLVM Output . . . 47
6.1.1 A Small Test . . . 47
6.1.2 A Larger Test . . . 52
6.2 Test Suite . . . 61
6.3 Automation . . . 62
6.3.1 testall.zsh . . . 62
6.3.2 make_test_exe.sh . . . 64
6.3.3 make_test_out.sh . . . 64
6.3.4 Makefile . . . 66

7 Lessons Learned **67**
7.1 Reflections . . . 67
7.2 Recommendations to Future Groups . . . 67

8 Appendix **68**
8.1 OCaml Files . . . 68

8.1.1 scanner.mll 68
8.1.2 ast.ml 69
8.1.3 parser.mly 72
8.1.4 sast.ml 74
8.1.5 semant.ml 76
8.1.6 codegen.ml 86
8.1.7 cll.mll 114
8.2 C Lib Files 115
8.2.1 find_prime.c 115
8.2.2 hash_table.c 117
8.2.3 linked_list.c 124
8.2.4 malloc_manager.c 127
8.2.5 regex.c 127
8.2.6 stdin.c 129

Chapter 1

Introduction

ConLangLang (CLL) is a semi-imperative programming language inspired by features of functional programming languages and designed for the purposes of categorizing, generating, and organizing language data. Dictionaries and lists are the two means of storing language data in CLL. These imperative data structures are mutable, but can be manipulated in a manner more similar to that of functional programming through standard library functions that include map and fold function application. The language also features regex functionality for matching and replacing substrings of text, user-based type designation for categorizing purposes, and access to whitespace-separated text piped through stdin.

CLL language design was inspired by the concise and effective nature of OCaml in particular. Like OCaml, CLL is strongly-typed. Also similarly to OCaml, almost every element of CLL is a expression, meaning that it has a return value. The only non-expressive elements of the language are definitions of user-defined types. In any place one might put a literal integer, one could also put an if-else that returns an integer. Also like OCaml, functions can be nested and any child function has access to its parent's (and it's parent's parents...) variables. Although functions cannot be passed or returned like other data objects, this transparent access to all variable of higher scope allows functions to access variables without need for excessive passing and returning of arguments by the user. Lastly, also similarly to OCaml, variable names in CLL do not have declared type designations. Instead, they are assigned the type of the literal value they are assigned. If a name is reused, the prior value assigned to the name is hidden by the new value/type designation.

Overall, CLL is intended to provide a compact and concise means of converting language data into the user's intended form with minimal boilerplate needed by the user.

Chapter 2

Language Tutorial

2.1 Getting Started

In terms of syntax, CLL is inspired greatly by OCaml. A CLL program consists of a series of statements each ending in a semicolon. These statements may be either user type/typdef definitions or expressions. Therefore, a program may be as simple as a single expression:

```
0;
```

That's not very exciting, so let's make some variables:

```
x = 5;
y = 'hello';
sprint(y); # prints hello
```

As you can see, variables are simply names assigned to values. A variable name is equivalent to the literal value it is assigned to. If that value is constant, once a variable is used, its value is set in that use (its value may change later, but it will never update previous uses). If that value is mutable, a use of that variable may change in value if that variable is reassigned or its data object modified.

2.2 Data Types

Following are the data types that may be used in CLL:

- **Constant**: int, float, bool
- **Mutable**: string, regex, list, dict

2.2.1 Operators

I'll briefly discuss operators for the above types here. Ints and floats have all the regular operations (strongly typed, i.e. both operands must be of the same type):

- add : `+`
- subtract : `-`
- multiply : `*`
- divide : `/`
- remainder : `%`

Booleans also have the regular operators (not, and, or):

- not : `!`
- and : `&&`
- or : `||`

Comparison operators can be used on ints, floats, and booleans:

- equals : `==`
- greater than : `>`
- less than : `<`

Strings and lists have the concat operator : `^`

Compatible* types can be converted to one another using the cast operator : `(`*type*`)`

*See the LRM for what is "compatible".

To assign a variable name a given value we use the assignment operator : `=`

2.2.2 Regexes

Regexes can be used to match a given POSIX-extended regular expression with a given string. They can also be used to replace all instances of a matched group within the string with a replacement string (group 0 is the whole string, group 1 is the first grouping, and so on),

```
result = rematch("ma(tch)|(sh)", 'mash');
# true
s_ = resub("be(gan)", 'he began', 'gins', 1);
# 'he begins'
```

2.2.3 Dicts and Lists

Dictionaries (a.k.a. dicts) (implemented as hashtables) and lists (implemented as linked lists) can hold any data types. The data type(s) held in a dict or list must be declared in angle brackets preceding the literal dict or list object. Lists can hold one type and dictionaries hold at most two types: one for keys, one for values. When declaring a complex object within a complex object, the complex object declaration must also contain a type declaration. The following is an example of some complex data objects:

```
l = <regex>{"a*", "b*", "c*"};
d1 = <int,string>{ 0:'a', 1:'b', 2:'c' };
d2 = <string,list<string>>{
    'a':<string>{'apple', 'above', 'alligator'},
    'b':<string>{'bacon', 'before', 'bee' }
};
```

Dicts and lists have a variety of functions to access and manipulate their contents:

- ladd & dadd
- lremove & dremove
- lmem & dmem
- lget & dget
- lsize & dsize
- dkeys

These functions are mostly self explanatory. Their specific function signatures can be seen in the 3.13 Standard Library. Here's an example of some of uses:

```
l = <int>[];
ladd(l, 1);
ladd(l, 2);
ladd(l, 0, lsize(l));
# l is now [2,1,0]
lmem(l, 1); # true
lremove(l); # removes 2
```

2.3 Functions

Functions (a.k.a. funs) can be used both in literal form or when assigned to a variable name. A fun literal contains its function signature in angle brackets preceding the function body. The return value of a fun is determined by the last statement reached within its function body. Therefore, a return value is mandatory. Something else to note about funs: variables in higher scopes are accessible to any child funs in lower scopes.

```
fun ret_10 = <:int>{
    10;
};
x = <int x:int>{
    x + ret_10();
}(24); # returns 34
```

In addition to regular function calls, funs can be applied to strings, lists, and dicts through the map and fold standard library functions. Mapping creates a new object of the same type with each element modified by the given function. Folding creates a new accumulated object (of any type) that is the result of applying the function to each element of a given object.

```
l = <string>['boy','girl','dog','cat'];
l_plural = lmap(<string s:string>{s ^ 's';}, l};
# ['boys','girls','dogs','cat's]

string_of_dict = <string a, int k, int v:string>{
    a ^ (string)k ^ ':' ^ (string)v ^ ' ';
};
dict_string = dfold(string_of_dict,'',<int,int>{1:212, 2:646, 3:917});
# '1:212 2:646 3:917'
```

2.4 User-Defined Types and TypeDefs

User-defined types are built upon existing data types for categorization purposes, while typedefs are structured objects with children variables of defined data types.

User-defined types are mostly aesthetic, in that the underlying data is same as a built-in data type; however, the compiler will treat user-defined types as different from their associated built-in types unless cast back to the associated built-in type. User-defined types are declared in groupings associated with a user-defined type variable, but this variable is functionally useless other than the aesthetic naming it provides to the group. User-defined types are assigned to values through casting. A value can have more than one user-defined type. (Syntax note: user-defined types MUST be capitalized.)

User-defined typedefs define memory structure templates. A given typedef structure is used to create instance variables of this structure to store actual data according to the declared types of

the typedef's children. Unlike other variables, typedef variable instances must be declared of the specific typedef that they should be structured after. (Syntax note: user-defined typedefs MUST be capitalized and preceded with a $.)

```
type verb = { Present<string>, Past<string>, Future<string>,
    Plural<string>, Singular<string>, Infinitive<string> };

typedef $Verb = {
    Infinitive inf;
    list<Present> pr;
    list<Past> pa;
    list<Future> f;
};

$Verb to_be = {
    inf = (Infinitive)'to be';
    pr = <Present>[(Present,Plural)'are', (Present,Singular)'is'];
    pa = <Past>[(Past,Plural)'were', (Past,Singular)'was'];
    f = <Future>[(Future,Singular,Plural)'will be'];
};
```

2.5 Control Flow

Control flow expressions are expressions that contain one or more "blocks", one of which will run on a given input. A "block" is simply a series of statements, the last being the return value of the entire block (thus, the last statement of any block must be an expression). The last statement of any possible block must match the declared return type of the control flow expression.

2.5.1 Match

A match matches a given input with one of its contained blocks. A match can either match by value or by user-defined type. The last block of a match is the "default" block, which will be run if no other block is matched with the input.

```
type t = { Foo<int> };
x = (Foo) 2;
match:string ((int)x) byvalue {
    1 { 'one'; }
    2 { 'two'; }
    default { 'idk'; }
}; # returns 'two'

match:string (x) bytype {
    Foo { 'is foo'; }
    default { 'is not foo'; }
}; # returns 'is foo'
```

2.5.2 If-Else

An if-else is two blocks, the first of which will run on a "true" input and the second of which will run on a "false" input.

```
if:string (x % 2 == 0) {
    'is even';
} else {
    'is odd';
};
```

2.5.3 Do-While

A do-while is one block that is run at least once, then repeatedly while its given input is true. The value of the last statement on the last performed iteration is the output.

```
l = <int>[0];
dowhile:list<int> (lsize(l) < 6) {
    x = lget(l,0);
    ladd(l,x+1);
}; # returns <int>[5,4,3,2,1,0]
```

Chapter 3

Language Reference Manual

3.1 Introduction

ConLangLang (CLL) is a language designed for natural and constructed language synthesis and analysis. Word pattern matching, word set creation, and word categorization are focuses of this language's functionality. As implied by the name, this language was inspired by the immense task of creating constructed languages (conlangs). CLL hopes to ease the task of generating words based off pattern matching and substitution of words, consolidating words based on word type, and printing out this information.

CLL is a semi-imperative programming language with functional programming language features, its main goal being the the output of language data. Users of this language are encouraged to build structure from existing language data or create new language data from the ground up. The highlights of this language include its regular expressions functionality, user-defined types, complex data objects, nested functions, automatic garbage collection, and ability to access stdin input.

3.2 LRM Syntax

The following formatting styles are used to indicate code in the CLL language:

```
Monolength
Italicized Monolength
```

Terminals (literal characters) are in regular monolength styling and nonterminals (productions) are in italicized monolength styling. Productions are to be expanded into additional nonterminals and terminals through the production rules laid out in the document.

The following format will be used to define these production rules:

$$nonterminal \rightarrow production$$

For a given nonterminal, either additional arrows or the '|' symbol can be used to indicate choice in production. Nonterminals are in *italics* and terminals are unstyled.

There are many nonterminals used throughout this document. Most are contextualized within the section in which they are used. You can find a top-down overview of more productions in 3.12.1 Production Rules. In particular, pay attention to the definitions of *type* and *expr*.

3.3 Lexical Conventions

A CLL program consists of a series of tokens interpreted from written text. There are 5 types of tokens: identifiers, keywords, literals, operators, and separators. A program is made up of a series of these tokens arranged according to the syntax and semantic conventions of the language. Whitespace and comments are ignored, except where they serve to separate tokens. Whitespace is required to separate tokens that would be otherwise unrecognizable as separate without the whitespace.

3.3.1 Comments

The following indicate the start and end of ignored comment sections:

Multi-line: *start* → `{#`, *end* → `#}`
Single-line: *start* → `#`, *end* → `\n`

3.3.2 Separators

In CLL, separators are any characters used alongside other tokens that allow for the syntactical and/or semantic understanding of the statements and expressions of a program. These tokens do not have meaning within themselves, but rather allow for the program to understand the intentions of the language user.

3.4 Identifiers

Identifiers are used to identify data variables (used to store evaluated data), function variables (used to store functions), type variables (used to define type set), and user-defined types and typedefs.

All variable identifiers must match the following regex pattern:

```
['a'-'z']['a'-'z' 'A'-'Z' '0'-'9' '_']*
```

User-defined type identifiers must match the following regex pattern:

```
['A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']*
```

User-defined typedef identifiers must match the following regex pattern:

```
'$'['A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']*
```

Throughout this document, *id* will refer to a variable identifier, *Id* will refer to a type identifier, and $*Id* will refer to a typedef identifier.

3.5 Keywords

The following keywords are reserved with predefined meaning and cannot be used for anything other than the compiler's predetermined use.

Types

```
int
float
bool
string
regex
list
dict
fun
type
typedef
```

Control Flow

```
match
with
byvalue
bytype
default
dowhile
if
else
```

Values

```
true
false
stdin
```

3.5.1 Stdin

Text that is passed to a CLL program though stdin is accessible through the keyword `stdin`, which is a list composed of the words passed through stdin separated by whitespace.

3.6 Literals

Literals exists for the following types: integer, float, boolean, string, regex, list, dictionary, and function. Literals are evaluated as expressions that contain data (or, in the case of function literals, procedures to output data).

There are 3 literals that contain or input/output other data type: list, dictionary, and function. These function literals will declare these data types using a *type* declaration, which allows for recursive definition of type (e.g. a dict that contains lists of lists of ...). The following are the *type* productions:

$$
\begin{aligned}
\textit{type} &\rightarrow \text{int} \mid \text{float} \mid \text{bool} \mid \text{string} \mid \text{regex} \mid Id \\
&\rightarrow \text{list}<\textit{type}> \\
&\rightarrow \text{dict}<\textit{type},\textit{type}> \\
&\rightarrow \text{fun}<formallist:\textit{type}>
\end{aligned}
$$

(We will define *formallist* later.)

3.6.1 Integer Literals

An integer literal consists of a series of digits `0-9` interpreted as a base-10 number. Any leading zeroes are ignored.

3.6.2 Float Literals

A float literal is a decimal number `[0-9]+.[0-9]+`. I.e. all decimals must have a decimal point surrounded by integers on both sides.

E.g. `1.0` and `0.1` accepted, but not `1.` or `.1`.

Table 3.1: Regex Conventions

Symbol	Interpretation
. :	Matches any character
* :	Matches the preceding expression zero, one or several times (postfix)
+ :	Matches the preceding expression one or several times (postfix)
? :	Matches the preceding expression once or not at all (postfix)
[..] :	Character set. Ranges are denoted with -, as in [a-z]. An initial ^, as in [^0-9], complements the set. To include a] character in a set, make it the first character of the set. To include a - character in a set, make it the first or the last character of the set.
^ :	Matches at beginning of line: either at the beginning of the matched string, or just after a \n character.
$:	Matches at end of line: either at the end of the matched string, or just before a \n character.
$\vert$:	Matches the preceding element or the following element.
(..) :	Grouping of the enclosed subexpression.
{n,m} :	Matches the preceding element at least m and not more than n times.
{m} :	Matches the preceding element exactly m times.
{m,} :	Matches the preceding element at least m times.
{,n} :	Matches the preceding element not more than n times.

3.6.3 Boolean Literals

A boolean literal is either the value `true` or `false`.

3.6.4 String Literals

A string literal is a sequence of ≥ 0 single-byte characters enclosed in single quotes. Since each character has a length of one byte, any language encoding that uses byte-length units works (e.g. ASCII and UTF-8). Note: string literals cannot contain the single-quote character `'`.

3.6.5 Regex Literals

A regex literal is a series of string literals and symbolic conventions enclosed within double quotes. This string is compiled into a regular expression. The symbolic conventions for creating regex literals are listed in Table 3.1 and follow the POSIX Extended Regular Expression standards. Note: regexe literals cannot contain the double-quote character `"`.

Example: `"(hello)*!"` matches the language {!, hello!, hellohello!, hellohellohello!, ...}

3.6.6 List Literals

A list literal is a linked list of identically-typed values and takes the following form:

<*type*>[*exprlist*]

Where *exprlist* is a series of ≥ 0 expressions (of the list's type) separated by commas:

$$exprlist \rightarrow expr,\ exprlist \mid expr \mid \epsilon$$

The following is an example of a list literal: <int>[1,2,x,3-5] (assuming `x` in an `int`)

3.6.7 Dictionary Literals

A dictionary literal is a hashtable mapping of key-value pairs and takes the following form:

```
<type,type>{exprpairlist}
```

Where *exprpairlist* is a series of ≥ 0 expression pairs:

$$exprpairlist \rightarrow exprpair,\ exprpairlist \mid exprpair \mid \epsilon$$
$$exprpair \rightarrow expr : expr$$

The following is an example dictionary literal: `<int,string>{ 0: 'hello', 1: 'world'}`

3.6.8 Function Literals

A function literal is a block expression that takes in formal arguments and outputs a given type based upon the expressions within its statement block. A function literal takes the following form:

```
<formallist:typeo>{ stmtblock }
```

Where $formallist$ is a series of ≥ 0 *type id* variable declarations and *stmtblock* is a series of ≥ 1 statements separated by semicolons:

$$formallist \rightarrow type\ id\ ,\ formallist \mid type\ id \mid \epsilon$$
$$stmtblock \rightarrow stmt;\ stmtblock \mid stmt;$$

3.7 Operators

Operators are expressions used to process either 1 or 2 inputs and produce an output. There are 2 types of operators: type-grouped and untyped. Table 3.2 indicates operator precedence, type (when applicable), symbol, function, number of inputs, and associativity. When stringing together operations, one can override precedence and associativity defaults by enclosing the desired expression in parentheses:

```
( expr )
```

3.7.1 Type-Grouped Operators

Type-grouped operators have known types associated with their operations. There are two formats of type-grouped operators: binary operators and unary operators.

Binary Operators

A binary operator takes in 2 inputs of identical type using infix notation:

```
expr binop expr
```

Unary Operators

A unary operator takes in 1 input in prefix notation:

```
unop expr
```

Table 3.2: Operators

<table>
<tr><th>Prec.</th><th>Type</th><th>Symbol</th><th>Function</th><th>Inputs</th><th>Assoc.</th></tr>
<tr><td>0</td><td></td><td>.</td><td>Child access</td><td>2</td><td>left</td></tr>
<tr><td>1</td><td></td><td>(type)</td><td>Cast</td><td>1</td><td>right</td></tr>
<tr><td rowspan="10">2</td><td rowspan="6">integer, float</td><td>-</td><td>Negation</td><td>1</td><td>right</td></tr>
<tr><td>*</td><td>Multiplication</td><td>2</td><td>left</td></tr>
<tr><td>/</td><td>Division</td><td>2</td><td>left</td></tr>
<tr><td>%</td><td>Remainder</td><td>2</td><td>left</td></tr>
<tr><td>+</td><td>Addition</td><td>2</td><td>left</td></tr>
<tr><td>-</td><td>Subtraction</td><td>2</td><td>left</td></tr>
<tr><td>string, list</td><td>^</td><td>Concatenation</td><td>2</td><td>left</td></tr>
<tr><td rowspan="3">boolean</td><td>!</td><td>Logical Not</td><td>1</td><td>right</td></tr>
<tr><td>&&</td><td>Logical And</td><td>2</td><td>left</td></tr>
<tr><td>||</td><td>Logical Or</td><td>2</td><td>left</td></tr>
<tr><td rowspan="4">3</td><td rowspan="3">integer, float, bool</td><td>==</td><td>Equals</td><td>2</td><td>left</td></tr>
<tr><td>></td><td>Greater than</td><td>2</td><td>left</td></tr>
<tr><td><</td><td>Less than</td><td>2</td><td>left</td></tr>
<tr><td></td><td>~=</td><td>Type Equals</td><td>2</td><td>left</td></tr>
<tr><td>4</td><td></td><td>=</td><td>Assignment</td><td>2</td><td>right</td></tr>
</table>

3.7.2 Untyped Operators

There are 3 operators that cannot be grouped by type: child access, cast, and assignment.

Child Access

Child access is an binary operator that takes an expression *expr* that evaluates to a typedef instance variable *id* on the left and a child identifier id_c on the right (id_c must be a declared child of the typedef variable structure $\$Id$ used to declare *id*). This operation outputs the value of the child id_c from the *id* variable on the left.

Cast

Cast is a unary operator that takes an expression *expr* on the right and (if compatible) converts its value to the *type* within the cast symbol. The following are valid types of casts:

1. integer to float
2. float to integer
3. integer, float, bool, or regex to string
4. user-defined type Id to its associated predefined type *type* (or compatible by rules 1, 2, or 3)
5. type *type* to user-defined type Id, when Id is defined as associated with type *type* (or compatible by rules 1, 2, or 3)
6. user-defined type Id_1 to user-defined type Id_2, when both are defined as associated with the same predefined type *type* (or compatible by rules 1, 2, or 3)

A variable can accommodate multiple user-defined types at once (as long as they are of the same associated predefined type). Casting a user-defined type to its associated built-in type causes it to no longer be of the user-defined type.

Type Equals

Type equals is a binary operator that takes an expression on its left and a user-defined type on its right. The return value is *true* if the expression is of the user-defined type and *false* if it is not.

Assignment

Assignment is a binary operator that takes an variable declaration statement on the left and an initialization expression *expr* on the right. It assigns *expr* to the variable declared on the left and outputs the value of *expr*. See 3.8 Data Variables, 3.9.1 Function Assignment, and 3.10 Type Variables for more information on variable assignment

3.8 Data Variables

Data variables are used to store literal data in an allocated place in memory to be accessed later. Declaration and initialization of data variables must occur concurrently. Because of this, variable names do not need a type declaration, since they will take on the type of the expression being assigned. Thus, (almost trivially) the format for data variable assignment is as follows:

```
id = expr
```

Data variables are stand-ins for literals of the values being assigned to them. Therefore, if a data variable is assigned a constant value, any use of that variable value will remain unchanged, even if the variable is assigned another value later. However, if a variable is assigned an mutable value, the use of that variable may change in value if the variable is reassigned another value or if the underlying data object is changed. If a variable is reassigned to a new value, that value must be of the same built-in type (a variable assigned a user-defined type can be reassigned to a value of the user-defined type's underlying built-in type, and vice versa).

One special case of a data variable is the typedef instance variable. Typedef instance variables are declared from user-defined typedef variables. Typdef instance variable assignment varies greatly from the initialization pattern above. Typedef instance variables must be preceded by a declaration of their defining typedef $\$Id$ and can only be assigned from a *decllist* of child variable assignments corresponding $\$Id$'s variable structure. However, after declaration and initialization, typedef variables are treated the same as complex data variables (see 3.10.2 TypeDef).

3.8.1 Primitive Data Types

Integers, floats, booleans, strings, and regexes are primitive data types, meaning they are the most basic data structures available. Ints, floats, and booleans are treated as constant values. Strings and regexes are mutable. A variable that holds a primitive data type is equivalent to a literal of that data type.

3.8.2 Complex Data Types

Lists and dictionaries are complex data types, meaning that they contain elements of other data types (including complex data types and functions). A complex data variables can be initialized either with a complex data type literal (see 3.6.6 List Literals and 3.6.7 Dictionary Literals) or with a modification to or copy of an existing complex data variable. Complex data types are mutable and can change in value over the course of a program, but standard library functions encourage users to fold or map complex data structures into new data structure elements.

3.9 Function Variables

A function variable is used to store a function block into a variable identifier. The identifier is simply a pointer to the function literal. It can be called or passed to a map or fold standard library function. Once an identifier is assigned a function, it cannot be reassigned. Function variables cannot be passed to or returned from user-defined functions.

Functions have ≥ 0 possible data types in the function arguments and exactly 1 output type.

3.9.1 Function Assignment

The function declaration and initialization is the same as with data variables and is as follows:

```
id = expr
```

The following is an example function the trivially outputs the value that was passed to it:

```
myFun = <int x:int>{ x; }
```

3.9.2 Function Call

A function call is the utilization of a function variable (whether user-defined or part of the standard library) to evaluate the function block of that variable on a specific input. To use a function call, one must provide input that exactly matches the function definition's formal arguments. The function call evaluates the identified function on that specific input and provides the output.

Function call format is as follows:

```
id(exprlist)
```

Where *exprlist* is a series of ≥ 0 expressions separated by commas:

$$exprlist \rightarrow expr,\ exprlist \mid expr \mid \epsilon$$

The length n of *exprlist* must exactly match the length of $formallist$ of *id*. Ordinally, the type of *expr* at position i of *exprlist* must match the *type* at position i of $formallist$ of *id*.

3.10 Type Variables

3.10.1 Type

A type variable holds a set of user-defined types that build upon existing data types. The variable *id* encompasses the entire set, while each type defined is considered a member of that set. Each type member must have an associated built-in data type that defines the type of data to be stored within this new user-defined type. These type members set can be used as any built-in data type (variable declaration and type matching) and can be initialized in the same way as their associated predefined data type. A value is set to a user-defined type using the cast operator. User types are purely aesthetic and allow the user to categorize variables according to their own designations. User types do not functionally change the value being categorized.

Type variable declaration and type member set initialization must occur simultaneously. The type variable id is uncapitalized, while the type members defined within the type set must be capitalized. Type variables and user types are immutable and cannot be overriden after declaration.

Type declaration and initialization is as follows:

```
type id = { Typelist }
```

Where *Typelist* is a series of ≥ 1 type member declarations:

$$\textit{Typelist} \rightarrow \textit{Id}<\textit{type}>, \textit{Typelist} \mid \textit{Id}<\textit{type}>$$

Example type declaration and initialization:

```
type foo = { Bar<int> | Baz<bool> | Bap<int> }
```

3.10.2 TypeDef

A typedef is a user-defined structured object type containing ≥ 0 children variables. Creating a typedef consists of two steps: (1) naming and defining the typedef and (2) declaring and initializing a variable instance of the typedef.

TypeDef Definition

Defining a typedef consists of naming the typedef *Id* and defining its structure by declaring its ≥ 1 children variables. Each child is declared by giving its type and id (each child must have a unique id within the typedef). Upon definition, a typedef is simply a template: it does not hold any actual data (i.e. no memory is taken up to hold the children variables).

Typedef definition is as follows:

```
typedef $Id = { decllist }
```

Where *decllist* is a series of ≥ 1 *type id* variable declarations:

$$\textit{decllist} \rightarrow \textit{type id};\ \textit{decllist} \mid \textit{type id};$$

TypeDef Variable Assignment

Declaring and initializing a typedef variable consists of using the typedef $\$Id$ in the declaration of a new variable. This instance variable is declared with an id of the form *id* and is initialized by assigning values to all the children declared as the structure of that typedef. (More explicitly: for each child c_i of a given typedef definition, the typedef instance variable must initialize an associated child c_i with an expression that matches the declared type of c_i in the typedef definition.) Once a typedef instance variable is declared and initialized, its type is of $\$Id$ (its defining typedef).

To declare and initialize a typedef variable, the variable declaration and children initialization must occur simultaneously in the following format:

```
$Id id = { childreninit }
```

Where *childreninit* is a series of ≥ 1 $id = expr$ assignments:

$$\textit{childreninit} \rightarrow \textit{id} = \textit{expr};\ \textit{childreninit} \mid \textit{id} = \textit{expr};$$

The use of typedef instance variables is identical to that of complex data variables: they are immutable, they are overwritable (old memory discarded once no longer in use), and they are passed by reference.

After a typedef instance variable is created, its children may be accessed with the child access operator along with a specific child's id:

```
id.id
```

Example typedef declaration, data variable declaration and initialization, and data access:

```
typedef $MyDef = { string s; float f; };
$MyDef myPi = { s = 'pi'; f = 3.141592 };
myPi.f; # outputs 3.141592
```

3.11 Control Flow Expressions

A block expression is a special type of expression used to create new scopes and determine control flow of expressions. There are 4 types of block expressions: function, match, if-else, and do-while. Every block contains at least one series of ≥ 1 statements (i.e. a *stmtblock*).

3.11.1 Variable Scope

A block expression has access to any variables defined in any blocks that it is enclosed within (i.e., it has access to its parent's variables, its parent's parent's variables, and so on). Any variables defined within a given *stmtblock* of a block are limited to use in the scope of that *stmtblock*. If a variable in a block has an identical id to a variable from a higher scope, the value of the lower-scoped variable is used (the higher-scoped variable is "hidden" by the lower-scoped variable) until the end of its scope is reached, at which point the higher-scoped variable becomes visible.

3.11.2 Expression Value

A block's expression value is defined as the last evaluated expression reached within the block. The type of the block's output is defined at the beginning of the block immediately after the block keyword.

3.11.3 Function Block

A function literal is a block expression that contains one *stmtblock*. A function literal can be utilized as an anonymous function or assigned to a function variable to be executed at a later point. The evaluated output of this function block is determined upon function call. Function literals are described in 3.6.8 Function Literals. Creating function variables to store function blocks and using function calls to evaluate function blocks are described under 3.9 Function Variables.

3.11.4 Match

A match is a value-matching or type-matching control flow expression. A match takes in an expression and matches it to a block that is evaluated as the expression value of the match. A match takes the following format:

```
match:type (expr) matchchoice
```

Where *type* is the output type, *expr* is an input expression and *matchchoice* has two options: matching the input *expr* by value or by type.

$$\begin{aligned} matchchoice &\rightarrow \text{byvalue } \{ \; valuematchlist \; \} \\ &\rightarrow \text{bytype } \{ \; typematchlist \; \} \end{aligned}$$

A *valuematchlist* is a series of ≥ 1 blocks to be matched with by value. A *typematchlist* is a series

of ≥ 1 blocks to be matched with by user-defined type. They take the following form:

$$\begin{aligned} \textit{valuematchlist} &\rightarrow \textit{valuesub-block};\ \textit{valuematchlist} \mid \textit{valuesub-block}; \\ \textit{valuesub-block} &\rightarrow \textit{valuematchexpr}\ \{\ \textit{stmtblock}\ \} \\ \textit{valuematchexpr} &\rightarrow \textit{expr} \mid \text{default} \end{aligned}$$

$$\begin{aligned} \textit{typematchlist} &\rightarrow \textit{typesub-block};\ \textit{typematchlist} \mid \textit{typesub-block}; \\ \textit{typesub-block} &\rightarrow \textit{typematchexpr}\ \{\ \textit{stmtblock}\ \} \\ \textit{typematchexpr} &\rightarrow \textit{Id} \mid \text{default} \end{aligned}$$

In a *valuematchlist*, a match matches the input value of *expr* with a block in *valuematchlist*. The *valuematchexpr* of each block in *valuematchlist* is either an *expr* that must be of the same type as the input *expr* OR the value `default`, which will match with any input *expr*.

In a *typematchlist*, a match matches the type of the input *expr* with a block in *typematchlist*. The *typematchexpr* of each block in *typematchlist* is either a user-defined type *Id* OR the value `default`, which will match with any input *expr*.

In either value- or type-matching, if multiple blocks are matched with the input *expr*, the block that is listed first will be the block matched. Also in either case, the last block must be a default block.

Once a block is matched, the *stmtblock* of that block is evaluated and output as the expression value of the encompassing match.

3.11.5 If-Else

An if-else is a pair of 2 blocks, one of which will be evaluated depending on the result of a boolean expression. Whichever block is evaluated is output as the expression value of the entire if-else. The format of an if-else is as follows:

```
if:type (expr) { stmtblock } else { stmtblock }
```

Where *type* is the output type and *expr* is a boolean expression. If *expr* evaluates to `true`, the first block's *stmtblock* is evaluated. If *expr* evaluates to `false`, the second block's *stmtblock* is evaluated.

3.11.6 Do-While

A do-while is a single block that is performed once and then repeatedly while a given boolean expression is evaluated as `true`. The last expression reached in the do-while *stmblock* is evaluated as the output expression value of the do-while loop. The format of a do-while is as follows:

```
while:type (expr) { stmtblock }
```

Where *type* is the output type and *expr* is the conditional boolean expression.

3.12 Program Structure

A CLL program consist of an series of ≥ 1 statements evaluated in-order until end-of-file (EOF) is reached. Statements include expressions, type variable definition, and typdef variable definition.

On the top-level of a CLL program, variable assignment is generally the most useful top-level expression statement, as every expression at the top-level of a CLL program has its value discarded.

3.12.1 Production Rules

The following production rule set gives an overview of an entire CLL program grammar:

$$
\begin{aligned}
start &\rightarrow stmtblock \text{ EOF} \\
stmtblock &\rightarrow stmt;\ stmtblock \mid stmt; \\
stmt &\rightarrow expr \mid tassign \mid tdassign \\
tassign &\rightarrow \text{type } id = \{Typelist\} \\
&\text{(...skipping Typelist productions...)} \\
tdassign &\rightarrow \text{typedef } id = \{decllist\} \\
&\text{(...skipping decllist productions...)} \\
expr &\rightarrow (expr) \\
&\rightarrow litint \mid litfloat \mid litbool \mid litstring \\
&\rightarrow litregex \mid litlist \mid litdict \mid litfun \mid litnull \\
&\rightarrow binop \mid unop \mid childaccess \mid cast \mid assign \\
&\rightarrow match \mid ifelse \mid while \mid id \\
&\text{(...skipping lit* productions...)} \\
&\text{(...skipping binop, unop productions...)} \\
&\text{(...skipping childaccess, cast productions...)} \\
assign &\rightarrow type\ id = expr \\
type &\rightarrow \text{int} \mid \text{float} \mid \text{bool} \mid \text{string} \mid \text{regex} \\
&\rightarrow \text{list}<type> \mid \text{dict}<type,type> \\
&\rightarrow \text{fun}<formallist{:}type> \mid Id \\
&\text{(...skipping match, ifelse, while productions...)}
\end{aligned}
$$

All skipped productions have been defined previously in this document.

3.13 Standard Library

The following are all built-in functions that allow for ease of operation upon data.

3.13.1 String

`sprint<string s : int>`

Prints the string `s`

`sfold<fun f <type x, string s : type>, type x, string s : type>`

Outputs an accumulated variable `x` after applying `f` to `x` and each character of `s`

`ssize<string s : int>`

Outputs the length of `s`

3.13.2 Regex

`rematch<regex r, string s : bool>`

Outputs true if `r` matches `s`, otherwise false

resub<regex r, string s, string t, int n : string>

Outputs `s` with all matches to `r` in group `n` replaced by `t` (note: group 0 is the whole regex, group 1 is the first grouping, and so on).

3.13.3 List

ladd<list<*type*> l, *type* x : list<*type*>>

Outputs `l` with `x` added to the beginning of `l`

ladd<list<*type*> l, *type* x, int n : list<*type*>>

Outputs `l` with `x` added to position `n` of `l` (if `n` is $\geq$ the length of `l`, `x` is placed at the end of `l`)

lfold<fun f<*type'* a, *type* x : *type'*>, *type'* a, list<*type*> l : *type'*>

Outputs an accumulated variable `a` after applying `f` to `a` and each element `x` of `l`

lget<list<*type*> l : *type*>

Outputs last element of `l` (throws runtime error if empty)

lmap<fun f <*type* x : *type*>, list<*type*> l : list<*type*>>

Outputs `l` with `f` applied to each element of `l`

lmem<list<*type*> l, *type* x : bool>

Outputs true if `x` is a member of `l`, otherwise false

lremove<list<*type*> l : list<*type*>>

Outputs `l` with the first element removed (throws runtime error if empty)

lremove<list<*type*> l, int n: list<*type*>>

Outputs `l` with element `n` removed (if `n` is $\geq$ than the length of `l`, then the last element of `l` is removed) (throws runtime error if empty)

lsize<list<*type*> l : int>

Outputs number of elements in `l`

3.13.4 Dictionary

dadd<dict<*type,type'*> d, *type* x, *type'* y : dict<*type,type'*>>

Outputs `d` with the key-value `x:y` pair added

dfold<fun f<*type''* a, *type* k, *type'* v: *type''*>, *type''* a, dict<*type*,*type'*> d : *type''*>

Outputs an accumulated variable `a` after applying `f` to `a` and each key-value pair `k` and `v` of `d` (order of elements is not defined)

dget<dict<*type,type'*> d, *type* x : *type'*>

Outputs value of `d` key in `x` (throws runtime error if not found)

dmap<fun f<*type* k, *type'* v: *type'*, dict<*type*,*type'*> d : dict<*type*,*type'*> >>

Outputs d with f applied to each key-value pair k and v of d (order of elements is not defined)

dmem<dict<*type*,*type'*> d, *type* x : bool>

Outputs true if x is a key of d, otherwise false

dremove<dict<*type*,*type'*> d, *type* x : dict<*type*,*type'*>>

Outputs d with x's key-value pair removed (throws runtime error if key not found)

dsize<dict<*type*,*type'*> d : int>

Outputs number of key-pair elements in d

dkeys<dict<*type*,*type'*> d : list<*type*>>

Outputs a List of type *type* containing all the keys of d

3.14 Example Program

```
words = stdin;
type verb = { Root<string>, Infinitive<string>, Present<string>, Past<string>, Future<
    string> };

typedef $Stems = {
   string inf;
   string pres;
   string past;
   string fut;
};

typedef $VerbDef = {
   Infinitive inf;
   Present pres;
   Past past;
   Future fut;
};

$Stems stems = {
   inf = 'i';
   pres = 'as';
   past = 'is';
   fut = 'os';
};

conj = <string,regex>{ stems.inf: "[a-z]+(i)$", stems.pres: "[a-z]+(as)$", stems.past: "[a
    -z]+(is)$", stems.fut: "[a-z]+(os)$" };

root = <list<Root> l, string w : list<Root>> {
   w = match:string (true) byvalue {
      rematch(dget(conj, stems.inf), w) {
         resub(dget(conj, stems.inf), w, '', 1);
      }
      rematch(dget(conj, stems.pres), w) {
         resub(dget(conj, stems.pres), w, '', 1);
      }
      rematch(dget(conj, stems.past), w) {
         resub(dget(conj, stems.past), w, '', 1);
      }
      rematch(dget(conj, stems.fut), w) {
```

```
            resub(dget(conj, stems.fut), w, '', 1);
        }
        default {
            '';
        }
    };
    if:list<Root> (ssize(w) > 0) {
        ladd(l,(Root)w);
    } else {
        l;
    };
};

create_verb = <dict<Root,$VerbDef> d, Root w: dict<Root,$VerbDef>> {
    $VerbDef v = {
        inf = (Infinitive)((string)w ^ stems.inf);
        pres = (Present)((string)w ^ stems.pres);
        past = (Past)((string)w ^ stems.past);
        fut = (Future)((string)w ^ stems.fut);
    };
    dadd(d, w, v);
};

words = lfold(root, <Root>[], words);
verb_dict = lfold(create_verb, <Root,$VerbDef>{}, words);

type pronoun = { First<string>, Second<string>, Third<string>, Singular<string>, Plural<
    string>, Fem<string>, Masc<string> };
typedef $PronounDef = {
    list<First> fst;
    Second snd;
    list<Third> thrd;
};
type pronounCase = { Personal<$PronounDef>, Accusative<$PronounDef> };

first_suffix = <list<First> l, string suf : list<First>> {
    addsuf = <First p:First> {
        (First)((string)p ^ suf);
    };
    lmap(addsuf, l);
};

third_suffix = <list<Third> l, string suf : list<Third>> {
    addsuf = <Third p:Third> {
        (Third)((string)p ^ suf);
    };
    lmap(addsuf, l);
};

Personal personal = {
    fst = <First>[(First,Singular) 'mi', (First,Plural) 'ni'];
    snd = (Second,Singular,Plural) 'vi';
    thrd = <Third>[(Third,Singular,Fem) 'si', (Third,Singular) 'gi', (Third,Plural) 'ili
        '];
};

suf = 'n';

Accusative accusative = {
    fst = first_suffix(personal.fst, suf);
    snd = personal.snd ^ suf;
    thrd = third_suffix(personal.thrd, suf);
};
```

```
pronoun_list = <$PronounDef>[personal, accusative];

typedef $Language = {
    dict<Root,$VerbDef> verbs;
    list<$PronounDef> pronouns;
};

$Language myLang = {
    verbs = verb_dict;
    pronouns = pronoun_list;
};

print_lang = <$Language lang:int> {
    print_verbs = <Root r, $VerbDef v :$VerbDef>{
        sprint('Root: ' ^ (string)r);
        sprint('Infinitive: ' ^ (string)v.inf);
        sprint('Present: ' ^ (string)v.pres);
        sprint('Past: ' ^ (string)v.past);
        sprint('Future: ' ^ (string)v.fut);
        v;
    };
    dmap(print_verbs, lang.verbs);
    print_pronouns = <$PronounDef p:$PronounDef>{
        sprint('First Person:');
        print_first = <First f : First>{
            sprint((string)f);
            f;
        };
        lmap(print_first, p.fst);

        sprint('Second Person:');
        sprint((string)p.snd);

        sprint('Third Person:');
        print_third = <Third t : Third>{
            sprint((string)t);
            t;
        };
        lmap(print_third, p.thrd);
        p;
    };
    lmap(print_pronouns, lang.pronouns);
    0;
};
print_lang(myLang);
```

3.14.1 Input

```
sendi
lerni
havi
helpi
esti
kanti
pano
pilko
auto
```

3.14.2 Output

```
Root: help
Infinitive: helpi
Present: helpas
Past: helpis
Future: helpos
Root: lern
Infinitive: lerni
Present: lernas
Past: lernis
Future: lernos
Root: send
Infinitive: sendi
Present: sendas
Past: sendis
Future: sendos
Root: hav
Infinitive: havi
Present: havas
Past: havis
Future: havos
Root: kant
Infinitive: kanti
Present: kantas
Past: kantis
Future: kantos
Root: est
Infinitive: esti
Present: estas
Past: estis
Future: estos
First Person:
mi
ni
Second Person:
vi
Third Person:
si
gi
ili
First Person:
min
nin
Second Person:
vin
Third Person:
sin
gin
ilin
```

Chapter 4

Project Plan

4.1 Development Process

4.1.1 Initial Stages

The initial stages of this project were composed primarily of my searching for inspirations for my language's potential use cases and deciding how to best accommodate the needs of my desired end user. I researched into past projects done for the class and tried to find areas of my own life where I might want a programming language specifically designed for that task. After deciding the use-case source of inspiration — easing conlang creation — I began writing mock-up programs that featured elements I would like to have in my programming language. After coalescing elements of these programs into a systematic whole, I wrote out a first draft of the grammar tree to be used by my language. This first grammar tree served as the jumping off point for writing a first draft of my LRM.

4.1.2 Writing the LRM

Before touching one line of code for the compiler, I wanted to ensure my LRM was as well-defined as possible to avoid excessive efforts in coding a feature only to realize it was poorly design or a bad idea altogether. As I wrote the LRM for this language, I adjusted the grammar tree and redesigned language features to make the language more effective and reduce the amount of work needed to be done to achieve the features I desired. I reference lecture materials, the microc language, and past semester LRMs for inspiration on what features I would like to include in (and exclude from) my language. Through this process, I kept in mind end-goal use to ensure that I was creating the most compact language I could while still retaining the functionality needed to serve my imagined end user.

4.1.3 AST and SAST

Against better judgement, I wrote out the entirety of the AST and then the entirety of the SAST for my desired programming language before I even started the LLVM code generation portion of my compiler. While this worked out in the end, I realized after I had done this that the better route would have been to write out each language feature from front to back, as incremental coding makes bug finding and fixing much easier. The actual process of writing out the AST and SAST was fairly easy, but I was unable to properly debug the language features because I couldn't compile any programs yet.

4.1.4 LLVM Code Generation

When I started the code generation portion of this project, I was initially overwhelmed by the difficulty of understanding the LLVM features being created by the opam bindings and the amount

of assembly refreshing I needed to think through the implementation of feature generation. There were also times when I was unsure if the code I was writing was interpreting rather than compiling. Additionally, with each new line of code I wrote, I continued to get errors regarding type mismatch, segfaults, and instructions not "dominating all use cases!". However, at a certain point, I gained momentum in understanding and began to get on a roll with implementation (generating LLVM of C files definitely helped too). After making some progress, I realized that certain features that I thought I would implement using OCaml (e.g. hashtables, regex functionality, garbage collection) I could actually more easily implement using C. I created C files for these more complicated tasks and linked them in as library files. I then worked through each feature of the language, building upon existing features when seemed fit (e.g. why make an empty list from scratch when I could just evaluate a dummy `SListLit`?).

4.1.5 Automation

As I made progress in code generation, I quickly realized the need to automate program generation and testing. I used a Makefile to compile the native compiler file, compile C library files, clean up build files, and run a test script. I then later created bash scripts to compile CLL program files with the compiled native and library files. The last bit of automation I created was a script to generate "out" files for each test file written, which I would then compare to test program output using the initial test script.

4.2 Style Conventions

- Each indentation consists of 4 spaces
- Variable names use `snake_case`
- Variable names should be repetitious as possible while still conveying accuracy
- Variables ending in `'` or `_` represent derivations of an existing variable
- Liberal use of empty lines to create visual "blocks" of related code within larger areas
- The OCaml keyword `in` should always be at the end of a line or on its own line, never the beginning (unless repetitious lines make this the easiest way to write the code)
- Brackets and parentheses used to denote a lengthy expression or block should either match on the same line or place the beginning bracket at the end of preceding line and end bracket at beginning of the following line

4.3 Project Timeline

Milestone	Date
Began brainstorming	20 March
First commit	30 March
Scanner and AST	5 April
Parser	7 April
SAST	8 April
Semant	14 April
Started test automation	15 April
Started Codegen	16 April
Hello World	21 April
Finished Codegen	10 May
Final report & presentation	12 May

4.4 Roles and Responsibilities

Since I completed this project myself, I thought I might use this section to reflect on the experience of undertaking this project as an individual. I avoided most issues encountered when working on a group project (e.g. decision-making conflicts, distribution of work, group coordination). However, this meant that I didn't have means of "sanity checking" throughout the process. As a single individual, I am prone to blindness to my own errors and may pursue strategic routes that would be better completed in another fashion. I regularly consulted OCaml, C, and LLVM resources and tutorials online to understand recommended programming practices and conventions to help mitigate these issues. While I do feel like I missed out on an integral part of the project process by working individually, I also know I increased my knowledge in all areas of the compiler creation process in ways I might have avoided if I could delegate work to someone else.

4.5 Software Development Environment

- **OS**: macOS Catalina 10.15.7
- **Shell**: Zsh 5.7.1
- **OCaml**: 4.12.0
- **Opam LLVM**: 11.0
- **GCC/Clang**: 12.0.0
- **Git**: 2.24.3

4.6 Project Log

```
commit f7f96fb36a494ee5a97fd6b307ac9e490be4549a (HEAD -> master)
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Wed May 12 17:47:43 2021 -0400

    clean cll.native, addition to README

commit f45e4c78148df2cd28f4ccf51b854e551c0147b8
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Wed May 12 17:14:48 2021 -0400

    added author signature, last minute changes

commit 26f640d0cd89f4dc26d245291a9f1680c7597b82
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
```

```
Date:   Wed May 12 13:48:41 2021 -0400

    more tests, fixed dmap bug in semant

commit 9029b6dfacb588b966d432b4f232cae2e0deb2cb
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Wed May 12 12:35:51 2021 -0400

    removed "dont_test" directory

commit 624ae9f3e6ad90e402666d0a351128a2823e7f60
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Wed May 12 12:33:03 2021 -0400

    added fail tests back, improved testing output

commit 914ac0c394ed5645802895e0b0c12f1ac9ce3639
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Wed May 12 11:43:00 2021 -0400

    scanner error uses line num

commit e3e193da13d8880b051ffe0bb81b949a0ce18559
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Wed May 12 11:17:56 2021 -0400

    variable must be reassigned to same associated type

commit af861302a944372a4559ff718584bad1af850254
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Tue May 11 23:14:03 2021 -0400

    fixed string key bug in user-typ dict

commit 9f9d6221e69e5793f3555a03bc10eb12a93f7090
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Tue May 11 16:02:02 2021 -0400

    implemented stdin

commit 9c1441e60299bc553ab1d36114b7d3bdaa90bc5e
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Mon May 10 21:26:20 2021 -0400

    garbage

commit c050b3aaa02131ab3cd1f770f52228520b599a2d
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Mon May 10 21:25:12 2021 -0400

    LRM

commit 9d2a07150f3d8ed56577c5236779ff6cb75b08ad
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Mon May 10 20:31:48 2021 -0400

    funs cannot be stored/passed

commit 62e3cbefe43b51b10ae3ffc925ccde99ed89241f
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Mon May 10 20:11:07 2021 -0400

    readme
```

```
commit 265889ca46a87d57ec9e675d80211e35e6ec7351
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Mon May 10 20:08:20 2021 -0400

    language reference manual

commit bdde2587c878ec703893058684b93c85abec215a
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Mon May 10 19:57:54 2021 -0400

    malloc usertypdef, add example program test

commit dce7f292993f2151e32ea2b2646b06eff47e253c
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Mon May 10 16:38:46 2021 -0400

    changed less than and greater than symbols

commit 86ad303aee1ad5842759f00fb525459e3efa48c0
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Mon May 10 16:13:35 2021 -0400

    changed single line comment to just '#'

commit c0fcfd6d689cb14f3e398cd58f1bacbe244f446d
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Mon May 10 16:09:57 2021 -0400

    remove unused variables, make_exe for non-test files, added to readme

commit b857760e72602de5d1716fef70409a60572e5a1e
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Mon May 10 15:31:00 2021 -0400

    change while to dowhile, change null returns to exit(1), remove some comments

commit 75614dd3900d87d01cd2a2f97b5a9139c9878a7e
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Mon May 10 15:14:06 2021 -0400

    removing printf statement from malloc manager

commit d52c617aa1fb9adfb39936aa9d2ae83c15f87e7a
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Mon May 10 15:12:48 2021 -0400

    malloc manager in ll too

commit b4e58e4caac6da010a9f4603d14ff4acfbf8bc4a
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Mon May 10 15:09:56 2021 -0400

    malloc manager working in codegen and hashtable

commit 06f342f4416d3f74ddd0d9ce445a9d6ea1a5a251
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Mon May 10 13:43:59 2021 -0400

    added malloc manager, change builder issue with lget

commit 4b7e738104e3e08881c50c2136d84c126dcadfe0
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
```

```
Date:   Mon May 10 01:47:59 2021 -0400

    safe malloc/hashtable of malloc'd addressed created... now just need to put everywhere
         mallocs happen

commit 1846ffe891c259b07b84ad57f69baacdb31a43f9
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Fri May 7 16:30:34 2021 -0400

    match bytyp implemented

commit 05b5f449046fd7da5f144a893742a83b94a63b95
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Fri May 7 02:14:24 2021 -0400

    implemented typcomp, but not super flexible

commit 13d836a91b53d5811035c3802f44d9a4f6b82c2b
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Fri May 7 00:39:51 2021 -0400

    bug fixes (new restrictions, listlit, regex, small scope stuff)

commit 6d0e8584239a5f4194bf87702d45b8ea11c309e5
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Fri May 7 00:35:28 2021 -0400

    add restrictions to semant

commit 04f379f6386a62f442a3fd213341278f0ed9dd9e
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Thu May 6 19:55:21 2021 -0400

    fixing key getting issue with hashtable

commit 1d918e3cf64f3f6b60caa73ce1e4c48ffb88937b
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Thu May 6 18:56:24 2021 -0400

    dmap should be working with parent var scope now

commit 525322e2240b19f15f9067ce86e61f115f7274f9
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Thu May 6 18:43:34 2021 -0400

    lfold parent vars passing correctly, not with dfold yet

commit 86f5350231496b7fbe2f82a9a4fc05cce83b3d79
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Thu May 6 15:48:07 2021 -0400

    concat list implemented, fixed naming issue for stdlib function args

commit 824c3056ff015dd1eb4f9c61d0a9d11534631aa8
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Thu May 6 15:14:50 2021 -0400

    init params everywhere, lmap and dmap need passing, started list concat

commit 77a4277b231a3cf6e9b93e0e6aeea13e130acedb
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Thu May 6 14:09:21 2021 -0400
```

```
    variables passing to inner scope, but if variable changes, it changes previously
        defined functions

commit 5c6e013fd518c8dd3729dbd3bcab56c8446106ed
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Wed May 5 15:22:07 2021 -0400

    dkeys implemented

commit ff2382ae60b7afe3b92465b085d0851ac0a07cc3
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Wed May 5 13:58:45 2021 -0400

    dmem implemented (and added generated types for lmem)

commit 501c47fa9a7d735e64fac729046c0600e4a15b3f
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Wed May 5 13:30:37 2021 -0400

    remove leftover utdid stuff

commit aadfec58fb280dbd1596ffb7ffeb4038822388bb
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Wed May 5 13:29:12 2021 -0400

    lmap implemented

commit c65f9318ee098d36c44fc6d5e6cb1b7b3ddfb97d
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Wed May 5 10:45:01 2021 -0400

    test for unicode

commit 0d9080c6fca9e9b32e340d0c2a20423d8630a9fa
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Wed May 5 00:41:22 2021 -0400

    dmem implemented

commit ffa83382198bff2b9358e6569e1776dd6a21da37
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Wed May 5 00:24:53 2021 -0400

    lmem implemented

commit 9915aae7e2b2d4d575ef735862f2abc10b77e077
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Tue May 4 23:18:35 2021 -0400

    not gonna do null/none stuff

commit 09deec42f32f4066963dab86b56b4fc618b3df90
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Tue May 4 23:13:42 2021 -0400

    resub implemented

commit 05523415d9d1ee77c86d5d4af0bbfdfff82ca213
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Tue May 4 21:43:41 2021 -0400

    relit and rematch implemented
```

```
commit 7e2c38b6aad91565a7071999e449cd2aaba75fdd
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Tue May 4 20:08:42 2021 -0400

    variable arguments for ladd and lremove

commit 435457521f145dbdc04f5a6f78d203340a9db583
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Tue May 4 18:39:37 2021 -0400

    ladd implemented

commit c0de2b081ae9ccdc7b6b4ab215663b9d5ac47365
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Tue May 4 18:02:21 2021 -0400

    lremove implemented

commit 3df75eafa204ee9cce9a91d318579cdcd654d974
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Tue May 4 17:43:40 2021 -0400

    dremove implemented

commit dc3a563d4eabe31fab8e6fa600294d6acd6ccb01
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Tue May 4 16:46:46 2021 -0400

    dfold implemented

commit 4ae8e1cb53b134514c45b4cb146bdd3d62eee951
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Tue May 4 02:46:39 2021 -0400

    lfold implemented

commit 1661bd90795eac06e2b811ce6b545ea5ac189274
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Mon May 3 23:34:45 2021 -0400

    making sfold a function

commit 0fb2225738ad30e2507cbc591a51c163378d6bcd
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Mon May 3 23:18:26 2021 -0400

    some testing refinement

commit 4669d251c6dfb7f3b7465d6ee3a9189ff38bdff7
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Mon May 3 22:43:14 2021 -0400

    sfold implemented

commit 4b4c889fa820fc1059f33cb21959ceaa6ba0e408
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Mon May 3 18:25:39 2021 -0400

    ssize, lsize, dsize stdlib

commit be92a5f6b2763dca1ebe25230e8592d9cb4c1420
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Mon May 3 17:05:55 2021 -0400
```

```
    match by value implemented

commit 14d6704d8b00d2389e31e09f38dd2743fdf9c8af
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Mon May 3 15:44:24 2021 -0400

    unop implemented

commit 4060d2c47cd52b95b455b21daeadf71c05f51966
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Mon May 3 15:15:21 2021 -0400

    usertypdef child access works (and changed id names for typdef vars)

commit 700828c5601695e02157310ba9c31c3584d75c39
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Mon May 3 14:37:41 2021 -0400

    usertyp and usertyp def creation and assignment implemented

commit 5b016c82a40b04612b88a72a604294e5ee1f40e0
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Mon May 3 12:57:48 2021 -0400

    fixed hash issue for string keys

commit eeaeef3586b72a18fbabb50afd5029e756f7f43a
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Mon May 3 12:48:44 2021 -0400

    usertyp and cast working (still need to do usertypdef)... and dget segfaulting again?

commit 2574007bd177dd6d00a3f092fb088d7b5f70ca23
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Sun May 2 12:21:50 2021 -0400

    removing unnecessary comments

commit 025587c7cd189e3153ef89ddfc002fdc29ce2773
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Sat May 1 23:59:46 2021 -0400

    some cleaning up, test changes

commit fc89da456924d32c079cc983bc9f7d6c5e9c49f0
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Sat May 1 22:35:22 2021 -0400

    hash table checks string contents

commit ae60122600b004339095dea310cd1d5af75a81c4
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Fri Apr 30 01:49:39 2021 -0400

    changed memory copying for hash_table too

commit c2ee0da74fc8dcb7e5f31d40635c74b0c15e368e
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Fri Apr 30 01:36:20 2021 -0400

    fixed copying of data in linked list
```

```
commit c9983c47f24e69678fe10be482cc0f2dce47a531
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Thu Apr 29 21:04:20 2021 -0400

    lget is a function now

commit 17f08b93d5d2a3805cce95632e92d308e2dd6328
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Thu Apr 29 20:13:14 2021 -0400

    dget function implemented, but segfaults

commit 44ac80bf7000f9515110c5431bd97930454a93fc
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Thu Apr 29 19:54:02 2021 -0400

    dset now a function

commit bf905df431ee30777b5d673f9a1adc590a791676
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Thu Apr 29 17:41:44 2021 -0400

    fun passing and returning working

commit eea87af682e557caf5ec18f93f4d702fc3192d6e
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Thu Apr 29 11:21:07 2021 -0400

    functions with function pointers in args compile, but break when called

commit 5e9b57ac8f29ccec0583465e1d224ccb5dc284af
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Wed Apr 28 23:18:00 2021 -0400

    simplified function type def... but how to store/pass functions???

commit a9e9fbe5bde7c537e57c0f4be4d1f26b46a947a6
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Wed Apr 28 21:52:52 2021 -0400

    expression as function caller works (i.e. anonymous functions)... but ht_grow is being
         weird

commit de299061211de9aa9991451703346e21e827d94b
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Wed Apr 28 14:18:57 2021 -0400

    fun passing correctly (with int and string)

commit f8d4acc57e42f904328c3d265b2dda5f03ed12c1
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Wed Apr 28 14:04:26 2021 -0400

    fixed fun local variable issue

commit 190e7ff6e9a5688b8656f621b185a456d166101e
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Wed Apr 28 14:00:35 2021 -0400

    can access formal variables in fun, but issue with strings

commit 172529b816a323f14547c96a38fc1b906e474d75
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
```

```
Date:   Wed Apr 28 13:40:20 2021 -0400

    parent function passed to all stmt/expr

commit db1ff24928536b344e33b55f2e5307b600a33306
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Wed Apr 28 11:07:06 2021 -0400

    funlit working without args

commit 25bfd28e9af78df81b515881c5d985b9e6cdcafe
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Tue Apr 27 21:58:48 2021 -0400

    started fun lit, segfaults

commit ed76bfbcf4d0bcf9b713a4d8aae3d4b217a07a2c
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Tue Apr 27 21:05:08 2021 -0400

    made a better while test

commit d30e773c67d0ca385fe3ec7ed8eeca686b8c0d27
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Tue Apr 27 20:58:22 2021 -0400

    testing even sexier (compares with .out file)

commit a5bd7f3e0f933c367327b68a6a86bba0cad431d9
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Tue Apr 27 20:14:20 2021 -0400

    janky printing for list and dict, most dict errors fixed

commit 242a687e925ab5672eb34bfc1c756fb3f22ef254
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Tue Apr 27 17:35:25 2021 -0400

    ll_print, ht_print rename

commit 083c00c81f1bdbee54d24d1bedae0fffe91bcbc4
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Tue Apr 27 17:28:49 2021 -0400

    implemented lget

commit 2251cf64c8393c677bb0532f0bb3c839570be879
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Mon Apr 26 21:52:48 2021 -0400

    starting to add external linked list functionality

commit 9b14e3457439c538f4ea2618f59f39bfac9e40e6
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Mon Apr 26 21:25:57 2021 -0400

    string concat implemented

commit 268c2d962e4ffff158dad2dd91cb6dd0722d757c
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Mon Apr 26 20:25:07 2021 -0400

    binop for arithmetic implemented
```

```
commit 8c7bf61855e92daa31c61df6339686b46969b5d2
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Mon Apr 26 19:27:52 2021 -0400

    while implemented

commit bb2340c87967fe19b9c41b4269d365afe482f9b5
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Mon Apr 26 18:46:40 2021 -0400

    if/else working

commit b361561eda80f1f78c1864baf427146a18c9363a
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Mon Apr 26 18:12:33 2021 -0400

    if else implemented

commit 08514b4a6902ddc96034f9a1e47353739a7257ce
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Mon Apr 26 16:53:13 2021 -0400

    assign and id implemented

commit bf4ba2c7eb245e95634bf7d76e6f67f77dfac444
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Mon Apr 26 16:22:13 2021 -0400

    dset working

commit c7dd40b00358d6ea69dec4c7df523833ad1dddc7
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Mon Apr 26 16:14:45 2021 -0400

    added dset, but haven't tested yet

commit 9a361e8fbee2253b13e12a9744247725bdac0c76
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Mon Apr 26 13:45:41 2021 -0400

    print cleanup and testing adjustment

commit 1a02d8502818c7f31d87882c92df3991da9e7403
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Mon Apr 26 13:24:07 2021 -0400

    dget updates built-in functions when new dict type encountered

commit 16fa2b2f755511f2b7d9fd476cacbfce83687df0
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Mon Apr 26 00:23:22 2021 -0400

    fixed hash function -- dict working reliably with strings

commit 61b6928a814676d6214eae90c9a25c80b8881d2d
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Sun Apr 25 23:19:46 2021 -0400

    added dictlit (hashtable implementation) and dget... but weird behavior when executed

commit d71eaf4c58ce3ad39ae4e943339a9439ec3f9d16
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
```

```
Date:   Fri Apr 23 18:47:49 2021 -0400

    checking for duplicate keys in dict

commit 58c33a8acb9edf71b7c915ecc0d5c409c3f23014
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Fri Apr 23 14:35:42 2021 -0400

    list accomodates list and string

commit 0e896ac0737ee7600818d05d0feae65f9347677d
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Wed Apr 21 22:36:52 2021 -0400

    cleaned up printing/comments

commit 25e208c501ffd071c03c76b276f855388074f7fd
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Wed Apr 21 22:35:33 2021 -0400

    list lit expr

commit ba964ce1894b91ba50dba6a0e7935ca524734e83
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Wed Apr 21 18:40:03 2021 -0400

    remove quotes from string literal

commit 80daa43705605bcf2b1234ce22c394ec20edcb65
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Wed Apr 21 18:36:42 2021 -0400

    prints stringgit add .
    '''

commit 904e3a258e6bf58a8726d162c022d1c2551b8bf3
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Wed Apr 21 18:35:12 2021 -0400

    got rid of stupid file

commit 5159745c10f3be9a2d8bb067fce43401dde939f4
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Wed Apr 21 18:33:50 2021 -0400

    printing hello world (hard coded)

commit 2c1d73bf2ec4e0b2f342dad18cdc2dd1633e71a7
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Wed Apr 21 17:00:30 2021 -0400

    alloc string

commit 3c378567f7ad23606b7a9504c9fbd9759edfc440
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Wed Apr 21 16:54:07 2021 -0400

    codegen zsh file, updated testall and clean

commit d27f3c1d765bc984941b080c022e092b23d7d862
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Wed Apr 21 16:52:46 2021 -0400
```

```
    moved tests

commit df04580f628d3ad60ad917850b800db658dd284c
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Tue Apr 20 23:04:31 2021 -0400

    make module body, expr checking started

commit d10c21400cc1775afdf34b001fce9c666ee3346a
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Tue Apr 20 21:31:33 2021 -0400

    renamed elements of ast/sast type structures

commit 33e7b2aabb8b34f7d573e6769a06e5d513e78af6
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Tue Apr 20 21:29:54 2021 -0400

    removed unused var warnings for now, make sandbox directory

commit c00f8ddef48ebd0bd03c7541ec334cdc088b4f7e
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Fri Apr 16 17:23:11 2021 -0400

    init codegen stuff

commit c368ed714f0d8aef0d30a483aaf206481dfc126b
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Thu Apr 15 13:55:22 2021 -0400

    testall printing

commit 7345a895586983c8b8ffad889e27267a9351ec1a
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Thu Apr 15 13:32:55 2021 -0400

    updated Makefile, cleaning, and testing

commit 8328e05faceca9d5c4487f05ec369840ba4153e7
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Thu Apr 15 12:06:50 2021 -0400

    changed output extension for semantic error

commit 91d29048ab528ed0a9b73ffe0027281dc79c7e48
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Thu Apr 15 12:02:53 2021 -0400

    more tests

commit 47f48b11c0578f55f76e2348fd21f2b9ce318849
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Wed Apr 14 23:28:24 2021 -0400

    lexing error handling

commit 57c1146a297ae3d5ee2550eb9d1254bb2436c5c8
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Wed Apr 14 23:19:47 2021 -0400

    parsing line error

commit 3e01065e5a372ec549688b1dbb3510c64caad314
```

```
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Wed Apr 14 22:59:17 2021 -0400

    added line numbers for semantic checker

commit ae1870c1cf256c1f75edbdfa91911de189aae5ca
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Wed Apr 14 22:58:14 2021 -0400

    update gitignore

commit e8880e0c6b1b3438067b1eabbc7742fcdafc0dfd
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Wed Apr 14 21:46:43 2021 -0400

    update gitignore

commit dc92a48a5b2f39de315751728a1c8c7a2e1c08d1
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Wed Apr 14 21:34:09 2021 -0400

    moved tests, created 'fail' folder

commit 8ddf6d00a4a8b4c518118e462e038763a7caabdf
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Wed Apr 14 20:31:34 2021 -0400

    checked multiple statements for blocks, fixed statement order for funlit

commit 1909b191cec75c071a8beaa347d3432de8640b11
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Wed Apr 14 20:18:28 2021 -0400

    added nested type list/dict, fixed double angle bracket issue

commit 0c1d2c7dd8a5aebe1d0275e02f5103fa86a626d3
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Wed Apr 14 19:50:06 2021 -0400

    added LRM to readme

commit 7dd80d88c0928dc7fafc15cdb170799e435b1e15
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Wed Apr 14 18:17:08 2021 -0400

    fix priting for typdefassign, add test for negation

commit ba34d1f79fd0f22ba8c40f79c1168d0886d761b5
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Wed Apr 14 17:04:54 2021 -0400

    changed input to cll.native

commit e5536b534cb1cafa9c1b11cd3f461c83f2e6e876
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Wed Apr 14 16:41:59 2021 -0400

    better printing for testall, fixed error in block test

commit f910e9147ea3e3d8bfc8124c11f1a73a19f72ecb
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Wed Apr 14 16:41:39 2021 -0400
```

```
    fixed variable passing for funlit

commit 9e6cfced1383c6f12e2f05fe52abe2ac65a3eb33
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Wed Apr 14 16:33:01 2021 -0400

    updated some tests

commit e62e54770c4d1f4b6897ede68bb6c6339b424048
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Wed Apr 14 16:32:48 2021 -0400

    semant compiling and working

commit 4ac3d91a26087ec8fc0db9f9990ab66c731e6693
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Wed Apr 14 16:32:19 2021 -0400

    fixed constructors for if/else/match blocks

commit 78e46cc8697258206551dfa52fa25e3f97639c3e
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Wed Apr 14 16:31:43 2021 -0400

    removed xtra string_of_typ

commit 1e6553323f803b26f2b7c02c1e08fa11c4d4e7e0
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Wed Apr 14 14:20:10 2021 -0400

    added typ printing to sast

commit 317e35c8f3078ecd2ff669f91f39affb22b48dbb
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Tue Apr 13 22:48:56 2021 -0400

    None now of typ (remove typ_or_none), multiple types to cast)

commit e017b73a5f57e04b472b59e39692746d5931fc73
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Fri Apr 9 12:23:32 2021 -0400

    fixed type match error, updated usertype test

commit aa5b9417a7e288422fccb02b999c1ced47fe95e4
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Fri Apr 9 01:45:55 2021 -0400

    started semant, not functional yet

commit f72c303d0feb7a14898741248d7e0274efbc2e53
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Fri Apr 9 01:45:32 2021 -0400

    small structural changes to ast

commit 75d452fcd0a0c07b6b5f1f101c60a024133ce7b8
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Thu Apr 8 19:40:25 2021 -0400

    sast created

commit 1aaa596335cb09f2393300631edc92207f4f098a
```

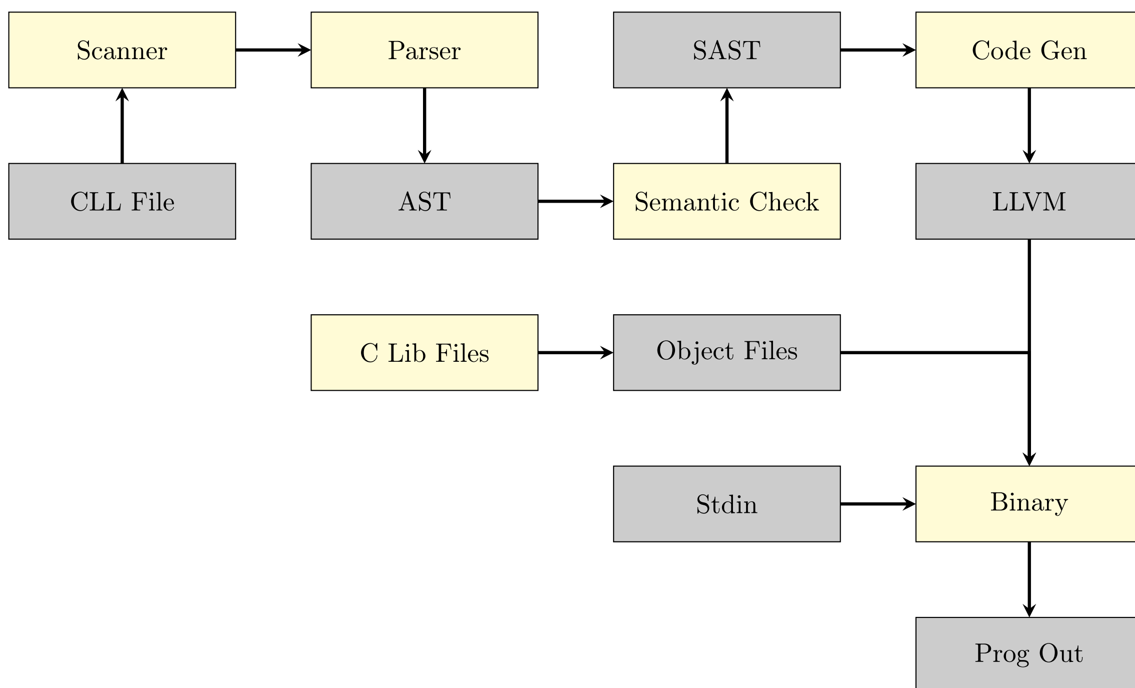
```
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Thu Apr 8 17:32:28 2021 -0400

    renaming some productions, fix newline in ast print for funlit and while

commit 54793469f84b04ca32838311b115a62f77d6fe14
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Thu Apr 8 13:34:56 2021 -0400

    changed id format, added fun literals, changed list/dict literals, output type
        different than decl type

commit b89a5b8fcc97d8673899c5f46c557d240e337e72
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Wed Apr 7 23:00:32 2021 -0400

    uppercase for regular variables now

commit 2e8a61265bc616fef2eb7da03554439c1368138c
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Wed Apr 7 22:55:46 2021 -0400

    type and typedef conflicts resolved

commit 944f0a8818ac41561d16d80627f61a974955b45d
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Wed Apr 7 20:11:42 2021 -0400

    trying to make typedef init work... stil shift/reduce errors

commit 1f9af0cac3d9cac4c2d06ecaa71a798f20a02d17
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Wed Apr 7 18:59:16 2021 -0400

    added concat to parser

commit 230c7e363311c994410c95e19aff34dc393e5d37
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Wed Apr 7 18:57:07 2021 -0400

    strlit and relit only contain non quote characters

commit 6c0892abd295b9f3006f9527a621609b7b6c9dd4
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Wed Apr 7 18:51:46 2021 -0400

    fixed quote removal for strings in pretty printing

commit ed647c1e171b514426e5ac36ce12a3d4e1b5119d
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Wed Apr 7 18:44:46 2021 -0400

    changeed newlines in pretty printing, removed quotes from strlit and relit

commit 36d6f159843817d0d986497047097c8267db0512
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Wed Apr 7 17:58:49 2021 -0400

    type specification for lists and dicts to ast/parser

commit d572ff7521245d42c1a97b260226c462c7aeac1c
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Wed Apr 7 17:20:10 2021 -0400
```

```
    fixed shift/reduce errors

commit d72fab72e60970b87f3c204236225b7207829586
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Wed Apr 7 17:19:54 2021 -0400

    starter Makefile

commit e1edc4ea8f0e538ce49edae600b9fa7aef97730f
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Wed Apr 7 16:06:52 2021 -0400

    Compiling to test pretty print, but Parse_error on running

commit 24dae28c2f5a1d6d76cacd4a7d1b09e7aa19e6e6
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Wed Apr 7 14:17:43 2021 -0400

    output types for blocks + code tuning

commit f0bdffe46aafc2e6818ac4b940fb8ce47ac8702f
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Mon Apr 5 23:28:02 2021 -0400

    parser almost done

commit 05cd84f901c11734574df2c8908ea6c6ea5affce
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Mon Apr 5 23:19:37 2021 -0400

    ast and scanner mostly done

commit 3b3f7dc1dabb87eac68e6c1eddf852f04591cdd5
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Mon Apr 5 12:40:25 2021 -0400

    gitignore

commit a04bcf1de9171a092c011e6a7a48c30dbade3b30
Author: Annalise Mariottini <annalise.mariottini@columbia.edu>
Date:   Tue Mar 30 00:30:31 2021 -0400

    ast, parser, scanner files created

commit 41190cda095bd111c71153817389839e80628e43
Author: Annalise Mariottini <annalisemariottini@Annalises-MacBook-Air.local>
Date:   Tue Mar 30 00:24:06 2021 -0400

    First commit
```

Chapter 5

Architectural Design

5.1 Compiler Pipeline Diagram



5.2 Component Descriptions

5.2.1 Scanner

The CLL scanner identifies tokens according to CLL's syntax. These tokens are symbols, keywords, and user identifiers. Comments and and whitespace are ignored. The scanner also keeps track of program line number for error reporting purposes in the event of an illegal character or parsing error later.

5.2.2 Parser

The CLL parser identifies the grammatical structure of a CLL program, creating an abstract syntax tree (AST) of program elements. The AST structure is defined by the context-free grammar that the parser creates through production rules. The tokens passed through the parser are identified

as elements of the right-hand-side of these production rules. Upon each new token read by the parser, the series of tokens encountered thus far is either shifted (a new token is added to the stack of tokens) or reduced according to a production rule of the parser. If a complete series of tokens can be reduced to the "start" production (i.e. a complete CLL program), the AST is accepted as syntactically valid. If at any point the parser has no action to take upon a new token read, it reports a parsing error and exits.

5.2.3 Semantic Check

The semantic checker takes in a syntactically correct AST and checks for any structures that violate the rules of the language. Since this is a strongly-typed language, the semantic checker ensures that all types are thoroughly checked according to the rules of the language. This includes, but is not limited to, variable reassignment, data object type, function arguments/return type, and expression output type. The semantic checker also ensure that variables are only used in their proper scope, user-defined types/typedefs are not reassigned, standard library functions are not reassigned, and user-defined and standard library functions are used only with arguments that exactly match their function definition.

If at any point the semantic checker encounters an invalid element, it reports a failure with a descriptive message and exits. This descriptive message includes a psuedo-"line number" of the error, which is generated from the line number on which the error occurs in the string-generated version of the AST (a string version of the AST is automatically written to filename.cll.ast in the event of a semantic error).

If the semantic checker traverses the entire AST without error, a new semantically-checked abstract syntax tree (SAST) is passed on in the compiler pipeline.

5.2.4 Code Generation

The most substantial portion of the compiler, code generation traverses a SAST and generates LLVM code for each element encountered. This includes loading data from memory, creating/reading from/writing to complex data objects, generating user-defined and standard library functions, calling functions, performing operations, generating control flow, and creating user-defined structures. At this point in the compiler, any errors encountered are internal errors of the compiler itself (please report any errors encountered to the maintainer of this compiler). If code generation runs successfully (as it should), an LLVM file is generated.

5.2.5 C Library Files

To assist in more complex tasks, C library files contain functions that handle complex data type functionality, regex functionality, stdin file input, and garbage collection. These files are linked with the generated LLVM file to create an executable binary file.

5.2.6 Binary Executable

Running the binary executable, when successful, will produce any text output printed by the initial CLL program. If a user decides to utilize stdin in their program, at this point they may pipe input into the binary file. The binary executable may fail if the user has utilized a data object in a way that the compiler cannot recognize as invalid at compile time (e.g. removing from an empty list or trying to access a key that is not in a dictionary). These errors will be reported by the OS at runtime.

Chapter 6

Test Plan

6.1 Example Tests and LLVM Output

6.1.1 A Small Test

```
f = <list<string> l:string>{
   lget(l,0);
};
l = <string>['a','b','c'];
sprint(f(l));
```

Output

```
; ModuleID = 'ConLangLang'
source_filename = "ConLangLang"

%"listi8*" = type { i8**, %ll_node* }
%ll_node = type { i8*, %ll_node* }
%hashtable_s = type { i32, i32, %entry_s**, i1 }
%entry_s = type { i8*, i8*, %entry_s* }
%regex_t = type { i32, i64, i8*, %re_guts* }
%re_guts = type opaque

@fmt = private unnamed_addr constant [4 x i8] c"%s\0A\00", align 1
@fmt.1 = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@fmt.2 = private unnamed_addr constant [3 x i8] c"%f\00", align 1

define i32 @main() {
entry:
  call void @init_malloc_addr()
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1**
      getelementptr (i1*, i1** null, i32 1) to i64), i64 2) to i32))
  %stdin = bitcast i8* %malloccall to %"listi8*"*
  %mallocnull = icmp eq %"listi8*"* %stdin, null
  br i1 %mallocnull, label %endprog, label %contprog

endprog:                                          ; preds = %entry
  call void @exit(i32 1)
  unreachable

contprog:                                         ; preds = %entry
  %caddr = bitcast %"listi8*"* %stdin to i8*
  call void @add_malloc_addr(i8* %caddr)
```

```
  %listltyp = getelementptr inbounds %"listi8*", %"listi8*"* %stdin, i32 0, i32 0
  %listhead = getelementptr inbounds %"listi8*", %"listi8*"* %stdin, i32 0, i32 1
  %malloccall3 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1** null,
      i32 1) to i32))
  %null = bitcast i8* %malloccall3 to i8**
  %mallocnull4 = icmp eq i8** %null, null
  br i1 %mallocnull4, label %endprog1, label %contprog2

endprog1:                                         ; preds = %contprog
  call void @exit(i32 1)
  unreachable

contprog2:                                        ; preds = %contprog
  %caddr5 = bitcast i8** %null to i8*
  call void @add_malloc_addr(i8* %caddr5)
  store i8* null, i8** %null, align 8
  store i8** %null, i8*** %listltyp, align 8
  store %ll_node* null, %ll_node** %listhead, align 8
  %stdinlisthead = getelementptr inbounds %"listi8*", %"listi8*"* %stdin, i32 0, i32 1
  %stdinheadnode = call %ll_node* @ll_of_stdin()
  store %ll_node* %stdinheadnode, %ll_node** %stdinlisthead, align 8
  %malloccall8 = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1**
      getelementptr (i1*, i1** null, i32 1) to i64), i64 2) to i32))
  %list = bitcast i8* %malloccall8 to %"listi8*"*
  %mallocnull9 = icmp eq %"listi8*"* %list, null
  br i1 %mallocnull9, label %endprog6, label %contprog7

endprog6:                                         ; preds = %contprog2
  call void @exit(i32 1)
  unreachable

contprog7:                                        ; preds = %contprog2
  %caddr10 = bitcast %"listi8*"* %list to i8*
  call void @add_malloc_addr(i8* %caddr10)
  %listltyp11 = getelementptr inbounds %"listi8*", %"listi8*"* %list, i32 0, i32 0
  %listhead12 = getelementptr inbounds %"listi8*", %"listi8*"* %list, i32 0, i32 1
  %malloccall15 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1** null,
      i32 1) to i32))
  %null16 = bitcast i8* %malloccall15 to i8**
  %mallocnull17 = icmp eq i8** %null16, null
  br i1 %mallocnull17, label %endprog13, label %contprog14

endprog13:                                        ; preds = %contprog7
  call void @exit(i32 1)
  unreachable

contprog14:                                       ; preds = %contprog7
  %caddr18 = bitcast i8** %null16 to i8*
  call void @add_malloc_addr(i8* %caddr18)
  store i8* null, i8** %null16, align 8
  store i8** %null16, i8*** %listltyp11, align 8
  %string = tail call i8* @malloc(i32 mul (i32 ptrtoint (i8* getelementptr (i8, i8* null,
      i32 1) to i32), i32 2))
  %mallocnull22 = icmp eq i8* %string, null
  br i1 %mallocnull22, label %endprog19, label %contprog20

endprog19:                                        ; preds = %contprog14
  call void @exit(i32 1)
  unreachable

contprog20:                                       ; preds = %contprog14
  call void @add_malloc_addr(i8* %string)
  %s0 = getelementptr inbounds i8, i8* %string, i32 0
```

```
  store i8 99, i8* %s0, align 1
  %s1 = getelementptr inbounds i8, i8* %string, i32 1
  store i8 0, i8* %s1, align 1
  %listlitadd = call %"listi8*"* @"ladd%listi8**"(%"listi8*"* %list, i8* %string, i32 0)
  %string26 = tail call i8* @malloc(i32 mul (i32 ptrtoint (i8* getelementptr (i8, i8* null
      , i32 1) to i32), i32 2))
  %mallocnull27 = icmp eq i8* %string26, null
  br i1 %mallocnull27, label %endprog23, label %contprog24

endprog23:                                        ; preds = %contprog20
  call void @exit(i32 1)
  unreachable

contprog24:                                       ; preds = %contprog20
  call void @add_malloc_addr(i8* %string26)
  %s028 = getelementptr inbounds i8, i8* %string26, i32 0
  store i8 98, i8* %s028, align 1
  %s129 = getelementptr inbounds i8, i8* %string26, i32 1
  store i8 0, i8* %s129, align 1
  %listlitadd30 = call %"listi8*"* @"ladd%listi8**"(%"listi8*"* %listlitadd, i8* %string26
      , i32 0)
  %string34 = tail call i8* @malloc(i32 mul (i32 ptrtoint (i8* getelementptr (i8, i8* null
      , i32 1) to i32), i32 2))
  %mallocnull35 = icmp eq i8* %string34, null
  br i1 %mallocnull35, label %endprog31, label %contprog32

endprog31:                                        ; preds = %contprog24
  call void @exit(i32 1)
  unreachable

contprog32:                                       ; preds = %contprog24
  call void @add_malloc_addr(i8* %string34)
  %s036 = getelementptr inbounds i8, i8* %string34, i32 0
  store i8 97, i8* %s036, align 1
  %s137 = getelementptr inbounds i8, i8* %string34, i32 1
  store i8 0, i8* %s137, align 1
  %l = call %"listi8*"* @"ladd%listi8**"(%"listi8*"* %listlitadd30, i8* %string34, i32 0)
  %funcall = call i8* @f(%"listi8*"* %l, %"listi8*"* %stdin)
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]*
      @fmt, i32 0, i32 0), i8* %funcall)
  call void @free_malloc_addrs()
  ret i32 0
}

declare void @exit(i32)

declare %ll_node* @ll_of_stdin()

declare void @init_malloc_addr()

declare void @add_malloc_addr(i8*)

declare void @free_malloc_addrs()

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

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

declare i8* @strcpy(i8*, i8*)

declare i8* @strcat(i8*, i8*)

declare i32 @strlen(i8*)
```

```
declare %ll_node* @ll_create(i8*)

declare %ll_node* @ll_add(%ll_node*, i8*, i32)

declare %ll_node* @ll_append(%ll_node*, %ll_node*)

declare %ll_node* @ll_next(%ll_node*)

declare i32 @ll_mem(%ll_node*, i8*, i1)

declare i8* @ll_get(%ll_node*, i32)

declare %ll_node* @ll_remove(%ll_node*, i32)

declare i32 @ll_print(%ll_node*)

declare i32 @ll_size(%ll_node*)

declare %hashtable_s* @ht_create(i32, i1)

declare i1 @ht_mem(%hashtable_s*, i8*)

declare i8* @ht_get(%hashtable_s*, i8*)

declare %hashtable_s* @ht_remove(%hashtable_s*, i8*)

declare %hashtable_s* @ht_add(%hashtable_s*, i8*, i8*)

declare i32 @ht_print(%hashtable_s*)

declare i32 @ht_size(%hashtable_s*)

declare i8** @ht_keys(%hashtable_s*)

declare %ll_node* @ht_keys_list(%hashtable_s*)

declare %regex_t* @re_create(i8*)

declare i1 @re_match(%regex_t*, i8*)

declare i8* @re_sub(%regex_t*, i8*, i8*, i32)

declare noalias i8* @malloc(i32)

define i8* @f(%"listi8*"* %l, %"listi8*"* %stdin) {
entry:
  %0 = alloca %"listi8*"*, align 8
  store %"listi8*"* %l, %"listi8*"** %0, align 8
  %1 = alloca %"listi8*"*, align 8
  store %"listi8*"* %stdin, %"listi8*"** %1, align 8
  %param0 = load %"listi8*"*, %"listi8*"** %0, align 8
  %param1 = load %"listi8*"*, %"listi8*"** %1, align 8
  %listt = getelementptr inbounds %"listi8*", %"listi8*"* %param0, i32 0, i32 0
  %lget = call i8* @"lget%listi8**"(%"listi8*"* %param0, i32 0)
  ret i8* %lget
}

define i8* @"lget%listi8**"(%"listi8*"* %"#l", i32 %"#n") {
entry:
  %0 = alloca %"listi8*"*, align 8
  store %"listi8*"* %"#l", %"listi8*"** %0, align 8
  %1 = alloca i32, align 4
```

```
  store i32 %"#n", i32* %1, align 4
  %param0 = load %"listi8*"*, %"listi8*"** %0, align 8
  %param1 = load i32, i32* %1, align 4
  %listt = getelementptr inbounds %"listi8*", %"listi8*"* %param0, i32 0, i32 0
  %listhead = getelementptr inbounds %"listi8*", %"listi8*"* %param0, i32 0, i32 1
  %"t*" = load i8**, i8*** %listt, align 8
  %headnode = load %ll_node*, %ll_node** %listhead, align 8
  %cdata = call i8* @ll_get(%ll_node* %headnode, i32 %param1)
  %data = bitcast i8* %cdata to i8**
  %dataload = load i8*, i8** %data, align 8
  ret i8* %dataload
}

define %"listi8*"* @"ladd%listi8**"(%"listi8*"* %"#l", i8* %"#e", i32 %"#n") {
entry:
  %0 = alloca %"listi8*"*, align 8
  store %"listi8*"* %"#l", %"listi8*"** %0, align 8
  %1 = alloca i8*, align 8
  store i8* %"#e", i8** %1, align 8
  %2 = alloca i32, align 4
  store i32 %"#n", i32* %2, align 4
  %param0 = load %"listi8*"*, %"listi8*"** %0, align 8
  %param1 = load i8*, i8** %1, align 8
  %param2 = load i32, i32* %2, align 4
  %listhead = getelementptr inbounds %"listi8*", %"listi8*"* %param0, i32 0, i32 1
  %headnode = load %ll_node*, %ll_node** %listhead, align 8
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1** null,
      i32 1) to i32))
  %3 = bitcast i8* %malloccall to i8**
  %mallocnull = icmp eq i8** %3, null
  br i1 %mallocnull, label %endprog, label %contprog

endprog:                                          ; preds = %entry
  call void @exit(i32 1)
  unreachable

contprog:                                         ; preds = %entry
  %caddr = bitcast i8** %3 to i8*
  call void @add_malloc_addr(i8* %caddr)
  store i8* %param1, i8** %3, align 8
  %cdata = bitcast i8** %3 to i8*
  %4 = call %ll_node* @ll_add(%ll_node* %headnode, i8* %cdata, i32 %param2)
  store %ll_node* %4, %ll_node** %listhead, align 8
  ret %"listi8*"* %param0
}
```

6.1.2 A Larger Test

```
r = "(abc)+";
f = <regex r, string s:int> {
    if:int (rematch(r,s)) {
        sprint('match');
    } else {
        sprint('no match');
    };
};

f(r,'abc');
f(r,'xxxxabcabcxxxxx');
f(r,'aaa');

r2 = "r(in)+(on)*g";
s = 'rininining';
s2 = resub(r2,s,'on',1);
sprint(s);
sprint(s2);

r3 = "r(o*(lo(ng)*))+(o*(lin))*";
s3 = 'rololongolong';
s4 = resub(r3,s3,'lin',2);
sprint(s3);
sprint(s4);
```

Output

```
; ModuleID = 'ConLangLang'
source_filename = "ConLangLang"

%"listi8*" = type { i8**, %ll_node* }
%ll_node = type { i8*, %ll_node* }
%regex_t = type { i32, i64, i8*, %re_guts* }
%re_guts = type opaque
%hashtable_s = type { i32, i32, %entry_s**, i1 }
%entry_s = type { i8*, i8*, %entry_s* }

@fmt = private unnamed_addr constant [4 x i8] c"%s\0A\00", align 1
@fmt.1 = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@fmt.2 = private unnamed_addr constant [3 x i8] c"%f\00", align 1

define i32 @main() {
entry:
  call void @init_malloc_addr()
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1**
      getelementptr (i1*, i1** null, i32 1) to i64), i64 2) to i32))
  %stdin = bitcast i8* %malloccall to %"listi8*"*
  %mallocnull = icmp eq %"listi8*"* %stdin, null
  br i1 %mallocnull, label %endprog, label %contprog

endprog:                                          ; preds = %entry
  call void @exit(i32 1)
  unreachable

contprog:                                         ; preds = %entry
  %caddr = bitcast %"listi8*"* %stdin to i8*
  call void @add_malloc_addr(i8* %caddr)
  %listltyp = getelementptr inbounds %"listi8*", %"listi8*"* %stdin, i32 0, i32 0
  %listhead = getelementptr inbounds %"listi8*", %"listi8*"* %stdin, i32 0, i32 1
  %malloccall3 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1** null,
      i32 1) to i32))
```

```
  %null = bitcast i8* %malloccall3 to i8**
  %mallocnull4 = icmp eq i8** %null, null
  br i1 %mallocnull4, label %endprog1, label %contprog2

endprog1:                                         ; preds = %contprog
  call void @exit(i32 1)
  unreachable

contprog2:                                        ; preds = %contprog
  %caddr5 = bitcast i8** %null to i8*
  call void @add_malloc_addr(i8* %caddr5)
  store i8* null, i8** %null, align 8
  store i8** %null, i8*** %listltyp, align 8
  store %ll_node* null, %ll_node** %listhead, align 8
  %stdinlisthead = getelementptr inbounds %"listi8*", %"listi8*"* %stdin, i32 0, i32 1
  %stdinheadnode = call %ll_node* @ll_of_stdin()
  store %ll_node* %stdinheadnode, %ll_node** %stdinlisthead, align 8
  %string = tail call i8* @malloc(i32 mul (i32 ptrtoint (i8* getelementptr (i8, i8* null,
      i32 1) to i32), i32 7))
  %mallocnull9 = icmp eq i8* %string, null
  br i1 %mallocnull9, label %endprog6, label %contprog7

endprog6:                                         ; preds = %contprog2
  call void @exit(i32 1)
  unreachable

contprog7:                                        ; preds = %contprog2
  call void @add_malloc_addr(i8* %string)
  %s0 = getelementptr inbounds i8, i8* %string, i32 0
  store i8 40, i8* %s0, align 1
  %s1 = getelementptr inbounds i8, i8* %string, i32 1
  store i8 97, i8* %s1, align 1
  %s2 = getelementptr inbounds i8, i8* %string, i32 2
  store i8 98, i8* %s2, align 1
  %s3 = getelementptr inbounds i8, i8* %string, i32 3
  store i8 99, i8* %s3, align 1
  %s4 = getelementptr inbounds i8, i8* %string, i32 4
  store i8 41, i8* %s4, align 1
  %s5 = getelementptr inbounds i8, i8* %string, i32 5
  store i8 43, i8* %s5, align 1
  %s6 = getelementptr inbounds i8, i8* %string, i32 6
  store i8 0, i8* %s6, align 1
  %r = call %regex_t* @re_create(i8* %string)
  %string13 = tail call i8* @malloc(i32 mul (i32 ptrtoint (i8* getelementptr (i8, i8* null
      , i32 1) to i32), i32 4))
  %mallocnull14 = icmp eq i8* %string13, null
  br i1 %mallocnull14, label %endprog10, label %contprog11

endprog10:                                        ; preds = %contprog7
  call void @exit(i32 1)
  unreachable

contprog11:                                       ; preds = %contprog7
  call void @add_malloc_addr(i8* %string13)
  %s015 = getelementptr inbounds i8, i8* %string13, i32 0
  store i8 97, i8* %s015, align 1
  %s116 = getelementptr inbounds i8, i8* %string13, i32 1
  store i8 98, i8* %s116, align 1
  %s217 = getelementptr inbounds i8, i8* %string13, i32 2
  store i8 99, i8* %s217, align 1
  %s318 = getelementptr inbounds i8, i8* %string13, i32 3
  store i8 0, i8* %s318, align 1
  %funcall = call i32 @f(%regex_t* %r, i8* %string13, %"listi8*"* %stdin)
```

```
  %string22 = tail call i8* @malloc(i32 mul (i32 ptrtoint (i8* getelementptr (i8, i8* null
      , i32 1) to i32), i32 16))
  %mallocnull23 = icmp eq i8* %string22, null
  br i1 %mallocnull23, label %endprog19, label %contprog20

endprog19:                                        ; preds = %contprog11
  call void @exit(i32 1)
  unreachable

contprog20:                                       ; preds = %contprog11
  call void @add_malloc_addr(i8* %string22)
  %s024 = getelementptr inbounds i8, i8* %string22, i32 0
  store i8 120, i8* %s024, align 1
  %s125 = getelementptr inbounds i8, i8* %string22, i32 1
  store i8 120, i8* %s125, align 1
  %s226 = getelementptr inbounds i8, i8* %string22, i32 2
  store i8 120, i8* %s226, align 1
  %s327 = getelementptr inbounds i8, i8* %string22, i32 3
  store i8 120, i8* %s327, align 1
  %s428 = getelementptr inbounds i8, i8* %string22, i32 4
  store i8 97, i8* %s428, align 1
  %s529 = getelementptr inbounds i8, i8* %string22, i32 5
  store i8 98, i8* %s529, align 1
  %s630 = getelementptr inbounds i8, i8* %string22, i32 6
  store i8 99, i8* %s630, align 1
  %s7 = getelementptr inbounds i8, i8* %string22, i32 7
  store i8 97, i8* %s7, align 1
  %s8 = getelementptr inbounds i8, i8* %string22, i32 8
  store i8 98, i8* %s8, align 1
  %s9 = getelementptr inbounds i8, i8* %string22, i32 9
  store i8 99, i8* %s9, align 1
  %s10 = getelementptr inbounds i8, i8* %string22, i32 10
  store i8 120, i8* %s10, align 1
  %s11 = getelementptr inbounds i8, i8* %string22, i32 11
  store i8 120, i8* %s11, align 1
  %s12 = getelementptr inbounds i8, i8* %string22, i32 12
  store i8 120, i8* %s12, align 1
  %s13 = getelementptr inbounds i8, i8* %string22, i32 13
  store i8 120, i8* %s13, align 1
  %s14 = getelementptr inbounds i8, i8* %string22, i32 14
  store i8 120, i8* %s14, align 1
  %s15 = getelementptr inbounds i8, i8* %string22, i32 15
  store i8 0, i8* %s15, align 1
  %funcall31 = call i32 @f(%regex_t* %r, i8* %string22, %"listi8*"* %stdin)
  %string35 = tail call i8* @malloc(i32 mul (i32 ptrtoint (i8* getelementptr (i8, i8* null
      , i32 1) to i32), i32 4))
  %mallocnull36 = icmp eq i8* %string35, null
  br i1 %mallocnull36, label %endprog32, label %contprog33

endprog32:                                        ; preds = %contprog20
  call void @exit(i32 1)
  unreachable

contprog33:                                       ; preds = %contprog20
  call void @add_malloc_addr(i8* %string35)
  %s037 = getelementptr inbounds i8, i8* %string35, i32 0
  store i8 97, i8* %s037, align 1
  %s138 = getelementptr inbounds i8, i8* %string35, i32 1
  store i8 97, i8* %s138, align 1
  %s239 = getelementptr inbounds i8, i8* %string35, i32 2
  store i8 97, i8* %s239, align 1
  %s340 = getelementptr inbounds i8, i8* %string35, i32 3
  store i8 0, i8* %s340, align 1
```

```
  %funcall41 = call i32 @f(%regex_t* %r, i8* %string35, %"listi8*"* %stdin)
  %string45 = tail call i8* @malloc(i32 mul (i32 ptrtoint (i8* getelementptr (i8, i8* null
      , i32 1) to i32), i32 13))
  %mallocnull46 = icmp eq i8* %string45, null
  br i1 %mallocnull46, label %endprog42, label %contprog43

endprog42:                                        ; preds = %contprog33
  call void @exit(i32 1)
  unreachable

contprog43:                                       ; preds = %contprog33
  call void @add_malloc_addr(i8* %string45)
  %s047 = getelementptr inbounds i8, i8* %string45, i32 0
  store i8 114, i8* %s047, align 1
  %s148 = getelementptr inbounds i8, i8* %string45, i32 1
  store i8 40, i8* %s148, align 1
  %s249 = getelementptr inbounds i8, i8* %string45, i32 2
  store i8 105, i8* %s249, align 1
  %s350 = getelementptr inbounds i8, i8* %string45, i32 3
  store i8 110, i8* %s350, align 1
  %s451 = getelementptr inbounds i8, i8* %string45, i32 4
  store i8 41, i8* %s451, align 1
  %s552 = getelementptr inbounds i8, i8* %string45, i32 5
  store i8 43, i8* %s552, align 1
  %s653 = getelementptr inbounds i8, i8* %string45, i32 6
  store i8 40, i8* %s653, align 1
  %s754 = getelementptr inbounds i8, i8* %string45, i32 7
  store i8 111, i8* %s754, align 1
  %s855 = getelementptr inbounds i8, i8* %string45, i32 8
  store i8 110, i8* %s855, align 1
  %s956 = getelementptr inbounds i8, i8* %string45, i32 9
  store i8 41, i8* %s956, align 1
  %s1057 = getelementptr inbounds i8, i8* %string45, i32 10
  store i8 42, i8* %s1057, align 1
  %s1158 = getelementptr inbounds i8, i8* %string45, i32 11
  store i8 103, i8* %s1158, align 1
  %s1259 = getelementptr inbounds i8, i8* %string45, i32 12
  store i8 0, i8* %s1259, align 1
  %r2 = call %regex_t* @re_create(i8* %string45)
  %s = tail call i8* @malloc(i32 mul (i32 ptrtoint (i8* getelementptr (i8, i8* null, i32
      1) to i32), i32 11))
  %mallocnull64 = icmp eq i8* %s, null
  br i1 %mallocnull64, label %endprog60, label %contprog61

endprog60:                                        ; preds = %contprog43
  call void @exit(i32 1)
  unreachable

contprog61:                                       ; preds = %contprog43
  call void @add_malloc_addr(i8* %s)
  %s065 = getelementptr inbounds i8, i8* %s, i32 0
  store i8 114, i8* %s065, align 1
  %s166 = getelementptr inbounds i8, i8* %s, i32 1
  store i8 105, i8* %s166, align 1
  %s267 = getelementptr inbounds i8, i8* %s, i32 2
  store i8 110, i8* %s267, align 1
  %s368 = getelementptr inbounds i8, i8* %s, i32 3
  store i8 105, i8* %s368, align 1
  %s469 = getelementptr inbounds i8, i8* %s, i32 4
  store i8 110, i8* %s469, align 1
  %s570 = getelementptr inbounds i8, i8* %s, i32 5
  store i8 105, i8* %s570, align 1
  %s671 = getelementptr inbounds i8, i8* %s, i32 6
```

```
  store i8 110, i8* %s671, align 1
  %s772 = getelementptr inbounds i8, i8* %s, i32 7
  store i8 105, i8* %s772, align 1
  %s873 = getelementptr inbounds i8, i8* %s, i32 8
  store i8 110, i8* %s873, align 1
  %s974 = getelementptr inbounds i8, i8* %s, i32 9
  store i8 103, i8* %s974, align 1
  %s1075 = getelementptr inbounds i8, i8* %s, i32 10
  store i8 0, i8* %s1075, align 1
  %string79 = tail call i8* @malloc(i32 mul (i32 ptrtoint (i8* getelementptr (i8, i8* null
      , i32 1) to i32), i32 3))
  %mallocnull80 = icmp eq i8* %string79, null
  br i1 %mallocnull80, label %endprog76, label %contprog77

endprog76:                                        ; preds = %contprog61
  call void @exit(i32 1)
  unreachable

contprog77:                                       ; preds = %contprog61
  call void @add_malloc_addr(i8* %string79)
  %s081 = getelementptr inbounds i8, i8* %string79, i32 0
  store i8 111, i8* %s081, align 1
  %s182 = getelementptr inbounds i8, i8* %string79, i32 1
  store i8 110, i8* %s182, align 1
  %s283 = getelementptr inbounds i8, i8* %string79, i32 2
  store i8 0, i8* %s283, align 1
  %s284 = call i8* @re_sub(%regex_t* %r2, i8* %s, i8* %string79, i32 1)
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]*
      @fmt, i32 0, i32 0), i8* %s)
  %printf85 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]*
      @fmt, i32 0, i32 0), i8* %s284)
  %string89 = tail call i8* @malloc(i32 mul (i32 ptrtoint (i8* getelementptr (i8, i8* null
      , i32 1) to i32), i32 26))
  %mallocnull90 = icmp eq i8* %string89, null
  br i1 %mallocnull90, label %endprog86, label %contprog87

endprog86:                                        ; preds = %contprog77
  call void @exit(i32 1)
  unreachable

contprog87:                                       ; preds = %contprog77
  call void @add_malloc_addr(i8* %string89)
  %s091 = getelementptr inbounds i8, i8* %string89, i32 0
  store i8 114, i8* %s091, align 1
  %s192 = getelementptr inbounds i8, i8* %string89, i32 1
  store i8 40, i8* %s192, align 1
  %s293 = getelementptr inbounds i8, i8* %string89, i32 2
  store i8 111, i8* %s293, align 1
  %s394 = getelementptr inbounds i8, i8* %string89, i32 3
  store i8 42, i8* %s394, align 1
  %s495 = getelementptr inbounds i8, i8* %string89, i32 4
  store i8 40, i8* %s495, align 1
  %s596 = getelementptr inbounds i8, i8* %string89, i32 5
  store i8 108, i8* %s596, align 1
  %s697 = getelementptr inbounds i8, i8* %string89, i32 6
  store i8 111, i8* %s697, align 1
  %s798 = getelementptr inbounds i8, i8* %string89, i32 7
  store i8 40, i8* %s798, align 1
  %s899 = getelementptr inbounds i8, i8* %string89, i32 8
  store i8 110, i8* %s899, align 1
  %s9100 = getelementptr inbounds i8, i8* %string89, i32 9
  store i8 103, i8* %s9100, align 1
  %s10101 = getelementptr inbounds i8, i8* %string89, i32 10
```

```
  store i8 41, i8* %s10101, align 1
  %s11102 = getelementptr inbounds i8, i8* %string89, i32 11
  store i8 42, i8* %s11102, align 1
  %s12103 = getelementptr inbounds i8, i8* %string89, i32 12
  store i8 41, i8* %s12103, align 1
  %s13104 = getelementptr inbounds i8, i8* %string89, i32 13
  store i8 41, i8* %s13104, align 1
  %s14105 = getelementptr inbounds i8, i8* %string89, i32 14
  store i8 43, i8* %s14105, align 1
  %s15106 = getelementptr inbounds i8, i8* %string89, i32 15
  store i8 40, i8* %s15106, align 1
  %s16 = getelementptr inbounds i8, i8* %string89, i32 16
  store i8 111, i8* %s16, align 1
  %s17 = getelementptr inbounds i8, i8* %string89, i32 17
  store i8 42, i8* %s17, align 1
  %s18 = getelementptr inbounds i8, i8* %string89, i32 18
  store i8 40, i8* %s18, align 1
  %s19 = getelementptr inbounds i8, i8* %string89, i32 19
  store i8 108, i8* %s19, align 1
  %s20 = getelementptr inbounds i8, i8* %string89, i32 20
  store i8 105, i8* %s20, align 1
  %s21 = getelementptr inbounds i8, i8* %string89, i32 21
  store i8 110, i8* %s21, align 1
  %s22 = getelementptr inbounds i8, i8* %string89, i32 22
  store i8 41, i8* %s22, align 1
  %s23 = getelementptr inbounds i8, i8* %string89, i32 23
  store i8 41, i8* %s23, align 1
  %s24 = getelementptr inbounds i8, i8* %string89, i32 24
  store i8 42, i8* %s24, align 1
  %s25 = getelementptr inbounds i8, i8* %string89, i32 25
  store i8 0, i8* %s25, align 1
  %r3 = call %regex_t* @re_create(i8* %string89)
  %s3126 = tail call i8* @malloc(i32 mul (i32 ptrtoint (i8* getelementptr (i8, i8* null,
      i32 1) to i32), i32 14))
  %mallocnull111 = icmp eq i8* %s3126, null
  br i1 %mallocnull111, label %endprog107, label %contprog108

endprog107:                                       ; preds = %contprog87
  call void @exit(i32 1)
  unreachable

contprog108:                                      ; preds = %contprog87
  call void @add_malloc_addr(i8* %s3126)
  %s0112 = getelementptr inbounds i8, i8* %s3126, i32 0
  store i8 114, i8* %s0112, align 1
  %s1113 = getelementptr inbounds i8, i8* %s3126, i32 1
  store i8 111, i8* %s1113, align 1
  %s2114 = getelementptr inbounds i8, i8* %s3126, i32 2
  store i8 108, i8* %s2114, align 1
  %s3115 = getelementptr inbounds i8, i8* %s3126, i32 3
  store i8 111, i8* %s3115, align 1
  %s4116 = getelementptr inbounds i8, i8* %s3126, i32 4
  store i8 108, i8* %s4116, align 1
  %s5117 = getelementptr inbounds i8, i8* %s3126, i32 5
  store i8 111, i8* %s5117, align 1
  %s6118 = getelementptr inbounds i8, i8* %s3126, i32 6
  store i8 110, i8* %s6118, align 1
  %s7119 = getelementptr inbounds i8, i8* %s3126, i32 7
  store i8 103, i8* %s7119, align 1
  %s8120 = getelementptr inbounds i8, i8* %s3126, i32 8
  store i8 111, i8* %s8120, align 1
  %s9121 = getelementptr inbounds i8, i8* %s3126, i32 9
  store i8 108, i8* %s9121, align 1
```

```
  %s10122 = getelementptr inbounds i8, i8* %s3126, i32 10
  store i8 111, i8* %s10122, align 1
  %s11123 = getelementptr inbounds i8, i8* %s3126, i32 11
  store i8 110, i8* %s11123, align 1
  %s12124 = getelementptr inbounds i8, i8* %s3126, i32 12
  store i8 103, i8* %s12124, align 1
  %s13125 = getelementptr inbounds i8, i8* %s3126, i32 13
  store i8 0, i8* %s13125, align 1
  %string130 = tail call i8* @malloc(i32 mul (i32 ptrtoint (i8* getelementptr (i8, i8*
      null, i32 1) to i32), i32 4))
  %mallocnull131 = icmp eq i8* %string130, null
  br i1 %mallocnull131, label %endprog127, label %contprog128

endprog127:                                       ; preds = %contprog108
  call void @exit(i32 1)
  unreachable

contprog128:                                      ; preds = %contprog108
  call void @add_malloc_addr(i8* %string130)
  %s0132 = getelementptr inbounds i8, i8* %string130, i32 0
  store i8 108, i8* %s0132, align 1
  %s1133 = getelementptr inbounds i8, i8* %string130, i32 1
  store i8 105, i8* %s1133, align 1
  %s2134 = getelementptr inbounds i8, i8* %string130, i32 2
  store i8 110, i8* %s2134, align 1
  %s3135 = getelementptr inbounds i8, i8* %string130, i32 3
  store i8 0, i8* %s3135, align 1
  %s4136 = call i8* @re_sub(%regex_t* %r3, i8* %s3126, i8* %string130, i32 2)
  %printf137 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]*
       @fmt, i32 0, i32 0), i8* %s3126)
  %printf138 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]*
       @fmt, i32 0, i32 0), i8* %s4136)
  call void @free_malloc_addrs()
  ret i32 0
}

declare void @exit(i32)

declare %ll_node* @ll_of_stdin()

declare void @init_malloc_addr()

declare void @add_malloc_addr(i8*)

declare void @free_malloc_addrs()

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

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

declare i8* @strcpy(i8*, i8*)

declare i8* @strcat(i8*, i8*)

declare i32 @strlen(i8*)

declare %ll_node* @ll_create(i8*)

declare %ll_node* @ll_add(%ll_node*, i8*, i32)

declare %ll_node* @ll_append(%ll_node*, %ll_node*)

declare %ll_node* @ll_next(%ll_node*)
```

```
declare i32 @ll_mem(%ll_node*, i8*, i1)

declare i8* @ll_get(%ll_node*, i32)

declare %ll_node* @ll_remove(%ll_node*, i32)

declare i32 @ll_print(%ll_node*)

declare i32 @ll_size(%ll_node*)

declare %hashtable_s* @ht_create(i32, i1)

declare i1 @ht_mem(%hashtable_s*, i8*)

declare i8* @ht_get(%hashtable_s*, i8*)

declare %hashtable_s* @ht_remove(%hashtable_s*, i8*)

declare %hashtable_s* @ht_add(%hashtable_s*, i8*, i8*)

declare i32 @ht_print(%hashtable_s*)

declare i32 @ht_size(%hashtable_s*)

declare i8** @ht_keys(%hashtable_s*)

declare %ll_node* @ht_keys_list(%hashtable_s*)

declare %regex_t* @re_create(i8*)

declare i1 @re_match(%regex_t*, i8*)

declare i8* @re_sub(%regex_t*, i8*, i8*, i32)

declare noalias i8* @malloc(i32)

define i32 @f(%regex_t* %r, i8* %s, %"listi8*"* %stdin) {
entry:
  %0 = alloca %regex_t*, align 8
  store %regex_t* %r, %regex_t** %0, align 8
  %1 = alloca i8*, align 8
  store i8* %s, i8** %1, align 8
  %2 = alloca %"listi8*"*, align 8
  store %"listi8*"* %stdin, %"listi8*"** %2, align 8
  %param0 = load %regex_t*, %regex_t** %0, align 8
  %param1 = load i8*, i8** %1, align 8
  %param2 = load %"listi8*"*, %"listi8*"** %2, align 8
  %rematch = call i1 @re_match(%regex_t* %param0, i8* %param1)
  %out = alloca i32, align 4
  br i1 %rematch, label %then, label %else

merge:                                            ; preds = %contprog2, %contprog
  %ifelseout = load i32, i32* %out, align 4
  ret i32 %ifelseout

then:                                             ; preds = %entry
  %string = tail call i8* @malloc(i32 mul (i32 ptrtoint (i8* getelementptr (i8, i8* null,
      i32 1) to i32), i32 6))
  %mallocnull = icmp eq i8* %string, null
  br i1 %mallocnull, label %endprog, label %contprog

endprog:                                          ; preds = %then
```

```
  call void @exit(i32 1)
  unreachable

contprog:                                         ; preds = %then
  call void @add_malloc_addr(i8* %string)
  %s0 = getelementptr inbounds i8, i8* %string, i32 0
  store i8 109, i8* %s0, align 1
  %s1 = getelementptr inbounds i8, i8* %string, i32 1
  store i8 97, i8* %s1, align 1
  %s2 = getelementptr inbounds i8, i8* %string, i32 2
  store i8 116, i8* %s2, align 1
  %s3 = getelementptr inbounds i8, i8* %string, i32 3
  store i8 99, i8* %s3, align 1
  %s4 = getelementptr inbounds i8, i8* %string, i32 4
  store i8 104, i8* %s4, align 1
  %s5 = getelementptr inbounds i8, i8* %string, i32 5
  store i8 0, i8* %s5, align 1
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]*
      @fmt, i32 0, i32 0), i8* %string)
  store i32 %printf, i32* %out, align 4
  br label %merge

else:                                             ; preds = %entry
  %string3 = tail call i8* @malloc(i32 mul (i32 ptrtoint (i8* getelementptr (i8, i8* null,
       i32 1) to i32), i32 9))
  %mallocnull4 = icmp eq i8* %string3, null
  br i1 %mallocnull4, label %endprog1, label %contprog2

endprog1:                                         ; preds = %else
  call void @exit(i32 1)
  unreachable

contprog2:                                        ; preds = %else
  call void @add_malloc_addr(i8* %string3)
  %s05 = getelementptr inbounds i8, i8* %string3, i32 0
  store i8 110, i8* %s05, align 1
  %s16 = getelementptr inbounds i8, i8* %string3, i32 1
  store i8 111, i8* %s16, align 1
  %s27 = getelementptr inbounds i8, i8* %string3, i32 2
  store i8 32, i8* %s27, align 1
  %s38 = getelementptr inbounds i8, i8* %string3, i32 3
  store i8 109, i8* %s38, align 1
  %s49 = getelementptr inbounds i8, i8* %string3, i32 4
  store i8 97, i8* %s49, align 1
  %s510 = getelementptr inbounds i8, i8* %string3, i32 5
  store i8 116, i8* %s510, align 1
  %s6 = getelementptr inbounds i8, i8* %string3, i32 6
  store i8 99, i8* %s6, align 1
  %s7 = getelementptr inbounds i8, i8* %string3, i32 7
  store i8 104, i8* %s7, align 1
  %s8 = getelementptr inbounds i8, i8* %string3, i32 8
  store i8 0, i8* %s8, align 1
  %printf11 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]*
      @fmt, i32 0, i32 0), i8* %string3)
  store i32 %printf11, i32* %out, align 4
  br label %merge
}
```

6.2 Test Suite

The test suite for the compiler is composed of the following files:

binop_bool.cll
binop_float.cll
binop_int.cll
binop_list.cll
binop_string.cll
bool.cll
change_variable_value.cll
dadd.cll
dadd_with_ids.cll
dfold_empty_dict.cll
dget.cll
dict_functions.cll
dict_of_bool_string.cll
dict_of_dict_dict.cll
dict_of_int_dict.cll
dict_of_int_int.cll
dict_of_int_string.cll
dict_of_string_string.cll
dmap_empty_dict.cll
example_program.cll
example_program_input
fail_binop_diff_typs.cll
fail_binop_wrong_typ_add.cll
fail_binop_wrong_typ_and.cll
fail_binop_wrong_typ_concat.cll
fail_binop_wrong_typ_div.cll
fail_binop_wrong_typ_equal.cll
fail_binop_wrong_typ_greater.cll
fail_binop_wrong_typ_less.cll
fail_binop_wrong_typ_mod.cll
fail_binop_wrong_typ_mult.cll
fail_binop_wrong_typ_or.cll
fail_binop_wrong_typ_sub.cll
fail_child_acc_non_typdef.cll
fail_child_acc_undeclared_child.cll
fail_comments.cll
fail_dget_not_in_dict.cll
fail_dictlit_typ_inconsistency1.cll
fail_dictlit_typ_inconsistency2.cll
fail_duplicate_builtin_var.cll
fail_duplicate_typ.cll
fail_duplicate_typdef.cll
fail_illegal_char.cll
fail_incompatible_cast_to_float.cll
fail_incompatible_cast_to_int.cll
fail_incompatible_cast_to_string.cll
fail_incompatible_cast_to_usertyp.cll
fail_invalid_typ_cast.cll
fail_invalid_typ_dict1.cll
fail_invalid_typ_dict2.cll
fail_invalid_typ_formal.cll
fail_invalid_typ_list.cll
fail_lget_empty_list.cll
fail_lremove_empty_list.cll
fail_match_expr_incompatible_typ.cll
fail_match_invalid_typ.cll
fail_match_no_default_bytyp.cll
fail_match_no_default_byval.cll
fail_no_contents.cll
fail_none_dict1.cll
fail_none_dict2.cll
fail_none_formal.cll
fail_none_list.cll
fail_not_bool_cond_ifelse.cll
fail_not_bool_cond_while.cll
fail_not_expr_last_stmt_else.cll
fail_not_expr_last_stmt_fun.cll
fail_not_expr_last_stmt_if.cll
fail_not_expr_last_stmt_match_bytyp.cll
fail_not_expr_last_stmt_match_byval.cll
fail_parsing_error.cll
fail_typ_inconsistency_dict1.cll
fail_typ_inconsistency_dict2.cll
fail_typ_inconsistency_list.cll
fail_typdef_assign_incompatible_child_typ.cll
fail_typdef_assign_undeclared_child.cll
fail_typdef_assign_undefined.cll
fail_undefined_id.cll
fail_undefined_utdid.cll
fail_unop_wrong_typ_neg.cll
fail_unop_wrong_typ_not.cll
float_cast.cll
fun.cll
fun_list.cll
fun_string.cll
funlit_call.cll
funlit_call_with_args.cll
if_else.cll
if_else_in_dict.cll
if_else_in_fun.cll
if_else_in_list.cll
int.cll
int_cast.cll
lfold_empty_list.cll
list_functions.cll
list_of_int.cll
list_of_list.cll
list_of_strings.cll
lmap.cll
lmap_empty_list.cll
match_bytyp.cll
match_byvalue.cll
regex.cll
scope.cll
stdin.cll
string.cll
string_cast.cll
string_functions.cll
test_input
typcomp.cll
unicode.cll
unop_bool.cll
unop_float.cll
unop_int.cll
usertyp.cll
usertyp_and_usertypdef.cll
usertypdef.cll
while.cll

These tests are integration tests that ensure that specific features of the language work as intended. There are also tests that are intended to fail to ensure the compiler catches invalid code at various points in the compilation pipeline.

6.3 Automation

6.3.1 testall.zsh

These test were automated to be compiled and have their program output check against intended output. The file `testall.zsh` performed the automation testing. During the testing process, the success/failure of each test was print to stdout and more verbose information (the contents of the LLVM file, the program output, error messages) were written to test.out. This file is as follows:

```
#!/bin/zsh

red=‘tput setaf 1‘
green=‘tput setaf 2‘
reset=‘tput sgr0‘

src_dir="test"
build_dir="cll_build"

touch test.out
output="***********************************\n"
output+="THESE TESTS SHOULD PASS\n"
output+="***********************************\n"
output+="\n"
for t in ./$src_dir/*.cll; do
    filename=${t%.cll}
    filename=${filename##*/}

    if [[ "${filename}" == *"fail"* ]]; then
        continue
    fi

    output+="***********************************"
    output+="***********************************\n"
    to_stdout="TESTING FILE: ${t}"
    output+="${to_stdout}\n"
    output+="***********************************\n"

    output+="LL File\n\n"
    output+="$(./cll.native $t)\n"
    output+="***********************************\n"

    output+="File Contents\n\n"
    output+="$(cat $t)\n"
    output+="***********************************\n"

    output+="Program Output\n\n"

    output+="$(./bin/make_test_exe.sh $filename 2>&1)\n"

    if [ $? -eq 0 ]; then
        if [ "${filename}" = "stdin" ]; then
            progoutput="$(cat ${src_dir}/test_input | ./${build_dir}/${filename})"
        else
            if [ "${filename}" = "example_program" ]; then
                progoutput="$(cat ${src_dir}/example_program_input | ./${build_dir}/${
                    filename})"
            else
                progoutput="$(./${build_dir}/${filename})"
            fi
```

```
        fi
        outputfile="${build_dir}/${filename}.out"
        echo $progoutput > $outputfile
        outputfilecheck="${src_dir}/${filename}.out"
        if cmp -s $outputfile $outputfilecheck; then
            echo "${green}[ ✓ ] ${to_stdout} ${reset}"
        else
            echo "${red}[ X ] !!!${to_stdout} output failed!!!${reset}"
        fi
        output+="${fileoutput}\n"
    else
        echo "${red}[ X ] !!!${to_stdout} compilation failed!!!${reset}"
    fi

    output+="***********************************"
    output+="***********************************\n"
    output+="\n"
done

output+="***********************************\n"
output+="THESE TESTS SHOULD FAIL\n"
output+="***********************************\n"
output+="\n"
for t in ./$src_dir/*.cll; do
    filename=${t%.cll}
    filename=${filename##*/}

    if ! [[ "${filename}" == *"fail"* ]]; then
        continue
    fi

    output+="***********************************\n"
    to_stdout="TESTING FILE: ${t}"
    output+="${to_stdout}\n"
    output+="***********************************\n"
    output+="$(./cll.native $t 2>&1)\n"
    if [ $? -eq 2 ]; then
        echo "${green}[ ✓ ] ${to_stdout}${reset}"
    else
    output+="$(./${build_dir}/${filename} 2>&1)\n"
        if [ $? -ne 0 ]; then
            echo "${green}[ ✓ ] ${to_stdout}${reset}"
        else
            echo "${red}[ X ] !!!${to_stdout} failed to fail!!!${reset}"
        fi
    fi
    output+="***********************************\n"
    output+="\n"
done
echo $output > test.out
```

When a test is run successfully, it is preceded with a `[ ✓ ]`. If a test fails, it is preceded with a `[ X ]`. Note: tests that are meant to fail will display `[ ✓ ]` upon failure and `[ X ]` if they manage to compile successfully. Following is a portion of the output from `testall.zsh`:

```
[ ✓ ] TESTING FILE: ./test/usertyp.cll
[ ✓ ] TESTING FILE: ./test/usertyp_and_usertypdef.cll
[ ✓ ] TESTING FILE: ./test/usertypdef.cll
[ ✓ ] TESTING FILE: ./test/while.cll
[ ✓ ] TESTING FILE: ./test/fail_binop_diff_typs.cll
[ ✓ ] TESTING FILE: ./test/fail_binop_wrong_typ_add.cll
[ ✓ ] TESTING FILE: ./test/fail_binop_wrong_typ_and.cll
[ ✓ ] TESTING FILE: ./test/fail_binop_wrong_typ_concat.cll
```

6.3.2 make_test_exe.sh

The file `make_test_exe.sh` is used for the compilation of individual test files and is as follows:

```
#!/bin/bash
red=‘tput setaf 1‘
green=‘tput setaf 2‘
reset=‘tput sgr0‘

for t in ./test/*.cll; do
    filename=${t%.cll}
    filename=${filename##*/}

    if [[ "${filename}" == *"fail"* ]]; then
        continue
    fi

    echo "**********************************"
    if [[ -f "test/${filename}.out" ]]; then
        echo "OUTPUT ALREADY EXISTS FOR ${filename}"
    else
        echo "MAKING OUTPUT FOR ${filename}"
        ./bin/make_test_exe.sh $filename
        if [ $? -eq 2 ]; then
            echo "${red}!!!NO OUTPUT FOR $filename!!!${reset}"
            continue
        fi
        outfile="test/${filename}.out"
        if [ "${filename}" = "stdin" ]; then
            cat test/test_input | ./cll_build/$filename
            output=$(cat test/test_input | ./cll_build/$filename)
        else
            if [ "${filename}" = "example_program" ]; then
                cat test/example_program_input | ./cll_build/$filename
                output=$(cat test/example_program_input | ./cll_build/$filename)
            else
                ./cll_build/$filename
                output=$(./cll_build/$filename)
            fi
        fi
        if [ $? -eq 139 ]; then
                echo "${red}!!!ERROR: SEGFAULT!!!"
            echo "${red}!!!NO OUTPUT FOR $filename!!!${reset}"

        else
            echo "${green}OUTPUT OK${reset}"
        fi
        echo "${output}" > $outfile
    fi
done
```

6.3.3 make_test_out.sh

The intended program output files were also generated, with sufficient printing at time of generation to ensure that the generated output indeed matched the intended output. The file `make_test_out.sh` performed the automated output generation. This file is as follows:

```
#!/bin/bash
red=‘tput setaf 1‘
green=‘tput setaf 2‘
reset=‘tput sgr0‘

for t in ./test/*.cll; do
    filename=${t%.cll}
```

```
    filename=${filename##*/}

    if [[ "${filename}" == *"fail"* ]]; then
        continue
    fi

    echo "*********************************"
    if [[ -f "test/${filename}.out" ]]; then
        echo "OUTPUT ALREADY EXISTS FOR ${filename}"
    else
        echo "MAKING OUTPUT FOR ${filename}"
        ./bin/make_test_exe.sh $filename
        if [ $? -eq 2 ]; then
            echo "${red}!!!NO OUTPUT FOR $filename!!!${reset}"
            continue
        fi
        outfile="test/${filename}.out"
        if [ "${filename}" = "stdin" ]; then
            cat test/test_input | ./cll_build/$filename
            output=$(cat test/test_input | ./cll_build/$filename)
        else
            if [ "${filename}" = "example_program" ]; then
                cat test/example_program_input | ./cll_build/$filename
                output=$(cat test/example_program_input | ./cll_build/$filename)
            else
                ./cll_build/$filename
                output=$(./cll_build/$filename)
            fi
        fi
        if [ $? -eq 139 ]; then
                echo "${red}!!!ERROR: SEGFAULT!!!"
            echo "${red}!!!NO OUTPUT FOR $filename!!!${reset}"

        else
            echo "${green}OUTPUT OK${reset}"
        fi
        echo "${output}" > $outfile
    fi
done
```

Here is a portion of an example output from running `make_test_out.sh`:

```
*********************************
MAKING OUTPUT FOR binop_list
a b c d e f
OUTPUT OK
*********************************
MAKING OUTPUT FOR binop_string
hello world!
OUTPUT OK
*********************************
OUTPUT ALREADY EXISTS FOR bool
*********************************
OUTPUT ALREADY EXISTS FOR change_variable_value
*********************************
OUTPUT ALREADY EXISTS FOR dadd
*********************************
OUTPUT ALREADY EXISTS FOR dadd_with_ids
*********************************
...
```

6.3.4 Makefile

For completeness's sake, I will also include the `Makefile` I used during the development process, which compiled the `cll.native` file and the C library files:

```
.PHONY: all
all: clean cll.native lib

.PHONY: clean
clean:
	ocamlbuild -clean
	./bin/clean.zsh

.PHONY: lib
lib: find_prime.o hash_table.o linked_list.o regex.o malloc_manager.o stdin.o

.PHONY: test
test:
	./bin/testall.zsh

cll.native:
	opam config exec -- \
	ocamlbuild -use-ocamlfind cll.native

find_prime.o:
	gcc -o cll_build/find_prime.o -c lib/find_prime.c

hash_table.o:
	gcc -o cll_build/hash_table.o -c lib/hash_table.c

linked_list.o:
	gcc -o cll_build/linked_list.o -c lib/linked_list.c

regex.o:
	gcc -o cll_build/regex.o -c lib/regex.c

malloc_manager.o:
	gcc -o cll_build/malloc_manager.o -c lib/malloc_manager.c

stdin.o:
	gcc -o cll_build/stdin.o -c lib/stdin.c

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

Chapter 7

Lessons Learned

7.1 Reflections

I'll start off by saying I learned that attempting a monumental project is best attacked in stages of incremental steps. Trying to contemplate the entirety of an application will just start your head spinning and make creating actionable steps all the more difficult. When I started this project, I thought I would be able to program large chunks of code at a time, but I learned that adding small changes to the code base were a more efficient way of implementing small and large features alike. I also learned that its ok to leave code in a less-than-optimal state, as oftentimes obsessing over thoroughness of code prevents one from understanding the small successes achieved over time. Also, committing code that works before moving onto trying to fix something else ensures you always have a reliable checkpoint to lean on in case things go really wrong.

7.2 Recommendations to Future Groups

Always keep in mind the end goal of your compiler. Reduce work whenever possible and think before you code. Understanding exactly *what* you're trying to accomplish helps you understand *how* you might accomplish it. Don't be afraid to scrap a "great" idea if it's going to be too complex/difficult/finicky.

Chapter 8

Appendix

8.1 OCaml Files

8.1.1 scanner.mll

```
(* Ocamllex scanner for CLL *)
(* Author: Annalise Mariottini (aim2120) *)

{
open Parser

let line_num: int ref = ref 1
}

let digit = ['0' - '9']
let digits = digit+
let lowercase = ['a' - 'z']
let uppercase = ['A' - 'Z']

rule token = parse
  [' ' '\t' '\r'] { token lexbuf } (* Whitespace *)
| '\n'    { line_num := !line_num + 1; token lexbuf }
| "{#"    { comment lexbuf }        (* Comments *)
| "#"    { onelinecomment lexbuf }
| '('     { LPAREN }
| ')'     { RPAREN }
| '{'     { LCURLY }
| '}'     { RCURLY }
| '['     { LSQUARE }
| ']'     { RSQUARE }
| '<'     { LANGLE }
| '>'     { RANGLE }
| ':'     { COLON }
| ';'     { SEMI }
| ','     { COMMA }
| '.'     { DOT }
| '-'     { MINUS }
| '*'     { TIMES }
| '/'     { DIVIDE }
| '%'     { MOD }
| '+'     { PLUS }
| '^'     { CONCAT }
| "!"     { NOT }
| "&&"    { AND }
```

```
| "||"    { OR }
| "~="    { TYPEQ }
| "=="    { EQ }
| '='     { ASSIGN }
| "int"   { INT }
| "bool"  { BOOL }
| "float" { FLOAT }
| "string" { STRING }
| "regex" { REGEX }
| "list"  { LIST }
| "dict"  { DICT }
| "fun"   { FUN }
| "type"  { TYP }
| "typedef" { TYPDEF }
| "match" { MATCH }
| "byvalue"{ BYVAL }
| "bytype" { BYTYP }
| "default"{ DEFAULT }
| "dowhile" { WHILE }
| "if"    { IF }
| "else"  { ELSE }
| digits as lxm { INTLIT(int_of_string lxm) }
| "true"  { BOOLLIT(true) }
| "false" { BOOLLIT(false) }
| digits '.' digit* as lxm { FLOATLIT(lxm) }
| '\'' [^ '\'']* '\'' as lxm { let s = List.hd (List.tl (String.split_on_char '\'' lxm))
    in STRLIT(s) }
| '"' [^ '"']* '"' as lxm { let s = List.hd (List.tl (String.split_on_char '"' lxm)) in
    RELIT(s) }
| lowercase ['a'-'z' 'A'-'Z' '0'-'9' '_']* as lxm { ID(lxm) }
| uppercase ['a'-'z' 'A'-'Z' '0'-'9' '_']* as lxm { UT(lxm) }
| '$' uppercase ['a'-'z' 'A'-'Z' '0'-'9' '_']* as lxm { UTD(lxm) }
| eof { EOF }
| _ as char { raise (Failure("illegal character " ^ Char.escaped char)) }

and comment = parse
 '\n' { line_num := !line_num + 1; comment lexbuf }
| "#}" { token lexbuf }
| _   { comment lexbuf }

and onelinecomment = parse
  '\n' { line_num := !line_num + 1; token lexbuf }
| _   { onelinecomment lexbuf }
```

8.1.2 ast.ml

```
(* CLL Abstract Syntax Tree and functions for printing it *)
(* Author: Annalise Mariottini (aim2120) *)

type binop = Mult | Div | Mod | Add | Sub | Concat | And | Or | Equal | Greater | Less

type uop = Neg | Not

type typ = Int | Bool | Float | String | Regex
  | List of typ
  | Dict of typ * typ
  | Fun of typ list * typ
  | UserTyp of string
  | UserTypDef of string

type typ_or_def = TypMatch of typ | DefaultTyp

type expr =
```

```
    IntLit of int
  | FloatLit of string
  | BoolLit of bool
  | StrLit of string
  | ReLit of string
  | ListLit of typ * expr list
  | DictLit of typ * typ * (expr * expr) list
  | FunLit of funlit
  | TypComp of expr * typ
  | Binop of expr * binop * expr
  | Unop of uop * expr
  | Cast of typ list * expr
  | ChildAcc of expr * string
  | Assign of string * expr
  | TypDefAssign of typ * string * (string * expr) list
  | Id of string
  | FunCall of expr * expr list
  | Match of mtch
  | IfElse of ifelse
  | While of whle
  | Expr of expr
and funlit = {
    formals: (typ * string) list;
    ftyp: typ;
    fblock: stmt list;
}
and mtch = {
    minput: expr;
    mtyp: typ;
    matchlist: matchlist;
}
and matchlist =
    ValMatchList of (expr_or_def * stmt list) list
  | TypMatchList of (typ_or_def * stmt list) list
and expr_or_def = ExprMatch of expr | DefaultExpr
and ifelse = {
    icond: expr;
    ityp: typ;
    ifblock: stmt list;
    elseblock: stmt list;
}
and whle = {
    wcond: expr;
    wtyp: typ;
    wblock: stmt list;
}
and stmt =
    ExprStmt of expr
  | TypDecl of string * (string * typ) list
  | TypDefDecl of string * (typ * string) list

type program = stmt list

(* Pretty-printing functions *)

let string_of_binop = function
    Mult -> "*"
  | Div -> "/"
  | Mod -> "%"
  | Add -> "+"
  | Sub -> "-"
  | Concat -> "^"
  | And -> "&&"
```

```
  | Or -> "||"
  | Equal -> "=="
  | Greater -> ">"
  | Less -> "<"

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

let rec string_of_typ = function
    Int -> "int"
  | Bool -> "bool"
  | Float -> "float"
  | String -> "string"
  | Regex -> "regex"
  | List(t) -> "list" ^ "<" ^ string_of_typ t ^ ">"
  | Dict(t1,t2) -> "dict" ^ "<" ^ string_of_typ t1 ^ "," ^ string_of_typ t2 ^ ">"
  | Fun(f,t) -> "fun" ^ "<" ^ String.concat ", " (List.map (fun t -> string_of_typ t) f) ^
        ":" ^ string_of_typ t ^ ">"
  | UserTyp(u) -> u
  | UserTypDef(u) -> u

let string_of_typ_or_def = function
    TypMatch(t) -> string_of_typ t
  | DefaultTyp -> "default"

let rec string_of_expr = function
    IntLit(i) -> string_of_int i
  | FloatLit(f) -> f
  | BoolLit(b) -> if b then "true" else "false"
  | StrLit(s) -> "'" ^ s ^ "'"
  | ReLit(r) -> "\"" ^ r ^ "\""
  | ListLit(t, l) -> "<" ^ string_of_typ t ^ ">" ^ "[" ^ String.concat ", " (List.map
      string_of_expr l) ^ "]"
  | DictLit(t1, t2, d) -> "<" ^ string_of_typ t1 ^ "," ^ string_of_typ t2 ^ ">" ^ "{" ^
     String.concat "," (List.map (fun p -> string_of_expr (fst p) ^ ":" ^ string_of_expr
         (snd p)) d) ^ "}"
  | FunLit(f) -> "<" ^ String.concat ", " (List.map (fun p -> string_of_typ (fst p) ^ " "
      ^ snd p) f.formals) ^ ":" ^
    string_of_typ f.ftyp ^ ">{" ^ string_of_stmtblock f.fblock ^ "}"
  | TypComp(e,t) -> string_of_expr e ^ "~=" ^ string_of_typ t
  | Binop(e1, o, e2) ->
     string_of_expr e1 ^ " " ^ string_of_binop o ^ " " ^ string_of_expr e2
  | Unop(o, e) -> string_of_uop o ^ string_of_expr e
  | ChildAcc(e, s) -> string_of_expr e ^ "." ^ s
  | Cast(t, e) -> "(" ^ String.concat ", " (List.map string_of_typ t) ^ ")" ^
      string_of_expr e
  | Assign(v, e) -> v ^ " = " ^ string_of_expr e
  | TypDefAssign(t, v, l) -> string_of_typ t ^ " " ^ v ^ " = {" ^ String.concat "" (List.
      map (fun p -> fst p ^ " = " ^ string_of_expr (snd p) ^ ";") l) ^ "}"
  | FunCall(e, l) -> string_of_expr e ^ "(" ^ String.concat ", " (List.map string_of_expr
      l) ^ ")"
  | Match(m) -> "match:" ^ string_of_typ m.mtyp ^ " (" ^ string_of_expr m.minput ^ ")" ^
      string_of_matchlist m.matchlist
  | IfElse(i) -> "if:" ^ string_of_typ i.ityp ^ " (" ^ string_of_expr i.icond ^ ") {" ^
     string_of_stmtblock i.ifblock ^ "} else {" ^ string_of_stmtblock i.elseblock ^ "}"
  | While(w) -> "while:" ^ string_of_typ w.wtyp ^ " (" ^ string_of_expr w.wcond ^ ") {" ^
      string_of_stmtblock w.wblock ^ "}"
  | Id(v) -> v
  | Expr(e) -> "(" ^ string_of_expr e ^ ")"
and string_of_expr_or_def = function
    ExprMatch(e) -> string_of_expr e
  | DefaultExpr -> "default"
```

```
and string_of_matchlist = function
    ValMatchList(l) -> " byvalue {" ^
    String.concat "" (List.map (fun p -> string_of_expr_or_def (fst p) ^ " {" ^
        string_of_stmtblock (snd p) ^ "}") l) ^ "}"
  | TypMatchList(l) -> " bytype {" ^
    String.concat "" (List.map (fun p -> string_of_typ_or_def (fst p) ^ " {" ^
        string_of_stmtblock (snd p) ^ "}") l) ^ "}"
and string_of_stmt = function
    ExprStmt(e) -> string_of_expr e
  | TypDecl(v, l) -> "type " ^ v ^ " = {" ^
      String.concat ", " (List.map (fun p -> fst p ^ "<" ^ string_of_typ (snd p) ^ ">") l)
           ^ "}"
  | TypDefDecl(v, l) -> "typedef " ^ v ^ " = {" ^
      String.concat ";" (List.map (fun p -> string_of_typ (fst p) ^ " " ^ snd p) l) ^ "}"

and string_of_stmtblock l = String.concat "" (List.map (fun e -> string_of_stmt e ^ ";\n")
      l)

let string_of_program stmtblock =
    string_of_stmtblock stmtblock
```

8.1.3 parser.mly

```
/* Ocamlyacc parser for CLL */
/* Author: Annalise Mariottini (aim2120) */

%{
open Ast
%}

%token LPAREN RPAREN LCURLY RCURLY LSQUARE RSQUARE LANGLE RANGLE COLON SEMI COMMA
%token DOT MINUS TIMES DIVIDE MOD PLUS CONCAT
%token NOT AND OR TYPEQ EQ LT GT ASSIGN
%token INT BOOL FLOAT STRING REGEX LIST DICT FUN
%token TYP TYPDEF
%token MATCH BYVAL BYTYP DEFAULT WHILE IF ELSE
%token <int> INTLIT
%token <bool> BOOLLIT
%token <string> FLOATLIT STRLIT RELIT ID UT UTD
%token EOF

%start program
%type <Ast.program> program

%left ELSE
%right ASSIGN
%left OR
%left AND
%left TYPEQ
%left EQ
%right LPAREN
%left LANGLE RANGLE
%left CONCAT
%left PLUS MINUS
%left TIMES DIVIDE MOD
%right NOT
%right RPAREN
%right DOT

%%

program:
    stmtblock EOF { List.rev $1 }
```

```
stmtblock:
      stmtblock stmt SEMI { $2::$1 }
    | stmt SEMI             { [$1] }

stmt:
    | expr { ExprStmt($1) }
    | TYP ID ASSIGN LCURLY utyplist RCURLY { TypDecl($2, $5) }
    | TYPDEF UTD ASSIGN LCURLY decllist RCURLY { TypDefDecl($2, $5) }

utyplist:
      utyplist COMMA UT LANGLE typ RANGLE { ($3,$5)::$1 }
    | UT LANGLE typ RANGLE { [($1,$3)] }

decllist:
      decllist typ ID SEMI { ($2,$3)::$1 }
    | typ ID SEMI { [($1,$2)] }

expr:
      INTLIT { IntLit($1) }
    | FLOATLIT { FloatLit($1) }
    | BOOLLIT { BoolLit($1) }
    | STRLIT { StrLit($1) }
    | RELIT { ReLit($1) }
    | LANGLE typ RANGLE LSQUARE exprlist_opt RSQUARE { ListLit($2, $5) }
    | LANGLE typ COMMA typ RANGLE LCURLY exprpairlist_opt RCURLY { DictLit($2, $4, $7) }
    | LANGLE formallist_opt COLON typ RANGLE LCURLY stmtblock RCURLY { FunLit({formals=$2;
        ftyp=$4; fblock=(List.rev $7);}) }
    | expr TYPEQ typ { TypComp($1, $3) }
    | expr AND expr { Binop($1, And, $3) }
    | expr OR expr { Binop($1, Or, $3) }
    | expr EQ expr { Binop($1, Equal, $3) }
    | expr LANGLE expr { Binop($1, Less, $3) }
    | expr RANGLE expr { Binop($1, Greater, $3) }
    | expr CONCAT expr { Binop($1, Concat, $3) }
    | expr PLUS expr { Binop($1, Add, $3) }
    | expr MINUS expr { Binop($1, Sub, $3) }
    | expr MOD expr { Binop($1, Mod, $3) }
    | expr TIMES expr { Binop($1, Mult, $3) }
    | expr DIVIDE expr { Binop($1, Div, $3) }
    | NOT expr { Unop(Not, $2) }
    | MINUS expr %prec NOT { Unop(Neg, $2) }
    | LPAREN typlist RPAREN expr { Cast(List.rev $2, $4) }
    | expr DOT ID { ChildAcc($1, $3) }
    | ID ASSIGN expr { Assign($1, $3) }
    | typ ID ASSIGN LCURLY initlist RCURLY { TypDefAssign($1, $2, (List.rev $5)) }
    | ID { Id($1) }
    | expr LPAREN exprlist_opt RPAREN { FunCall($1, $3) }
    | MATCH COLON typ LPAREN expr RPAREN matchlist { Match({minput=$5; mtyp=$3; matchlist=
       $7;}) }
    | IF COLON typ LPAREN expr RPAREN LCURLY stmtblock RCURLY ELSE LCURLY stmtblock RCURLY
        { IfElse({icond=$5; ityp=$3; ifblock=(List.rev $8); elseblock=(List.rev $12);}) }
    | WHILE COLON typ LPAREN expr RPAREN LCURLY stmtblock RCURLY { While({wcond=$5; wtyp=
       $3; wblock=(List.rev $8);}) }
    | LPAREN expr RPAREN { Expr($2) }

typlist_opt:
    /* nothing */ { [] }
    | typlist { List.rev $1 }

typlist:
    typlist COMMA typ { $3::$1 }
    | typ { [$1] }
```

```
formallist_opt:
      /* nothing */ { [] }
    | formallist { List.rev $1 }

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

exprlist_opt:
    /* nothing */ { [] }
    | exprlist { List.rev $1 }

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

exprpairlist_opt:
      /* nothing */ { [] }
    | exprpairlist { List.rev $1 }

exprpairlist:
      exprpairlist COMMA expr COLON expr { ($3,$5)::$1 }
    | expr COLON expr { [($1,$3)] }

initlist:
      initlist ID ASSIGN expr SEMI { ($2,$4)::$1 }
    | ID ASSIGN expr SEMI { [($1, $3)] }

matchlist:
      BYVAL LCURLY valuematchlist RCURLY { ValMatchList(List.rev $3) }
    | BYTYP LCURLY typmatchlist RCURLY { TypMatchList(List.rev $3) }

valuematchlist:
      valuematchlist expr_or_def LCURLY stmtblock RCURLY { ($2,(List.rev $4))::$1 }
    | expr_or_def LCURLY stmtblock RCURLY { [$1,(List.rev $3)] }

typmatchlist:
      typmatchlist typ_or_def LCURLY stmtblock RCURLY { ($2,(List.rev $4))::$1 }
    | typ_or_def LCURLY stmtblock RCURLY { [$1,(List.rev $3)] }

expr_or_def:
      DEFAULT { DefaultExpr }
    | expr { ExprMatch($1) }

typ_or_def:
      DEFAULT { DefaultTyp }
    | typ { TypMatch($1) }

typ:
      INT { Int }
    | FLOAT { Float }
    | BOOL { Bool }
    | STRING { String }
    | REGEX { Regex }
    | LIST LANGLE typ RANGLE { List($3) }
    | DICT LANGLE typ COMMA typ RANGLE { Dict($3, $5) }
    | FUN LANGLE typlist_opt COLON typ RANGLE { Fun($3, $5) }
    | UT { UserTyp($1) }
    | UTD { UserTypDef($1) }
```

8.1.4 sast.ml

```
(* CLL Semantically-checked Abstract Syntax Tree and functions for printing it *)
(* Author: Annalise Mariottini *)

open Ast

(* accomodates multiple usertypes *)
type sexpr = typ list * sx
and sx =
    SIntLit of int
  | SFloatLit of string
  | SBoolLit of bool
  | SStrLit of string
  | SReLit of string
  | SListLit of typ * sexpr list
  | SDictLit of typ * typ * (sexpr * sexpr) list
  | SFunLit of sfunlit
  | STypComp of sexpr * typ
  | SBinop of sexpr * binop * sexpr
  | SUnop of uop * sexpr
  | SCast of typ list * sexpr
  | SChildAcc of sexpr * string
  | SAssign of string * sexpr
  | STypDefAssign of typ * string * (string * sexpr) list
  | SId of string
  | SFunCall of sexpr * sexpr list
  | SMatch of smtch
  | SIfElse of sifelse
  | SWhile of swhle
and sfunlit = {
    sformals: (typ * string) list;
    sftyp: typ;
    sfblock: sstmt list;
}
and smtch = {
    sminput: sexpr;
    smtyp: typ;
    smatchlist: smatchlist;
}
and smatchlist =
    SValMatchList of (sexpr_or_def * sstmt list) list
  | STypMatchList of (typ_or_def * sstmt list) list
and sexpr_or_def = SExprMatch of sexpr | SDefaultExpr
and sifelse = {
    sicond: sexpr;
    sityp: typ;
    sifblock: sstmt list;
    selseblock: sstmt list;
}
and swhle = {
    swcond: sexpr;
    swtyp: typ;
    swblock: sstmt list;
}
and sstmt =
    SExprStmt of sexpr
  | STypDecl of string * (string * typ) list
  | STypDefDecl of string * (typ * string) list

type sprogram = sstmt list

(* Pretty-printing functions *)

let rec string_of_sexpr sexpr =
```

```
  let s = match snd sexpr with
    SIntLit(i) -> string_of_int i
  | SFloatLit(f) -> f
  | SBoolLit(b) -> if b then "true" else "false"
  | SStrLit(s) -> "'" ^ s ^ "'" (* substring removes quotes around string *)
  | SReLit(r) -> "\"" ^ r ^ "\"" (* substring removes quotes around string *)
  | SListLit(t, l) -> "<" ^ string_of_typ t ^ ">[" ^ String.concat ", " (List.map
      string_of_sexpr l) ^ "]"
  | SDictLit(t1, t2, d) -> "<" ^ string_of_typ t1 ^ "," ^ string_of_typ t2 ^ ">" ^ "{" ^
     String.concat "," (List.map (fun p -> string_of_sexpr (fst p) ^ ":" ^
          string_of_sexpr (snd p)) d) ^ "}"
  | SFunLit(f) -> "<" ^ String.concat ", " (List.map (fun p -> string_of_typ (fst p) ^ " "
       ^ snd p) f.sformals) ^ ":" ^
    string_of_typ f.sftyp ^ ">{\n" ^ string_of_sstmtblock f.sfblock ^ "}"
  | STypComp(e,t) -> string_of_sexpr e ^ "~=" ^ string_of_typ t
  | SBinop(e1, o, e2) -> string_of_sexpr e1 ^ " " ^ string_of_binop o ^ " " ^
      string_of_sexpr e2
  | SUnop(o, e) -> string_of_uop o ^ string_of_sexpr e
  | SChildAcc(e, s) -> string_of_sexpr e ^ "." ^ s
  | SCast(t, e) -> "(" ^ String.concat ", " (List.map string_of_typ t) ^ ")" ^
      string_of_sexpr e
  | SAssign(v, e) -> v ^ " = " ^ string_of_sexpr e
  | STypDefAssign(t, v, l) -> string_of_typ t ^ " " ^ v ^ " = {" ^ String.concat "" (List.
      map (fun p -> fst p ^ " = " ^ string_of_sexpr (snd p) ^ ";") l) ^ "}"
  | SFunCall(e, l) -> string_of_sexpr e ^ "(" ^ String.concat ", " (List.map
      string_of_sexpr l) ^ ")"
  | SMatch(m) -> "match:" ^ string_of_typ m.smtyp ^ " (" ^ string_of_sexpr m.sminput ^ ")"
       ^ string_of_smatchlist m.smatchlist
  | SIfElse(i) -> "if:" ^ string_of_typ i.sityp ^ " (" ^ string_of_sexpr i.sicond ^ ") {\n
      " ^
     string_of_sstmtblock i.sifblock ^ "} else {\n" ^ string_of_sstmtblock i.selseblock ^
         "}"
  | SWhile(w) -> "while:" ^ string_of_typ w.swtyp ^ " (" ^ string_of_sexpr w.swcond ^ ")
      {\n" ^ string_of_sstmtblock w.swblock ^ "}"
  | SId(v) -> v
  in "(typs: " ^ String.concat ", " (List.map string_of_typ (fst sexpr)) ^ ")" ^ s
and string_of_sexpr_or_def = function
    SExprMatch(e) -> string_of_sexpr e
  | SDefaultExpr -> "default"
and string_of_smatchlist = function
    SValMatchList(l) -> " byvalue {" ^
    String.concat "" (List.map (fun p -> "\n" ^ string_of_sexpr_or_def (fst p) ^ " {\n" ^
        string_of_sstmtblock (snd p) ^ "}") l) ^ "}"
  | STypMatchList(l) -> " bytype {" ^
    String.concat "" (List.map (fun p -> "\n" ^ string_of_typ_or_def (fst p) ^ " {\n" ^
        string_of_sstmtblock (snd p) ^ "}") l) ^ "}"
and string_of_sstmt = function
    SExprStmt(e) -> string_of_sexpr e
  | STypDecl(v, l) -> "type " ^ v ^ " = {" ^
     String.concat ", " (List.map (fun p -> fst p ^ "<" ^ string_of_typ (snd p) ^ ">") l)
         ^ "}"
  | STypDefDecl(v, l) -> "typedef " ^ v ^ " = {" ^
     String.concat ";" (List.map (fun p -> string_of_typ (fst p) ^ " " ^ snd p) l) ^ "}"
and string_of_sstmtblock l = String.concat "" (List.map (fun e -> string_of_sstmt e ^ ";\n
    ") l)

let string_of_sprogram sstmtblock =
    string_of_sstmtblock sstmtblock
```

8.1.5 semant.ml

```
(* Semantic checking for the CLL compiler *)
(* Author: Annalise Mariottini (aim2120) *)
```

```
open Ast
open Sast

module StringMap = Map.Make(String)

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

   Check each stmt of program *)

type semantic_env = {
    tvsym: (string * typ) list StringMap.t;
    (* type variable name: list of user type names and definitions *)
    tsym: (typ * typ) StringMap.t;
    (* user type name: defining type and built-in associate type *)
    tdsym: (typ * string) list StringMap.t;
    (* user typedef name: child declaration list *)
    vsym: typ list StringMap.t;
    (* user variables: all types *)
}

let check_ast ast =
    let line_num: int ref = ref 1 in
    let make_err err = raise (Failure ("!!!ERROR!!! line " ^ string_of_int !line_num ^ ":
        " ^ err)) in
    let add_built_in map (id, formals, t) =
        let new_f = Fun(formals, t) in
        if StringMap.mem id map then
            let already_exists f = (f = new_f) in
            let fun_defs = StringMap.find id map in
            if List.exists already_exists fun_defs then map
            else StringMap.add id (new_f::fun_defs) map
        else
            StringMap.add id [new_f] map
    in
    let built_in_funs =
        let l = [
            ("sprint",
                [String],
                Int);
            ("ssize",
                [String],
                Int);
            ("sfold",
                [Fun([String;String],String);String;String],
                String);
            ("lprint",
                [List(String)],
                Int);
            ("lmem",
                [List(String);String],
                Int);
            ("lget",
                [List(String);Int],
                String);
            ("ladd",
                [List(String);String],
                List(String));
            ("ladd",
                [List(String);String; Int],
                List(String));
            ("lremove",
```

```
            [List(String)],
            List(String));
        ("lremove",
            [List(String); Int],
            List(String));
        ("lsize",
            [List(String)],
            Int);
        ("lfold",
            [Fun([String;String],String);String;List(String)],
            String);
        ("lmap",
            [Fun([String],String);List(String)],
            List(String));
        ("dprint",
            [Dict(String,String)],
            Int);
        ("dmem",
            [Dict(String,String); String],
            Bool);
        ("dget",
            [Dict(String,String); String],
            String);
        ("dadd",
            [Dict(String,String); String; String],
            Dict(String,String));
        ("dremove",
            [Dict(String,String); String],
            Dict(String,String));
        ("dsize",
            [Dict(String,String)],
            Int);
        ("dfold",
            [Fun([String;String;String],String);String;Dict(String,String)],
            String);
        ("dmap",
            [Fun([String;String],String);Dict(String,String)],
            Dict(String,String));
        ("dkeys",
            [Dict(String,String)],
            List(String));
        ("rematch",
            [Regex;String],
            Bool);
        ("resub",
            [Regex;String;String;Int],
            String);
    ] in
    List.fold_left add_built_in StringMap.empty l
in
let add_built_in_list vsym t =
    let l = [
        ("lprint", [List(t)], Int);
        ("lmem", [List(t);t], Int);
        ("lget", [List(t);Int], t);
        ("ladd", [List(t);t], List(t));
        ("ladd", [List(t);t; Int], List(t));
        ("lremove", [List(t)], List(t));
        ("lremove", [List(t); Int], List(t));
        ("lsize", [List(t)], Int);
    ] in
    List.fold_left add_built_in vsym l
in
```

```
let add_built_in_dict vsym t1 t2 =
    let l = [
        ("dprint", [Dict(t1,t2)], Int);
        ("dmem", [Dict(t1,t2);t1], Bool);
        ("dget", [Dict(t1,t2);t1], t2);
        ("dadd", [Dict(t1,t2);t1; t2], Dict(t1,t2));
        ("dremove", [Dict(t1,t2); t1], Dict(t1,t2));
        ("dsize", [Dict(t1,t2)], Int);
        ("dkeys", [Dict(t1,t2)], List(t1));
    ] in
    List.fold_left add_built_in vsym l
in
let empty_env = {
    tvsym=StringMap.empty;
    tsym=StringMap.empty;
    tdsym=StringMap.empty;
    vsym=(StringMap.add "stdin" [List(String)] built_in_funs);
}
in
let add_env_tvsym env tvsym = {
    tvsym=tvsym;
    tsym=env.tsym;
    tdsym=env.tdsym;
    vsym=env.vsym;
}
in
let add_env_tsym env tsym = {
    tvsym=env.tvsym;
    tsym=tsym;
    tdsym=env.tdsym;
    vsym=env.vsym;
}
in
let add_env_tdsym env tdsym = {
    tvsym=env.tvsym;
    tsym=env.tsym;
    tdsym=tdsym;
    vsym=env.vsym;
}
in
let add_env_vsym env vsym = {
    tvsym=env.tvsym;
    tsym=env.tsym;
    tdsym=env.tdsym;
    vsym=vsym;
}
in
let find_in_map map id err =
    try StringMap.find id map
    with Not_found -> make_err err
in
let to_assc_typ tsym t = match t with
    UserTyp(ut) -> let (_, at) = find_in_map tsym ut (ut ^ " not a defined type")
        in at
    | _ -> t
in
let add_typ_var tvsym (id, (l:(string * typ) list)) =
    let already_decl_err = "type variable " ^ id ^ " may not be redefined" in
    match id with
        _ when StringMap.mem id tvsym -> make_err already_decl_err
        | _ -> StringMap.add id l tvsym
in
let add_typ tsym (id, (typ:typ)) =
```

```
    let already_decl_err = "type " ^ id ^ " may not be redefined" in
    let assc_typ = to_assc_typ tsym typ in
    match id with
        _ when StringMap.mem id tsym -> make_err already_decl_err
        | _ -> StringMap.add id (typ, assc_typ) tsym
in
let add_typdef tdsym (id, (l:(typ * string) list)) =
    let already_decl_err = "typedef " ^ id ^ " may not be redefined" in
    match id with
        _ when StringMap.mem id tdsym -> make_err already_decl_err
        | _ -> StringMap.add id l tdsym
in
let add_var vsym (id, (typlist:typ list)) =
    let built_in_err = "std lib function " ^ id ^ " may not be redefined" in
    match id with (* No duplicate functions or redefinitions of built-ins *)
        _ when StringMap.mem id built_in_funs -> make_err built_in_err
        | _ -> StringMap.add id typlist vsym
in
let rec check_typlist typlist t err =
    match typlist with
    hd::tl -> if t = hd then t else check_typlist tl t err
    | [] -> make_err err
in
let rec check_typlists typlist1 typlist2 err =
    match typlist2 with
    hd::tl -> (try check_typlist typlist1 hd err with Failure(_) -> check_typlists
         typlist1 tl err)
    | [] -> make_err err
in
let rec check_valid_typ env t =
    let t = to_assc_typ env.tsym t in
    match t with
        List(t') ->
            let vsym = check_valid_typ env t' in
            add_built_in_list vsym t'
        | Dict(t1,t2) ->
            let vsym = check_valid_typ env t1 in
            let vsym = check_valid_typ (add_env_vsym env vsym) t2 in
            add_built_in_dict vsym t1 t2
        | UserTypDef(utd) ->
            let _ = find_in_map env.tdsym utd (utd ^ " not a defined type") in
            env.vsym
        | Fun(_,_) -> make_err "funs cannot be a stored/passed datatype"
        | _ -> env.vsym
in
let check_last_stmt stmts t =
    let last_stmt = List.hd stmts in (* assume already reversed *)
    let (typlist, _) = match last_stmt with
        SExprStmt(s) -> s
        | _ -> make_err "last statement of block must be an expression"
    in
    check_typlist typlist t "last expression of function does not return declared type"
in
let rec check_stmt (env, stmts) stmt =
    let output = match stmt with
    ExprStmt(e) ->
        let (e_t, se, vsym) = check_expr env e in
        let env' = add_env_vsym env vsym in
        (env', SExprStmt((e_t, se))::stmts)
    | TypDecl(id, l) ->
        let tvsym = add_typ_var env.tvsym (id, l) in
        let tsym = List.fold_left add_typ env.tsym l in
        let env' = add_env_tvsym (add_env_tsym env tsym) tvsym in
```

```
        (env', STypDecl(id, l)::stmts)
    | TypDefDecl(id, l) ->
        let tdsym = add_typdef env.tdsym (id, l) in
        let env' = add_env_tdsym env tdsym in
        (env', STypDefDecl(id,l)::stmts)
    in
    line_num := !line_num + 1;
    output
and check_expr env expr =
    match expr with
    IntLit(i) -> ([Int], SIntLit i, env.vsym)
    | FloatLit(f) -> ([Float], SFloatLit f, env.vsym)
    | BoolLit(b) -> ([Bool], SBoolLit b, env.vsym)
    | StrLit(s) -> ([String], SStrLit s, env.vsym)
    | ReLit(r) -> ([Regex], SReLit r, env.vsym)
    | ListLit(t,l) ->
        let vsym = check_valid_typ env t in
        let err = "list literal type inconsistency" in
        let check_list (l, vsym) e =
            let (e_typlist, se, vsym') = check_expr (add_env_vsym env vsym) e in
            let _ = check_typlist e_typlist t err in
            ((e_typlist, se)::l, vsym')
        in
        let (slist, vsym') = List.fold_left check_list ([], vsym) l in
        let vsym' = add_built_in_list vsym' t in
        ([List(t)], SListLit(t,(List.rev slist)), vsym')
    | DictLit(t1,t2,l) ->
        let vsym = check_valid_typ env t1 in
        let vsym = check_valid_typ (add_env_vsym env vsym) t2 in
        let keys = Hashtbl.create (List.length l) in
        let err = "dictionary literal type inconsistency" in
        let check_dict (l', vsym) (e1,e2) =
            let (e1_typlist, e1', vsym') = check_expr (add_env_vsym env vsym) e1 in
            if Hashtbl.mem keys e1' then make_err "dictionary keys must be unique"
            else Hashtbl.add keys e1' 1;
            let (e2_typlist, e2', vsym'') = check_expr (add_env_vsym env vsym') e2 in
            let _ = check_typlist e1_typlist t1 err in
            let _ = check_typlist e2_typlist t2 err in
            (((e1_typlist, e1'), (e2_typlist, e2'))::l', vsym'')
        in
        let (slist, vsym') = List.fold_left check_dict ([], vsym) l in
        let vsym' = add_built_in_dict vsym' t1 t2 in
        ([Dict(t1,t2)], SDictLit(t1, t2, List.rev slist), vsym')
    | FunLit(f) ->
        let vsym = check_valid_typ env f.ftyp in
        let check_formal vsym (t, id) =
            let vsym = check_valid_typ (add_env_vsym env vsym) t in
            add_var vsym (id, [t])
        in
        let temp_env = add_env_vsym env (List.fold_left check_formal vsym f.formals) in
        let (_, sfblock) = List.fold_left check_stmt (temp_env, []) f.fblock in
        let _ = check_last_stmt sfblock f.ftyp in
        let f' = {
            sformals = f.formals;
            sftyp = f.ftyp;
            sfblock = List.rev sfblock;
        } in
        let ret_typ = f'.sftyp in
        let formal_typs = List.map (fun (t,_) -> t) f'.sformals in
        let vsym = (match Fun(formal_typs, ret_typ) with
            Fun([t;String],t') when t = t' ->
                let vsym = add_built_in vsym ("sfold", [Fun([t;String],t);t;String], t)
                    in
```

```
          add_built_in vsym ("lfold", [Fun([t;String],t);t;List(String)], t)
      | Fun([t;x],t') when t = t' -> (
          let vsym = add_built_in vsym ("lfold", [Fun([t;x],t);t;List(x)], t) in
          if x = t' then (
              add_built_in vsym ("dmap", [Fun([x;x],x);Dict(x,x)], Dict(x,x))
          ) else vsym
      )
      | Fun([t;x;y],t') when t = t' ->
          add_built_in vsym ("dfold", [Fun([t;x;y],t);t;Dict(x,y)], t)
      | Fun([t],t') when t = t' -> (
              add_built_in vsym ("lmap", [Fun([t],t);List(t)], List(t))
      )
      | Fun([k;v],v') when v = v' -> (
              add_built_in vsym ("dmap", [Fun([k;v],v);Dict(k,v)], Dict(k,v))
      )
      | _ -> vsym
    )
    in
    ([Fun(formal_typs, ret_typ)], SFunLit(f'), vsym)
| TypComp(e,t) ->
    let err = "type comparison must be of usertype" in
    let check_for_usertyp t =
      (match t with
          UserTyp(_) -> ()
          | _ -> make_err err)
    in
    check_for_usertyp t;
    let (typlist, e', vsym) = check_expr env e in
    let vsym' = check_valid_typ (add_env_vsym env vsym) t in
    ([Bool], STypComp((typlist, e'), t), vsym')
| Binop(e1,o,e2) ->
    let (typlist1, se1, vsym) = check_expr env e1 in
    let (typlist2, se2, vsym') = check_expr (add_env_vsym env vsym) e2 in
    let e1_at = to_assc_typ env.tsym (List.hd typlist1) in
    let e2_at = to_assc_typ env.tsym (List.hd typlist2) in
    let err = "binary operation requires operands of the same type" in
    let at = if e1_at = e2_at then e1_at else make_err err in
    let typlist = match o with
    Mult | Div | Mod | Add | Sub when at = Int || at = Float -> typlist1
    | Concat when at = String || (match at with List(_) -> true | _ -> false) ->
        typlist1
    | And | Or when at = Bool -> typlist1
    | Equal | Greater | Less when at = Int || at = Float || at = String -> [Bool]
    | _ -> make_err "binary operation on operands of incorrect type"
    in
    (* binop casts to typlist of 1st operand except with comparison operators *)
    (typlist, SBinop((typlist1, se1),o,(typlist2, se2)), vsym')
| Unop(o,e) ->
    let (typlist, se, vsym) = check_expr env e in
    let at = to_assc_typ env.tsym (List.hd typlist) in
    (match o with
      Neg when at = Int || at = Float -> ()
    | Not when at = Bool -> ()
    | _ -> make_err "unary operation on operand of incorrect type");
    (typlist, SUnop(o, (typlist, se)), vsym)
| Cast(l,e) ->
    let vsym = List.fold_left (fun vsym t -> check_valid_typ (add_env_vsym env vsym
        ) t) env.vsym l in
    let at = to_assc_typ env.tsym (List.hd l) in
    let (typlist, se, vsym) = check_expr (add_env_vsym env vsym) e in
    let e_at = to_assc_typ env.tsym (List.hd typlist) in
    if at = e_at then
      (l, SCast(l, (typlist, se)), vsym)
```

```
    else
        let e_at_int = e_at = Int in
        let e_at_float = e_at = Float in
        let e_at_bool = e_at = Bool in
        let e_at_regex = e_at = Regex in
        (match at with
            Int when e_at_float -> ()
            | Float when e_at_int -> ()
            | String when e_at_int || e_at_float || e_at_bool || e_at_regex -> ()
            | _ -> make_err "incompatible cast");
        (l, SCast(l, (typlist, se)), vsym)
| ChildAcc(e,id) ->
    let (typlist, se, vsym) = check_expr env e in
    let t = List.hd typlist in (* typdef expr will only have 1 element in typlist
        *)
    let ut = match t with
        UserTypDef(ut) -> ut
        | _ -> make_err "attempting access into non-typedef variable"
    in
    let td_children = find_in_map env.tdsym ut (ut ^ " not a defined typedef") in
        (* should never fail *)
    let rec find x = function
        [] -> make_err "attempting access of undeclared typedef child"
        | hd::tl -> let (child_t,child_id) = hd in if x = child_id then child_t
            else find x tl
    in
    let child_t = find id td_children in
    ([child_t], SChildAcc((typlist, se), id), vsym)
| Assign(id,e) ->
    (if id = "stdin" then make_err "cannot redefine stdin variable");
    let (typlist, se, vsym) = check_expr env e in
    let assc_t = to_assc_typ env.tsym (List.hd typlist) in
    let old_assc_t = (try Some (
        let old_typlist = find_in_map vsym id "" in
        (to_assc_typ env.tsym (List.hd old_typlist))
    ) with Failure(_) -> None) in
    (match old_assc_t with
        Some old_assc_t -> (match old_assc_t with
            Fun(_,_) -> make_err ("cannot redefine function variable " ^ id)
            | _ when old_assc_t = assc_t -> ()
            | _ -> make_err ("variable reassignment of " ^ id ^ " must be of the
                previous type " ^ string_of_typ old_assc_t));
        | None -> ()
    );
    let vsym' = add_var vsym (id, typlist) in
    (typlist, SAssign(id, (typlist, se)), vsym')
| TypDefAssign(td,id,l) ->
    let rec get_ut = (function
        UserTyp(u) -> get_ut (to_assc_typ env.tsym (UserTyp(u)))
        | UserTypDef(ut) -> ut
        | _ -> make_err "typedef assignment using non-typedef typ"
    ) in
    let ut = get_ut td in
    let td_children = find_in_map env.tdsym ut ("typedef " ^ ut ^ " not defined")
        in
    let check_assignment (vsym, l) (id, e) =
        let (typlist, se, vsym') = check_expr (add_env_vsym env vsym) e in
        let rec find x = function
            [] -> make_err ("attempting assignment of undeclared child of typdef " ^
                ut)
            | hd::tl -> let (_, child_id) = hd in if x = child_id then hd else find
                x tl
        in
```

```
        let (child_t, child_id) = find id td_children in
        let t = check_typlist typlist child_t ("incompatible assignment to typedef
            " ^ ut ^ " child " ^ child_id) in
        (* if e has multiple types, only keep the type that matches the child
            declaration *)
        (vsym', (id, ([t], se))::l)
    in
    let (vsym, sl) = List.fold_left check_assignment (env.vsym, []) l in
    let vsym' = add_var vsym (id, [UserTypDef(ut)]) in
    ([UserTypDef(ut)], STypDefAssign(td, id, List.rev sl), vsym')
  | Id(id) ->
    let typlist = find_in_map env.vsym id (id ^ " not defined") in
    (typlist, SId(id), env.vsym)
  | FunCall(e,l) ->
    let make_actuals (vsym, actuals) a =
        let (a_typlist, a_e, vsym') = check_expr (add_env_vsym env vsym) a in
        (vsym', (a_typlist, a_e)::actuals)
    in
    let (vsym, actuals) = List.fold_left make_actuals (env.vsym, []) l in
    let actuals = List.rev actuals in
    let (typlist, se, vsym) = check_expr (add_env_vsym env vsym) e in
    let get_formals_and_ret = function
        Fun(formals, typ) -> (formals, typ)
        | _ -> make_err ("trying to call a non-function expression")
    in
    (* multiple function definitions can exist *)
    let fun_defs = List.map get_formals_and_ret typlist in
    let rec check_args formals actuals = (
        match formals, actuals with
        f_hd::f_tl, a_hd::a_tl ->
            let (a_typlist, _) = a_hd in
            ignore(check_typlist a_typlist f_hd "function argument type doesn't
                match formal definition");
            (* only keep actual argument typ that matches formal definition *)
            check_args f_tl a_tl
        | [], [] -> ()
        | _, _ -> make_err "function arguments incompatible with function
            definition"
    )
    in
    let rec try_def (formals, typ) fun_defs =
        try
            (check_args formals actuals;
            ([typ], SFunCall(([Fun(formals,typ)],se), actuals), vsym))
        with Failure(e) -> (match fun_defs with
            hd::tl -> try_def hd tl
            | [] -> make_err e)
    in
    try_def (List.hd fun_defs) (List.tl fun_defs)
  | Match(m) ->
    let vsym = check_valid_typ env m.mtyp in
    let (input_typlist, input_se, vsym) = check_expr (add_env_vsym env vsym) m.
        minput in
    let smatchlist = match m.matchlist with
        ValMatchList(l) ->
            let (last_e,_) = List.hd (List.rev l) in
            (match last_e with
                DefaultExpr -> ()
                | _ -> make_err "match block doesn't end with default block");
            let check_block blocks (e_or_d, stmts) =
                let (se_or_d, vsym') = match e_or_d with
                    ExprMatch(e) ->
                        let (e_typlist, se, vsym') = check_expr (add_env_vsym env
```

```
                        vsym) e in
                    let err = "match expression type doesn't match input type" in
                    let t = check_typlists input_typlist e_typlist err in
                    (* only keep type that matches input ahd exprmatch *)
                    (SExprMatch(([t], se)), vsym')
                  | DefaultExpr -> (SDefaultExpr, vsym)
                in
                let temp_env = add_env_vsym env vsym' in
                let (_,sstmts) = List.fold_left check_stmt (temp_env, []) stmts in
                let _ = check_last_stmt sstmts m.mtyp in
                (se_or_d, List.rev sstmts)::blocks
            in let l' = List.fold_left check_block [] l in
            SValMatchList(List.rev l')
        | TypMatchList(l) ->
            let rec find_default = function
                [] -> make_err "match block has no default block"
                | (t,_)::tl -> (match t with DefaultTyp -> () | _ -> find_default tl
                    )
            in find_default l;
            let check_block blocks (t_or_d, stmts) =
                let vsym = (match t_or_d with
                    TypMatch(t) -> check_valid_typ env t
                    | DefaultTyp -> env.vsym
                ) in
                let temp_env = add_env_vsym env vsym in
                let (_,sstmts) = List.fold_left check_stmt (temp_env, []) stmts in
                let _ = check_last_stmt sstmts m.mtyp in
                (t_or_d, List.rev sstmts)::blocks
            in let l' = List.fold_left check_block [] l in
            STypMatchList(List.rev l')
    in
    let m' = {
        sminput = (input_typlist, input_se);
        smtyp = m.mtyp;
        smatchlist = smatchlist;
    } in
    ([m'.smtyp], SMatch(m'), vsym)
| IfElse(i) ->
    let vsym = check_valid_typ env i.ityp in
    let (cond_typlist, cond_se, vsym) = check_expr (add_env_vsym env vsym) i.icond
        in
    let at = to_assc_typ env.tsym (List.hd cond_typlist) in
    (match at with
        Bool -> ()
        | _ -> make_err "if/else condition must be a boolean");
    let temp_env = add_env_vsym env vsym in
    let (_, sifblock) = List.fold_left check_stmt (temp_env, []) i.ifblock in
    let _ = check_last_stmt sifblock i.ityp in
    let (_, selseblock) = List.fold_left check_stmt (temp_env, []) i.elseblock in
    let _ = check_last_stmt selseblock i.ityp in
    let i' = {
        sicond = (cond_typlist, cond_se);
        sityp = i.ityp;
        sifblock = List.rev sifblock;
        selseblock = List.rev selseblock;

    } in
    ([i'.sityp], SIfElse(i'), vsym)
| While(w) ->
    let vsym = check_valid_typ env w.wtyp in
    let (cond_typlist, cond_se, vsym) = check_expr (add_env_vsym env vsym) w.wcond
        in
    let at = to_assc_typ env.tsym (List.hd cond_typlist) in
```

```
          (match at with
              Bool -> ()
              | _ -> make_err "while condition must be a boolean");
          let temp_env = add_env_vsym env vsym in
          let (_, sblock) = List.fold_left check_stmt (temp_env, []) w.wblock in
          let _ = check_last_stmt sblock w.wtyp in
          let w' = {
              swcond = (cond_typlist, cond_se);
              swtyp = w.wtyp;
              swblock = List.rev sblock;
          } in
          ([w'.swtyp], SWhile(w'), vsym)
      | Expr(e) ->
          check_expr env e

    in
    let (env, sast) = List.fold_left check_stmt (empty_env, []) ast
    in
(*
    let (x, _) = List.fold_left (fun (x, num) y -> print_string ("stmt" ^ string_of_int
        num); (check_stmt x y, num + 1)) ((empty_env, []), 0) ast
    in
    let sast = snd x in
*)
    (env, List.rev sast)
```

8.1.6 codegen.ml

```
(* CLL Code generation: takes a SAST and produces LLVM IR *)
(* Author: Annalise Mariottini (aim2120) *)

module L = Llvm
module A = Ast
open Semant
open Sast

module StringMap = Map.Make(String)

(* translate : Sast.program -> Llvm.module *)
let translate (env : semantic_env) (sast : sstmt list) =
    (* Create the LLVM compilation module into which
       we will generate code *)
    let context = L.global_context () in
    let the_module = L.create_module context "ConLangLang" in

    (* to store different types of lists/dicts *)
    let list_typs : (string, L.lltype) Hashtbl.t = Hashtbl.create 10 in
    let dict_typs : (string, L.lltype) Hashtbl.t = Hashtbl.create 10 in
    let ut_typs  : (string, L.lltype) Hashtbl.t = Hashtbl.create 10 in
    let utd_typs = Hashtbl.create 10 in
    let fun_name_i : int ref = ref 0 in
    let ut_i : int ref = ref 0 in

    (* Get types from the context *)
    let i32_t       = L.i32_type   context
    in let i64_t    = L.i64_type   context
    in let i1_t     = L.i1_type    context
    in let i8_t     = L.i8_type    context
    in let float_t  = L.double_type context
    in let string_t = L.pointer_type i8_t
    in let void_t   = L.void_type  context
    in
```

```
(* matches struct in lib/linked_list.c *)
let ll_node = L.named_struct_type context "ll_node" in
L.struct_set_body ll_node [|string_t; L.pointer_type ll_node|] false;

(* matches structs in lib/hash_table.c *)
let ht_entry = L.named_struct_type context "entry_s"
in L.struct_set_body ht_entry [|string_t; string_t; L.pointer_type ht_entry|] false;
let ht_t = L.named_struct_type context "hashtable_s"
in L.struct_set_body ht_t [|i32_t; i32_t; L.pointer_type (L.pointer_type ht_entry);
    i1_t|] false;

(* matches structs in lib/regex.c *)
let re_guts = L.named_struct_type context "re_guts" in
let regex_t = L.named_struct_type context "regex_t" in
L.struct_set_body regex_t [|i32_t;i64_t;string_t;L.pointer_type re_guts|] false;
let regmatch_t = L.named_struct_type context "regmatch_t" in
L.struct_set_body regmatch_t [|i64_t;i64_t|] false;

(* convenient numbers/strings *)
let zero = L.const_int i32_t 0 in
let one = L.const_int i32_t 1 in
let two = L.const_int i32_t 2 in
let max_int = L.const_int i32_t 2147483647 in
let tru = L.const_int i1_t 1 in
let fals = L.const_int i1_t 0 in
let internal_err = "internal error: " in

(* Return the LLVM type for a MicroC type *)
let str_of_ltyp ltyp =
   let janky_str =
      let s = L.string_of_lltype ltyp in
      let r = Str.regexp " " in
      Str.global_replace r "_" s
   in
   let s = (match L.classify_type ltyp with
      L.TypeKind.Struct -> (match L.struct_name ltyp with
         Some n -> n
         | None -> janky_str
      )
      | _ -> janky_str
   ) in
   let r = Str.regexp "\"" in
   Str.global_replace r "" s
in
let rec ltyp_of_typ = function
     A.Int   -> i32_t
   | A.Bool  -> i1_t
   | A.Float -> float_t
   | A.String -> string_t
   | A.Regex -> L.pointer_type regex_t
   | A.List(t)->
      (*
       * struct list { t1 *dummy; ll_node *head; }
       * dummy value is to keep track of actual values in list
       * because ll_node can only accomodate string_t
       *)
      let ltyp = ltyp_of_typ t in
      let ltyp_s = str_of_ltyp ltyp in
      let list_name = "list" ^ ltyp_s in
      L.pointer_type (if Hashtbl.mem list_typs list_name then
         Hashtbl.find list_typs list_name
      else (
         let list_t = L.named_struct_type context list_name in
```

```
            L.struct_set_body list_t [|(L.pointer_type ltyp); (L.pointer_type ll_node)
                |] false;
            Hashtbl.add list_typs list_name list_t;
            list_t
        ))
    | A.Dict(t1,t2) ->
        (*
         * struct dict { t1 *dummy; t2 *dummy; ht_t *table; }
         * dummy values are to keep track of actual values in table
         * becuase ht_t can only accomodate string_t
         *)
        let ltyp1 = ltyp_of_typ t1 in
        let ltyp2 = ltyp_of_typ t2 in
        let ltyp_s = (str_of_ltyp ltyp1) ^ (str_of_ltyp ltyp2) in
        let dict_name = "dict" ^ ltyp_s in
        L.pointer_type (if Hashtbl.mem dict_typs dict_name then
            Hashtbl.find dict_typs dict_name
        else (
            let dict_t = L.named_struct_type context dict_name in
            L.struct_set_body dict_t [|(L.pointer_type ltyp1); (L.pointer_type ltyp2);
                (L.pointer_type ht_t)|] false;
            Hashtbl.add dict_typs dict_name dict_t;
            dict_t
        ))
    | A.Fun(f,t) ->
        let ltyp = ltyp_of_typ t in
        let f_typs = Array.of_list (List.map ltyp_of_typ f) in
        (L.pointer_type (L.function_type ltyp f_typs))
    | A.UserTyp(ut) -> Hashtbl.find ut_typs ut
    | A.UserTypDef(utd) -> L.pointer_type (fst (Hashtbl.find utd_typs utd))
in

(* Creating top-level function *)
let main_t = L.function_type i32_t [||] in
let main = L.define_function "main" main_t the_module in
let builder = L.builder_at_end context (L.entry_block main) in

(* variable name -> (value, parent_function) table *)
let var_tbl : (string, (L.llvalue * L.llvalue)) Hashtbl.t = Hashtbl.create 10 in
(* func name -> locals table *)
let func_params_tbl : (string, (string, L.llvalue) Hashtbl.t) Hashtbl.t = Hashtbl.
    create 10 in
let func_tbl : (string, L.llvalue) Hashtbl.t = Hashtbl.create 10 in

(* start external functions *)
let declare_func (name, ret, args) var_arg =
    let t = if var_arg then L.var_arg_function_type ret args
    else L.function_type ret args
    in
    let func = L.declare_function name t the_module in
    func
in
let declare_funcs map (def : string * L.lltype * L.lltype array) =
    let (name, _, _) = def in
    StringMap.add name (declare_func def false) map
in

let exit_func : L.llvalue = declare_func ("exit", void_t, [|i32_t|]) false in
let ll_of_stdin_func : L.llvalue = declare_func ("ll_of_stdin", (L.pointer_type
    ll_node), [||]) false in

(* malloc manager functions *)
let init_malloc_addr_func : L.llvalue = declare_func ("init_malloc_addr", void_t,
```

```
    [||]) false in
let add_malloc_addr_func : L.llvalue = declare_func ("add_malloc_addr", void_t, [|L.
    pointer_type i8_t|]) false in
let free_malloc_addrs_func : L.llvalue = declare_func ("free_malloc_addrs", void_t,
    [||]) false in

ignore(L.build_call init_malloc_addr_func [||] "" builder);

(* string stuff *)
let printf_func : L.llvalue = declare_func ("printf", i32_t, [|string_t|]) true in
let snprintf_func : L.llvalue = declare_func ("snprintf", i32_t, [|string_t; i32_t;
    string_t|]) true in
let strcpy_func : L.llvalue = declare_func ("strcpy", string_t, [|string_t; string_t
    |]) false in
let strcat_func : L.llvalue = declare_func ("strcat", string_t, [|string_t; string_t
    |]) false in
let strlen_func : L.llvalue = declare_func ("strlen", i32_t, [|string_t;|]) false in
let str_format = L.build_global_stringptr "%s\n" "fmt" builder in
let i32_format = L.build_global_stringptr "%d" "fmt" builder in
let float_format = L.build_global_stringptr "%f" "fmt" builder in

(* start stdlib functions *)
(* linked list functions *)
let ll_create = "ll_create" in
let ll_add   = "ll_add" in
let ll_append = "ll_append" in
let ll_next  = "ll_next" in
let ll_mem   = "ll_mem" in
let ll_get   = "ll_get" in
let ll_remove = "ll_remove" in
let ll_print = "ll_print" in
let ll_size  = "ll_size" in
let ll_defs = [
    (ll_create, (L.pointer_type ll_node), [|string_t|]);
    (ll_add, (L.pointer_type ll_node), [|L.pointer_type ll_node;string_t;i32_t|]);
    (ll_append, (L.pointer_type ll_node), [|L.pointer_type ll_node;L.pointer_type
        ll_node|]);
    (ll_next, (L.pointer_type ll_node), [|L.pointer_type ll_node|]);
    (ll_mem, (i32_t), [|L.pointer_type ll_node;string_t;i1_t|]);
    (ll_get, (string_t), [|L.pointer_type ll_node; i32_t|]);
    (ll_remove, (L.pointer_type ll_node), [|L.pointer_type ll_node;i32_t|]);
    (ll_print, i32_t, [|L.pointer_type ll_node|]);
    (ll_size, i32_t, [|L.pointer_type ll_node|]);
] in
let ll_funcs = List.fold_left declare_funcs StringMap.empty ll_defs in
let ll_add_func = StringMap.find ll_add ll_funcs in
let ll_append_func = StringMap.find ll_append ll_funcs in
let ll_next_func = StringMap.find ll_next ll_funcs in
let ll_mem_func = StringMap.find ll_mem ll_funcs in
let ll_get_func = StringMap.find ll_get ll_funcs in
let ll_remove_func = StringMap.find ll_remove ll_funcs in
let ll_print_func = StringMap.find ll_print ll_funcs in
let ll_size_func = StringMap.find ll_size ll_funcs in

(* hash table functions *)
let ht_create  = "ht_create" in
let ht_add     = "ht_add" in
let ht_mem     = "ht_mem" in
let ht_get     = "ht_get" in
let ht_remove  = "ht_remove" in
let ht_print   = "ht_print" in
let ht_size    = "ht_size" in
let ht_keys    = "ht_keys" in
```

```
let ht_keys_list = "ht_keys_list" in
let ht_defs = [
    (ht_create, (L.pointer_type ht_t), [|i32_t; i1_t|]);
    (ht_mem, i1_t, [|(L.pointer_type ht_t); string_t|]);
    (ht_get, string_t, [|(L.pointer_type ht_t); string_t|]);
    (ht_remove, (L.pointer_type ht_t), [|(L.pointer_type ht_t); string_t|]);
    (ht_add, (L.pointer_type ht_t), [|(L.pointer_type ht_t); string_t; string_t|]);
    (ht_print, i32_t, [|(L.pointer_type ht_t)|]);
    (ht_size, i32_t, [|L.pointer_type ht_t|]);
    (ht_keys, (L.pointer_type string_t), [|L.pointer_type ht_t|]);
    (ht_keys_list, (L.pointer_type ll_node), [|L.pointer_type ht_t|]);
] in
let ht_funcs = List.fold_left declare_funcs StringMap.empty ht_defs in
let ht_create_func = StringMap.find ht_create ht_funcs in
let ht_add_func = StringMap.find ht_add ht_funcs in
let ht_mem_func = StringMap.find ht_mem ht_funcs in
let ht_get_func = StringMap.find ht_get ht_funcs in
let ht_remove_func = StringMap.find ht_remove ht_funcs in
let ht_print_func = StringMap.find ht_print ht_funcs in
let ht_size_func = StringMap.find ht_size ht_funcs in
let ht_keys_func = StringMap.find ht_keys ht_funcs in
let ht_keys_list_func = StringMap.find ht_keys_list ht_funcs in

(* regex functions *)
let re_create_func : L.llvalue = declare_func ("re_create", L.pointer_type regex_t, [|
    string_t|]) false in
let re_match_func : L.llvalue = declare_func ("re_match", i1_t, [|L.pointer_type
    regex_t;string_t|]) false in
let re_sub_func : L.llvalue = declare_func ("re_sub", string_t, [|L.pointer_type
    regex_t;string_t;string_t;i32_t|]) false in

(* end stdlib functions *)

let rec expr parent_func builder (e : sx) =
    let assc_typ_of_typlist typlist =
        let t = List.hd typlist in
        (match t with
            A.UserTyp(ut) -> snd (StringMap.find ut env.tsym)
            | _ -> t)
    in

    let make_safe_malloc malloc_call (t : L.lltype) (builder : L.llbuilder) (func : L.
        llvalue) =
        let exit_bb = L.append_block context "endprog" func in
        let exit_builder = L.builder_at_end context exit_bb in
        ignore(L.build_call exit_func [|one|] "" exit_builder);
        ignore(L.build_unreachable exit_builder);

        let cont_bb = L.append_block context "contprog" func in

        let addr = malloc_call builder in
        let cond = L.build_icmp L.Icmp.Eq addr (L.const_null (L.pointer_type (t))) "
            mallocnull" builder in
        ignore(L.build_cond_br cond exit_bb cont_bb builder);

        let builder = L.builder_at_end context cont_bb in
        let c_addr = L.build_bitcast addr (L.pointer_type i8_t) "caddr" builder in
        ignore(L.build_call add_malloc_addr_func [|c_addr|] "" builder);

        (builder, addr)
    in

    let make_addr (e : L.llvalue) (t : L.lltype) (malloc : bool) (builder : L.llbuilder
```

```
 ) (func : L.llvalue) =
    let (builder, addr) = if malloc then (
        make_safe_malloc (L.build_malloc t "") t builder func
    ) else (builder, (L.build_alloca t "" builder))
    in
    ignore(L.build_store e addr builder);
    (builder, addr)
in

let make_func f_name formals ret_typ =
    let formals_arr = Array.of_list (List.map (fun (t,_) -> t) formals) in
    let func_typ = L.function_type ret_typ formals_arr in
    let func = L.define_function f_name func_typ the_module in

    let function_builder = L.builder_at_end context (L.entry_block func) in
    let params = Array.to_list (L.params func) in
    let func_locals = Hashtbl.create (List.length formals) in
    let make_param p (t,n) =
        L.set_value_name n p;
        let addr = L.build_alloca t "" function_builder in
        ignore(L.build_store p addr function_builder);
        Hashtbl.add func_locals n addr;
    in
    List.iter2 make_param params formals;
    Hashtbl.add func_params_tbl f_name func_locals;
    (func, function_builder)
in

let add_parent_vars_to_formals formals =
    let last : string ref = ref "" in
    let get_parent_vars k (v, v_parent_func) a =
        (* only get the most recently added k *)
        if k = !last then a
        (* don't add vars that share name with a formal param *)
        else (match (List.find_opt (fun (_,n) -> k = n) formals) with
            Some (_,_) -> a
            | None -> (match L.classify_value v with
                L.ValueKind.Instruction(_) ->
                    if v_parent_func = parent_func then (
                        let t = L.type_of v in
                        last := k;
                        (t,k)::a
                    )
                    else a
                | _ -> a
            )
        )
    in
    let parent_vars = Hashtbl.fold get_parent_vars var_tbl [] in
    formals @ parent_vars
in

let add_parent_vars_to_actuals params actuals =
    let actuals_len = List.length actuals in
    let get_parent_vars (parent_vars, i) param =
        (* skip the vars that are part of normal function declaration *)
        if i < actuals_len then (parent_vars, i+1)
        else (
            let (v, _) = Hashtbl.find var_tbl (L.value_name param) in
            (v::parent_vars, i+1)
        )
    in
    let (parent_vars, _) = Array.fold_left get_parent_vars ([], 0) params in
```

```
    actuals @ (List.rev parent_vars)
in

let init_params func f_name builder =
    let init_param i param =
        let param_name = L.value_name param in
        let out = try Hashtbl.find (Hashtbl.find func_params_tbl f_name) param_name
            with Not_found -> raise (Failure ("Param not found: " ^ param_name))
        in
        let out_load = L.build_load out ("param" ^ string_of_int i) builder in
        Hashtbl.add var_tbl param_name (out_load, func);
    in
    Array.iteri init_param (L.params func);
in

let cleanup_func_vars func =
    let find_funlit_vars k (_, v_parent_func) to_remove =
        if v_parent_func = func then
            k::to_remove
        else to_remove
    in
    let to_remove = Hashtbl.fold find_funlit_vars var_tbl [] in
    List.iter (fun k -> Hashtbl.remove var_tbl k) to_remove;
in

let make_ladd_func (f_name : string) (addr : L.llvalue) (e' : L.llvalue) =
    let addr_typ = L.type_of addr in
    let ltyp = L.type_of e' in
    let (func, function_builder) = make_func f_name [(addr_typ,"#l");(ltyp,"#e");
        i32_t,"#n"] addr_typ in

    init_params func f_name function_builder;

    let (function_builder, addr) = expr func function_builder (SId("#l")) in
    let (function_builder, e') = expr func function_builder (SId("#e")) in
    let (function_builder, n') = expr func function_builder (SId("#n")) in

    let addr_head = L.build_in_bounds_gep addr [|zero;one|] "listhead"
        function_builder in
    let head_node = L.build_load addr_head "headnode" function_builder in

    let (function_builder, data) = make_addr e' ltyp true function_builder func in
    let c_data = L.build_bitcast data string_t "cdata" function_builder in

    let head_node' = L.build_call ll_add_func [|head_node;c_data;n'|] ""
        function_builder in
    ignore(L.build_store head_node' addr_head function_builder);
    ignore(L.build_ret addr function_builder);

    cleanup_func_vars func;

    func
in

let make_dadd_func (f_name : string) (addr : L.llvalue) (k' : L.llvalue) (v' : L.
    llvalue) =
    let addr_typ = L.type_of addr in
    let k_typ = L.type_of k' in
    let v_typ = L.type_of v' in
    let (func, function_builder) = make_func f_name [(addr_typ,"#d");(k_typ,"#k");(
        v_typ,"#v")] addr_typ in

    init_params func f_name function_builder;
```

```
    let (function_builder, addr) = expr func function_builder (SId("#d")) in
    let (function_builder, k') = expr func function_builder (SId("#k")) in
    let (function_builder, v') = expr func function_builder (SId("#v")) in

    let addr_t1 = L.build_in_bounds_gep addr [|zero;zero|] "dictt1"
        function_builder in
    let addr_t2 = L.build_in_bounds_gep addr [|zero;one|] "dictt2" function_builder
         in
    let addr_ht = L.build_in_bounds_gep addr [|zero;two|] "dictht" function_builder
         in
    let ltyp1 = L.element_type (L.element_type (L.type_of addr_t1)) in
    let ltyp2 = L.element_type (L.element_type (L.type_of addr_t2)) in

    let (function_builder, k_data) = make_addr k' ltyp1 false function_builder func
         in
    let (function_builder, v_data) = make_addr v' ltyp2 false function_builder func
         in
    let c_k_data = L.build_bitcast k_data string_t "ckdata" function_builder in
    let c_v_data = L.build_bitcast v_data string_t "cvdata" function_builder in
    let ht = L.build_load addr_ht "ht" function_builder in
    let ht' = L.build_call ht_add_func [|ht;c_k_data;c_v_data|] "ht_"
        function_builder in
    ignore(L.build_store ht' addr_ht function_builder);
    ignore(L.build_ret addr function_builder);

    cleanup_func_vars func;

    func
in

let make_lget_func (f_name : string) (addr : L.llvalue) (builder: L.llbuilder) =
    let addr_typ = L.type_of addr in
    let addr_t = L.build_in_bounds_gep addr [|zero;zero|] "listt" builder in
    let t = L.element_type (L.element_type (L.type_of addr_t)) in
    let (func, function_builder) = make_func f_name [(addr_typ,"#l");(i32_t,"#n")]
        t in

    init_params func f_name function_builder;

    let (function_builder, addr) = expr func function_builder (SId("#l")) in
    let (function_builder, n') = expr func function_builder (SId("#n")) in

    let addr_t = L.build_in_bounds_gep addr [|zero;zero|] "listt" function_builder
        in
    let addr_head = L.build_in_bounds_gep addr [|zero;one|] "listhead"
        function_builder in
    let ltyp_ptr = L.build_load addr_t "t*" function_builder in

    let head_node = L.build_load addr_head "headnode" function_builder in
    let c_data = L.build_call ll_get_func [|head_node;n'|] "cdata" function_builder
         in
    let data = L.build_bitcast c_data (L.type_of ltyp_ptr) "data" function_builder
        in
    let data_load = L.build_load data "dataload" function_builder in
    ignore(L.build_ret data_load function_builder);

    cleanup_func_vars func;

    func
in

let make_lfold_func (f_name : string) (arg_func : L.llvalue) (a' : L.llvalue) (addr
```

```
  : L.llvalue) =

let arg_func_typ = L.type_of arg_func in
let accum_typ = L.type_of a' in
let addr_typ = L.type_of addr in

let formals = [(arg_func_typ,"#f");(accum_typ,"#a");(addr_typ,"#l")] in
let (func, function_builder) = make_func f_name (add_parent_vars_to_formals
    formals) accum_typ in

init_params func f_name function_builder;

let arg_func_params = L.params arg_func in

let (function_builder, arg_func) = expr func function_builder (SId("#f")) in
let (function_builder, a') = expr func function_builder (SId("#a")) in
let (function_builder, addr) = expr func function_builder (SId("#l")) in

let i_addr = L.build_alloca i32_t "iaddr" function_builder in
ignore(L.build_store zero i_addr function_builder);
let accum_t = L.type_of a' in
let (function_builder, accum_addr) = make_safe_malloc (L.build_malloc accum_t "
    accum") accum_t function_builder func in
ignore(L.build_store a' accum_addr function_builder);

let addr_t = L.build_in_bounds_gep addr [|zero;zero|] "listltyp"
    function_builder in
let addr_head = L.build_in_bounds_gep addr [|zero;one|] "listhead"
    function_builder in
let head_node = L.build_load addr_head "headnode" function_builder in
let curr_node_t = L.pointer_type ll_node in
let (function_builder, curr_node) = make_safe_malloc (L.build_malloc
    curr_node_t "currnode") curr_node_t function_builder func in
ignore(L.build_store head_node curr_node function_builder);

let ltyp_ptr = L.element_type (L.type_of addr_t) in
let len = L.build_call ll_size_func [|head_node|] "len" function_builder in

let cond_bb = L.append_block context "cond" func in
let cond_builder = L.builder_at_end context cond_bb in
let i = L.build_load i_addr "i" cond_builder in
let cond = L.build_icmp L.Icmp.Slt i len "lessthan" cond_builder in

let body_bb = L.append_block context "foldbody" func in
let body_builder = L.builder_at_end context body_bb in

let node = L.build_load curr_node "node" body_builder in
let c_data = L.build_call ll_get_func [|node;zero|] "cdata" body_builder in
let data = L.build_bitcast c_data ltyp_ptr "data" body_builder in
let data_load = L.build_load data "dataload" body_builder in

let a' = L.build_load accum_addr "accumload" body_builder in

let actuals = add_parent_vars_to_actuals arg_func_params [a';data_load] in
let actuals_arr = Array.of_list actuals in
let a' = L.build_call arg_func actuals_arr "accumresult" body_builder in

ignore(L.build_store a' accum_addr body_builder);

let i = L.build_add i one "i" body_builder in
ignore(L.build_store i i_addr body_builder);

let next_node = L.build_call ll_next_func [|node|] "nextnode" body_builder in
```

```
    ignore(L.build_store next_node curr_node body_builder);

    let merge_bb = L.append_block context "merge" func in

    ignore(L.build_br cond_bb function_builder);
    ignore(L.build_br cond_bb body_builder);
    ignore(L.build_cond_br cond body_bb merge_bb cond_builder);

    let function_builder = L.builder_at_end context merge_bb in
    let accum_final = L.build_load accum_addr "lfoldaccum" function_builder in

    ignore(L.build_ret accum_final function_builder);

    cleanup_func_vars func;

    func
in

let make_dfold_func (f_name : string) (arg_func : L.llvalue) (a' : L.llvalue) (addr
     : L.llvalue) =
    let arg_func_typ = L.type_of arg_func in
    let accum_typ = L.type_of a' in
    let addr_typ = L.type_of addr in

    let formals = [(arg_func_typ,"#f");(accum_typ,"#a");(addr_typ,"#d")] in
    let (func, function_builder) = make_func f_name (add_parent_vars_to_formals
        formals) accum_typ in

    init_params func f_name function_builder;

    let arg_func_params = L.params arg_func in

    let (function_builder, arg_func) = expr func function_builder (SId("#f")) in
    let (function_builder, a') = expr func function_builder (SId("#a")) in
    let (function_builder, addr) = expr func function_builder (SId("#d")) in

    let i_addr = L.build_alloca i32_t "iaddr" function_builder in
    ignore(L.build_store zero i_addr function_builder);
    let (function_builder, accum_addr) = make_safe_malloc (L.build_malloc accum_typ
         "accum") accum_typ function_builder func in
    ignore(L.build_store a' accum_addr function_builder);

    let dictt1_addr = L.build_in_bounds_gep addr [|zero;zero|] "dictt1addr"
        function_builder in
    let dictt2_addr = L.build_in_bounds_gep addr [|zero;one|] "dictt2addr"
        function_builder in
    let t1_ptr = L.element_type (L.type_of dictt1_addr) in
    let t2_ptr = L.element_type (L.type_of dictt2_addr) in
    let dictht_addr = L.build_in_bounds_gep addr [|zero;two|] "dicthtaddr"
        function_builder in
    let ht = L.build_load dictht_addr "ht" function_builder in
    let keys = L.build_call ht_keys_func [|ht|] "keys" function_builder in
    let size = L.build_call ht_size_func [|ht|] "size" function_builder in

    let cond_bb = L.append_block context "cond" func in
    let cond_builder = L.builder_at_end context cond_bb in
    let i = L.build_load i_addr "i" cond_builder in
    let cond = L.build_icmp L.Icmp.Slt i size "lessthan" cond_builder in

    let body_bb = L.append_block context "foldbody" func in
    let body_builder = L.builder_at_end context body_bb in

    let curr_key_gep = L.build_in_bounds_gep keys [|i|] "currkeygep" body_builder
```

```
        in
    let curr_key_c = L.build_load curr_key_gep "currkeyc" body_builder in
    let curr_value_c = L.build_call ht_get_func [|ht;curr_key_c|] "currvaluec"
        body_builder in
    let curr_key_addr = L.build_bitcast curr_key_c t1_ptr "currkeyaddr"
        body_builder in
    let curr_value_addr = L.build_bitcast curr_value_c t2_ptr "currvalueaddr"
        body_builder in
    let curr_key = L.build_load curr_key_addr "currkey" body_builder in
    let curr_value = L.build_load curr_value_addr "currval" body_builder in

    let a' = L.build_load accum_addr "accumload" body_builder in
    let actuals = add_parent_vars_to_actuals arg_func_params [a';curr_key;
        curr_value] in
    let actuals_arr = Array.of_list actuals in
    let a' = L.build_call arg_func actuals_arr "accumresult" body_builder in
    ignore(L.build_store a' accum_addr body_builder);

    let i = L.build_add i one "i" body_builder in
    ignore(L.build_store i i_addr body_builder);

    let merge_bb = L.append_block context "merge" func in

    ignore(L.build_br cond_bb function_builder);
    ignore(L.build_br cond_bb body_builder);
    ignore(L.build_cond_br cond body_bb merge_bb cond_builder);

    let function_builder = L.builder_at_end context merge_bb in
    let accum_final = L.build_load accum_addr "dfoldaccum" function_builder in

    ignore(L.build_ret accum_final function_builder);

    cleanup_func_vars func;

    func
in

match e with

SIntLit(i) -> let out = L.const_int i32_t i in
    (builder, out)

| SFloatLit(f) -> let out = L.const_float_of_string float_t f in
    (builder, out)

| SBoolLit(b) -> let out = L.const_int i1_t (if b then 1 else 0) in
    (builder, out)

| SStrLit(s) ->
    let prefix = "s" in
    let len = String.length s + 1 in
    let (builder, addr) = make_safe_malloc (L.build_array_malloc i8_t (L.const_int
        i32_t len) "string") i8_t builder parent_func in
    let store_char i c =
        let i' = string_of_int i in
        let c' = Char.code c in
        let addr_i = L.build_in_bounds_gep addr [|L.const_int i32_t i|] (prefix ^ i
            ') builder in
        ignore(L.build_store (L.const_int i8_t c') addr_i builder);
    in
    String.iteri store_char (s ^ "\x00");
    (builder, addr)
```

```
| SReLit(r) ->
   let (builder, addr) = expr parent_func builder (SStrLit(r)) in
   let regex = L.build_call re_create_func [|addr|] "regex" builder in
   (builder,regex)
| SListLit(t, l) ->
   let list_t = L.element_type (ltyp_of_typ (A.List(t))) in
   let ltyp = ltyp_of_typ t in

   let (builder, addr) = make_safe_malloc (L.build_malloc list_t "list") list_t
       builder parent_func in
   let addr_ltyp = L.build_in_bounds_gep addr [|zero;zero|] "listltyp" builder in
   let addr_head = L.build_in_bounds_gep addr [|zero;one|] "listhead" builder in

   let (builder, null_t) = make_safe_malloc (L.build_malloc ltyp "null") ltyp
       builder parent_func in
   ignore(L.build_store (L.const_null ltyp) null_t builder);
   ignore(L.build_store null_t addr_ltyp builder);

   let l' = List.rev (List.map (fun e -> snd e) l) in
   let (builder, addr) = (match l' with

       last::the_rest -> (
           let (builder, last') = expr parent_func builder last in

           let ladd_f_name = "ladd" ^ (str_of_ltyp (L.type_of addr)) in
           let ladd_func = (try Hashtbl.find func_tbl ladd_f_name
               with Not_found -> (
                   let func = make_ladd_func ladd_f_name addr last' in
                   Hashtbl.add func_tbl ladd_f_name func;
                   func
               )
           ) in
           let addr = L.build_call ladd_func [|addr;last';zero|] "listlitadd"
               builder in

           let add_node (builder, addr) e =
               let (builder, e') = expr parent_func builder e in
               let addr = L.build_call ladd_func [|addr;e';zero|] "listlitadd"
                   builder in
               (builder, addr)
           in
           List.fold_left add_node (builder, addr) the_rest
       )
       | [] -> (
           ignore(L.build_store (L.const_null (L.pointer_type ll_node)) addr_head
               builder);
           (builder, addr)
       )
   ) in
   (builder, addr)

| SDictLit(t1, t2, d) ->
   let dict_t = L.element_type (ltyp_of_typ (A.Dict(t1,t2))) in
   let ltyp1 = ltyp_of_typ t1 in
   let ltyp2 = ltyp_of_typ t2 in
   let (builder, addr) = make_safe_malloc (L.build_malloc dict_t "dict") dict_t
       builder parent_func in

   (* dummy values so we know the actual types of the table *)
   let (builder, null_t1) = make_safe_malloc (L.build_malloc ltyp1 "nullt1") ltyp1
        builder parent_func in
   let (builder, null_t2) = make_safe_malloc (L.build_malloc ltyp2 "nullt2") ltyp2
        builder parent_func in
```

```
    let addr_t1 = L.build_in_bounds_gep addr [|zero;zero|] "dictt1" builder in
    let addr_t2 = L.build_in_bounds_gep addr [|zero; one|] "dictt2" builder in
    ignore(L.build_store (L.const_null ltyp1) null_t1 builder);
    ignore(L.build_store (L.const_null ltyp2) null_t2 builder);
    ignore(L.build_store null_t1 addr_t1 builder);
    ignore(L.build_store null_t2 addr_t2 builder);

    (* create dict *)
    let t1_assc = assc_typ_of_typlist [t1] in
    let key_is_string = match t1_assc with A.String -> 1 | _ -> 0 in
    let ht = L.build_call ht_create_func [|L.const_int i32_t (List.length d);L.
        const_int i1_t key_is_string|] "tbl" builder in
    let addr_ht = L.build_in_bounds_gep addr [|zero;two|] "dictht" builder in
    ignore(L.build_store ht addr_ht builder);

    (* adding all the dict k:v pairs *)
    let d' = List.map (fun (se1, se2) -> (snd se1, snd se2)) d in
    let add_pair (builder, i) (k,v) =
        let i' = string_of_int i in
        let (builder, k') = expr parent_func builder k in
        let (builder, v') = expr parent_func builder v in
        let (builder, k_data) = make_addr k' ltyp1 true builder parent_func in
        let (builder, v_data) = make_addr v' ltyp2 true builder parent_func in
        let c_k_data = L.build_bitcast k_data string_t ("ckdata" ^ i') builder in
        let c_v_data = L.build_bitcast v_data string_t ("cvdata" ^ i') builder in
        ignore(L.build_call ht_add_func [|ht;c_k_data;c_v_data|] "" builder);
        (builder, i+1)
    in
    let (builder, _) = List.fold_left add_pair (builder, 0) d' in
    (builder, addr)

| STypComp((typlist,_), t) ->
    let out_addr = L.build_alloca i1_t "typcompout" builder in
    ignore(L.build_store fals out_addr builder);

    let merge_bb = L.append_block context "merge" parent_func in

    let true_bb = L.append_block context "typcomptrue" parent_func in
    let true_builder = L.builder_at_end context true_bb in
    ignore(L.build_store tru out_addr true_builder);
    ignore(L.build_br merge_bb true_builder);

    let rhs_name = match t with A.UserTyp(n) -> n | _ -> raise (Failure (
        internal_err ^ "rhs_name")) in
    let rhs_val = match (L.lookup_global rhs_name the_module) with Some g -> g |
        None -> raise (Failure (internal_err ^ "rhs_val")) in
    let make_typ_comp (blocks, i) typ = (
        match typ with
            A.UserTyp(lhs_name) -> (
                let i' = string_of_int i in
                let lhs_val = match (L.lookup_global lhs_name the_module) with Some
                    g -> g | None -> raise (Failure (internal_err ^ "lhs_val")) in
                let cond_bb = L.append_block context ("typcomp" ^ i') parent_func in
                let cond_builder = L.builder_at_end context cond_bb in
                let comp_result = L.build_icmp L.Icmp.Eq rhs_val lhs_val ("
                    typcompresult" ^ i') cond_builder in
                ignore(L.build_cond_br comp_result true_bb (List.hd blocks)
                    cond_builder);
                (cond_bb::blocks, i+1)
            )
            (* do nothing if not a usertyp *)
            | _ -> (blocks, i)
        )
```

```
    in
    let (blocks, _) = List.fold_left make_typ_comp ([merge_bb], 0) typlist in

    ignore(L.build_br (List.hd blocks) builder);

    let builder = L.builder_at_end context merge_bb in
    let out_load = L.build_load out_addr "typcompout" builder in
    (builder, out_load)

  | SBinop((typlist,e1), o, (_,e2)) ->
    let t = assc_typ_of_typlist typlist in
    let (builder, e1') = expr parent_func builder e1 in
    let (builder, e2') = expr parent_func builder e2 in
    let (builder, out) = (match t with

        A.Int -> (builder, (match o with
              A.Add    -> L.build_add
            | A.Sub    -> L.build_sub
            | A.Mult   -> L.build_mul
            | A.Div    -> L.build_sdiv
            | A.Mod    -> L.build_srem
            | A.Equal  -> L.build_icmp L.Icmp.Eq
            | A.Greater -> L.build_icmp L.Icmp.Sgt
            | A.Less   -> L.build_icmp L.Icmp.Slt
            | _ -> raise (Failure (internal_err ^ "binop int"))
        ) e1' e2' "out" builder)

        | A.Float -> (builder, (match o with
              A.Add    -> L.build_fadd
            | A.Sub    -> L.build_fsub
            | A.Mult   -> L.build_fmul
            | A.Div    -> L.build_fdiv
            | A.Mod    -> L.build_frem
            | A.Equal  -> L.build_fcmp L.Fcmp.Oeq
            | A.Greater -> L.build_fcmp L.Fcmp.Ogt
            | A.Less   -> L.build_fcmp L.Fcmp.Olt
            | _ -> raise (Failure (internal_err ^ "binop float"))
        ) e1' e2' "out" builder)

        | A.Bool -> (builder, (match o with
              A.And -> L.build_and
            | A.Or -> L.build_or
            | A.Equal  -> L.build_icmp L.Icmp.Eq
            | A.Greater -> L.build_icmp L.Icmp.Sgt
            | A.Less   -> L.build_icmp L.Icmp.Slt
            | t -> raise (Failure (internal_err ^ "binop bool " ^ (A.string_of_binop
                t)))
        ) e1' e2' "out" builder)

        | A.String -> (match o with
            A.Concat ->
                let len1 = L.build_call strlen_func [|e1'|] "e1len" builder in
                let len2 = L.build_call strlen_func [|e2'|] "e2len" builder in
                let len = L.build_add len1 len2 "len" builder in
                let len = L.build_add len one "len" builder in
                let (builder, out_addr) = make_safe_malloc (L.build_array_malloc
                    i8_t len "out") i8_t builder parent_func in
                let out_addr = L.build_call strcpy_func [|out_addr; e1'|] "cpy"
                    builder in
                let out_addr = L.build_call strcat_func [|out_addr; e2'|] "cat"
                    builder in
                (builder, out_addr)
            | _ -> raise (Failure (internal_err ^ "binop string"))
```

```
        )

        | A.List(t) -> (match o with
            A.Concat ->
                let (builder, addr) = expr parent_func builder (SListLit(t,[])) in
                let addr_head = L.build_in_bounds_gep addr [|zero;one|] "listhead"
                    builder in
                let e1_addr_head = L.build_in_bounds_gep e1' [|zero;one|] "
                    e1listhead" builder in
                let e2_addr_head = L.build_in_bounds_gep e2' [|zero;one|] "
                    e2listhead" builder in
                let e1_head_node = L.build_load e1_addr_head "e1headnode" builder in
                let e2_head_node = L.build_load e2_addr_head "e2headnode" builder in
                let head_node = L.build_call ll_append_func [|e1_head_node;
                    e2_head_node|] "concatheadnode" builder in
                ignore(L.build_store head_node addr_head builder);
                (builder, addr)

            | _ -> raise (Failure (internal_err ^ "binop list"))
        )
        | _ -> raise (Failure (internal_err ^ "binop something else"))
    )
    in
    (builder, out)

| SUnop(o, (typlist,e)) ->
    let (builder, e') = expr parent_func builder e in
    let t =
        let t = List.hd typlist in
        (match t with
            A.UserTyp(ut) -> snd (StringMap.find ut env.tsym)
            | _ -> t)
    in
    let out = (match o with
        A.Not -> L.build_not e' "not" builder
        | A.Neg -> (
            match t with
            A.Int -> L.build_neg e' "neg" builder
            | A.Float -> L.build_fneg e' "neg" builder
            | _ -> raise (Failure (internal_err ^ "unop neg"))
        )
    ) in
    (builder, out)

| SFunCall((t,SId("ladd")), [(t_l,l); (t_e,e)]) ->
    expr parent_func builder (SFunCall((t,SId("ladd")), [(t_l,l); (t_e,e); ([A.Int
        ],SIntLit(0))]))

| SFunCall((_,SId("ladd")), [(_,l); (_,e); (_,n)]) ->
    let (builder, addr) = expr parent_func builder l in
    let (builder, e') = expr parent_func builder e in
    let (builder, n') = expr parent_func builder n in
    let f_name = "ladd" ^ (str_of_ltyp (L.type_of addr)) in
    let func = (try Hashtbl.find func_tbl f_name
        with Not_found -> (
            let func = make_ladd_func f_name addr e' in
            Hashtbl.add func_tbl f_name func;
            func
        )
    ) in
    let addr' = L.build_call func [|addr;e';n'|] "ladd" builder in
    (builder, addr')
```

```
| SFunCall((_,SId("dadd")), [(_, dict); (_,k); (_,v)]) ->
   let (builder, addr) = expr parent_func builder dict in
   let (builder, k') = expr parent_func builder k in
   let (builder, v') = expr parent_func builder v in
   let f_name = "dadd" ^ (str_of_ltyp (L.type_of addr)) in
   let func = (try Hashtbl.find func_tbl f_name
      with Not_found -> (
         let func = make_dadd_func f_name addr k' v' in
         Hashtbl.add func_tbl f_name func;
         func
      )
   ) in

   let addr = L.build_call func [|addr;k';v'|] "dadd" builder in
   (builder, addr)

| SFunCall((_,SId("lmem")), [(_, l); (_,e)]) ->
   let (builder, addr) = expr parent_func builder l in
   let (builder, e') = expr parent_func builder e in
   let addr_typ = L.type_of addr in
   let f_name = "lmem" ^ (str_of_ltyp addr_typ) in
   let func = (try Hashtbl.find func_tbl f_name
      with Not_found -> (
         let addr_t = L.build_in_bounds_gep addr [|zero;zero|] "listt" builder in
         let t = L.element_type (L.element_type (L.type_of addr_t)) in
         let (func, function_builder) = make_func f_name [(addr_typ,"#l");(t,"#e
             ")] i32_t in

         init_params func f_name function_builder;

         let (function_builder, addr) = expr func function_builder (SId("#l")) in
         let (function_builder, e') = expr func function_builder (SId("#e")) in

         let addr_head = L.build_in_bounds_gep addr [|zero;one|] "listhead"
             function_builder in
         let head_node = L.build_load addr_head "headnode" function_builder in

         let (function_builder, data) = make_addr e' t false function_builder
             func in
         let c_data = L.build_bitcast data string_t "cdata" function_builder in

         let n = if t = string_t then
            L.build_call ll_mem_func [|head_node;c_data;tru|] "cdata"
                function_builder
         else
            L.build_call ll_mem_func [|head_node;c_data;fals|] "cdata"
                function_builder
         in
         ignore(L.build_ret n function_builder);

         cleanup_func_vars func;
         Hashtbl.add func_tbl f_name func;

         func
      )
   ) in

   let n = L.build_call func [|addr;e'|] "lmem" builder in
   (builder, n)

| SFunCall((_,SId("dmem")), [(_, dict); (_,k)]) ->
   let (builder, addr) = expr parent_func builder dict in
   let (builder, k') = expr parent_func builder k in
```

```
let addr_typ = L.type_of addr in
let f_name = "dmem" ^ (str_of_ltyp addr_typ) in
let func = (try Hashtbl.find func_tbl f_name
    with Not_found -> (
        let k_typ = L.type_of k' in
        let (func, function_builder) = make_func f_name [(addr_typ,"#d");(k_typ
            ,"#k")] i1_t in

        init_params func f_name function_builder;

        let (function_builder, addr) = expr func function_builder (SId("#d")) in
        let (function_builder, k') = expr func function_builder (SId("#k")) in

        let addr_t1 = L.build_in_bounds_gep addr [|zero;zero|] "dictt1"
            function_builder in
        let addr_ht = L.build_in_bounds_gep addr [|zero;two|] "dictht"
            function_builder in
        let ht = L.build_load addr_ht "ht" function_builder in
        let ltyp1 = L.element_type (L.element_type (L.type_of addr_t1)) in

        let (function_builder, k_data) = make_addr k' ltyp1 false
            function_builder func in
        let c_k_data = L.build_bitcast k_data string_t "ckdata" function_builder
             in
        let is_mem = L.build_call ht_mem_func [|ht;c_k_data|] "dmem"
            function_builder in
        ignore(L.build_ret is_mem function_builder);

        cleanup_func_vars func;
        Hashtbl.add func_tbl f_name func;

        func
    )
) in

let is_mem = L.build_call func [|addr;k'|] "dmem" builder in
(builder, is_mem)



| SFunCall((_,SId("lget")), [(_, l); (_,n)]) ->
    let (builder, addr) = expr parent_func builder l in
    let (builder, n') = expr parent_func builder n in
    let addr_type = L.type_of addr in
    let f_name = "lget" ^ (str_of_ltyp addr_type) in
    let func = (try Hashtbl.find func_tbl f_name
        with Not_found -> (
            let func = make_lget_func f_name addr builder in
            Hashtbl.add func_tbl f_name func;
            func
        )
    ) in

    let data_load = L.build_call func [|addr;n'|] "lget" builder in
    (builder, data_load)

| SFunCall((_,SId("dget")), [(_, dict); (_,k)]) ->
    let (builder, addr) = expr parent_func builder dict in
    let (builder, k') = expr parent_func builder k in
    let addr_typ = L.type_of addr in
    let f_name = "dget" ^ (str_of_ltyp addr_typ) in

    let func = (try Hashtbl.find func_tbl f_name
```

```
        with Not_found -> (
            let k_typ = L.type_of k' in
            let addr_t2 = L.build_in_bounds_gep addr [|zero;one|] "dictt2" builder
                in
            let v_typ = L.element_type (L.element_type (L.type_of addr_t2)) in
            let (func, function_builder) = make_func f_name [(addr_typ,"#d");(k_typ
                ,"#k")] v_typ in

            init_params func f_name function_builder;

            let (function_builder, addr) = expr func function_builder (SId("#d")) in
            let (function_builder, k') = expr func function_builder (SId("#k")) in

            let addr_t1 = L.build_in_bounds_gep addr [|zero;zero|] "dictt1"
                function_builder in
            let addr_t2 = L.build_in_bounds_gep addr [|zero;one|] "dictt2"
                function_builder in
            let addr_ht = L.build_in_bounds_gep addr [|zero;two|] "dictht"
                function_builder in
            let ltyp1 = L.element_type (L.element_type (L.type_of addr_t1)) in
            let (function_builder, k_data) = make_addr k' ltyp1 false
                function_builder func in
            let c_k_data = L.build_bitcast k_data string_t "ckdata" function_builder
                 in
            let ht = L.build_load addr_ht "ht" function_builder in
            let c_v_data = L.build_call ht_get_func [|ht;c_k_data|] "cvdata"
                function_builder in
            let v_data = L.build_bitcast c_v_data (L.element_type (L.type_of addr_t2
                )) "vdata" function_builder in
            let v_data_load = L.build_load v_data "vdataload" function_builder in
            ignore(L.build_ret v_data_load function_builder);

            cleanup_func_vars func;
            Hashtbl.add func_tbl f_name func;

            func
        )
    ) in

    let v_data_load = L.build_call func [|addr;k'|] "dget" builder in
    (builder, v_data_load)

| SFunCall((t,SId("lremove")), [(t_l, l)]) ->
    expr parent_func builder (SFunCall((t,SId("lremove")), [(t_l, l); ([A.Int],
        SIntLit(0))]))

| SFunCall((_,SId("lremove")), [(_, l); (_,n)]) ->
    let (builder, addr) = expr parent_func builder l in
    let (builder, n') = expr parent_func builder n in
    let addr_typ = L.type_of addr in
    let f_name = "lremove" ^ (str_of_ltyp addr_typ) in

    let func = (try Hashtbl.find func_tbl f_name
        with Not_found -> (
            let (func, function_builder) = make_func f_name [(addr_typ,"#l");(i32_t
                ,"#n")] addr_typ in

            init_params func f_name function_builder;

            let (function_builder, addr) = expr func function_builder (SId("#l")) in
            let (function_builder, n') = expr func function_builder (SId("#n")) in

            let addr_head = L.build_in_bounds_gep addr [|zero;one|] "listhead"
```

```
            function_builder in
        let head_node = L.build_load addr_head "headnode" function_builder in
        let head_node' = L.build_call ll_remove_func [|head_node;n'|] "headnode_
            " function_builder in
        ignore(L.build_store head_node' addr_head function_builder);
        ignore(L.build_ret addr function_builder);

        cleanup_func_vars func;
        Hashtbl.add func_tbl f_name func;

        func
      )
    ) in

    let addr' = L.build_call func [|addr;n'|] "lremove" builder in
    (builder, addr')

| SFunCall((_,SId("dremove")), [(_, dict); (_,k)]) ->
    let (builder, addr) = expr parent_func builder dict in
    let (builder, k') = expr parent_func builder k in
    let addr_typ = L.type_of addr in
    let f_name = "dremove" ^ (str_of_ltyp addr_typ) in
    let func = (try Hashtbl.find func_tbl f_name
      with Not_found -> (
        let k_typ = L.type_of k' in
        let (func, function_builder) = make_func f_name [(addr_typ,"#d");(k_typ
            ,"#k")] addr_typ in

        init_params func f_name function_builder;

        let (function_builder, addr) = expr func function_builder (SId("#d")) in
        let (function_builder, k') = expr func function_builder (SId("#k")) in

        let addr_t1 = L.build_in_bounds_gep addr [|zero;zero|] "dictt1"
            function_builder in
        let addr_ht = L.build_in_bounds_gep addr [|zero;two|] "dictht"
            function_builder in
        let ltyp1 = L.element_type (L.element_type (L.type_of addr_t1)) in
        let (function_builder, k_data) = make_addr k' ltyp1 false
            function_builder func in
        let c_k_data = L.build_bitcast k_data string_t "ckdata" function_builder
             in
        let ht = L.build_load addr_ht "ht" function_builder in
        let ht' = L.build_call ht_remove_func [|ht;c_k_data|] "ht_"
            function_builder in
        ignore(L.build_store ht' addr_ht function_builder);
        ignore(L.build_ret addr function_builder);

        cleanup_func_vars func;
        Hashtbl.add func_tbl f_name func;

        func
      )
    ) in

    let addr = L.build_call func [|addr;k'|] "dremove" builder in
    (builder, addr)

| SFunCall((_,SId("ssize")), [(_, s)]) ->
    let (builder, addr) = expr parent_func builder s in
    let size = L.build_call strlen_func [|addr|] "ssize" builder in
    (builder, size)
```

```
| SFunCall((_,SId("lsize")), [(_, l)]) ->
   let (builder, addr) = expr parent_func builder l in
   let addr_head = L.build_in_bounds_gep addr [|zero;one|] "listhead" builder in
   let head_node = L.build_load addr_head "headnode" builder in
   let size = L.build_call ll_size_func [|head_node|] "lsize" builder in
   (builder, size)

| SFunCall((_,SId("dsize")), [(_,d)]) ->
   let (builder, addr) = expr parent_func builder d in
   let addr_ht = L.build_in_bounds_gep addr [|zero;two|] "dictht" builder in
   let ht = L.build_load addr_ht "ht" builder in
   let size = L.build_call ht_size_func [|ht|] "dsize" builder in
   (builder, size)

| SFunCall((_,SId("sprint")), [(_,e)]) ->
   let (builder, e') = expr parent_func builder e in
   let out = L.build_call printf_func [|str_format;e'|] "printf" builder in
   (builder, out)

| SFunCall((_,SId("lprint")), [(_,e)]) ->
   let (builder, addr) = expr parent_func builder e in
   let addr_head = L.build_in_bounds_gep addr [|zero;one|] "listhead" builder in
   let head_node = L.build_load addr_head "headnode" builder in
   let out = L.build_call ll_print_func [|head_node|] "printlist" builder in
   (builder, out)

| SFunCall((_,SId("dprint")), [(_,e)]) ->
   let (builder, addr) = expr parent_func builder e in
   let addr_ht = L.build_in_bounds_gep addr [|zero;two|] "dictht" builder in
   let ht = L.build_load addr_ht "ht" builder in
   let out = L.build_call ht_print_func [|ht|] "printdict" builder in
   (builder, out)

| SFunCall((_,SId("sfold")), [(_,f);(_,a);(_,s)]) ->
   let (builder, arg_func) = expr parent_func builder f in
   let (builder, a') = expr parent_func builder a in
   let (builder, addr) = expr parent_func builder s in
   let arg_func_typ = L.type_of arg_func in
   let f_name = "sfold" ^ (str_of_ltyp (L.element_type arg_func_typ)) in

   let func = (try Hashtbl.find func_tbl f_name
       with Not_found -> (

           let accum_typ = L.type_of a' in
           let formals = [(arg_func_typ,"#f");(accum_typ,"#a");(string_t,"#s")] in
           let (func, function_builder) = make_func f_name (
               add_parent_vars_to_formals formals) accum_typ in

           init_params func f_name function_builder;

           let arg_func_params = L.params arg_func in

           let (function_builder, arg_func) = expr func function_builder (SId("#f")
               ) in
           let (function_builder, a') = expr func function_builder (SId("#a")) in
           let (function_builder, addr) = expr func function_builder (SId("#s")) in

           let i_addr = L.build_alloca i32_t "iaddr" function_builder in
           ignore(L.build_store zero i_addr function_builder);
           let (function_builder, accum_addr) = make_safe_malloc (L.build_malloc
               accum_typ "accum") accum_typ function_builder func in
           ignore(L.build_store a' accum_addr function_builder);
           let len = L.build_call strlen_func [|addr|] "len" function_builder in
```

```
            let cond_bb = L.append_block context "cond" func in
            let cond_builder = L.builder_at_end context cond_bb in
            let i = L.build_load i_addr "i" cond_builder in
            let cond = L.build_icmp L.Icmp.Slt i len "lessthan" cond_builder in

            let body_bb = L.append_block context "foldbody" func in
            let body_builder = L.builder_at_end context body_bb in
            let (body_builder, c_str) = make_safe_malloc (L.build_array_malloc i8_t
                two "cstr") i8_t body_builder func in
            let c_char = L.build_in_bounds_gep c_str [|zero|] "cchar" body_builder
                in
            let c_null = L.build_in_bounds_gep c_str [|one|] "cnull" body_builder in
            ignore(L.build_store (L.const_int i8_t 0) c_null body_builder);
            let c_addr = L.build_in_bounds_gep addr [|i|] "caddr" body_builder in
            let c = L.build_load c_addr "c" body_builder in
            ignore(L.build_store c c_char body_builder);
            let a = L.build_load accum_addr "accumload" body_builder in

            let actuals = add_parent_vars_to_actuals arg_func_params [a;c_str] in
            let actuals_arr = Array.of_list actuals in
            let a = L.build_call arg_func actuals_arr "accumresult" body_builder in

            ignore(L.build_store a accum_addr body_builder);
            let i = L.build_add i one "i" body_builder in
            ignore(L.build_store i i_addr body_builder);

            let merge_bb = L.append_block context "merge" func in

            ignore(L.build_br cond_bb function_builder);
            ignore(L.build_br cond_bb body_builder);
            ignore(L.build_cond_br cond body_bb merge_bb cond_builder);

            let function_builder = L.builder_at_end context merge_bb in
            let accum_final = L.build_load accum_addr "sfoldaccum" function_builder
                in

            ignore(L.build_ret accum_final function_builder);

            cleanup_func_vars func;
            Hashtbl.add func_tbl f_name func;

            func
        )
    ) in

    let actuals = add_parent_vars_to_actuals (L.params func) [arg_func;a';addr] in
    let actuals_arr = Array.of_list actuals in
    let accum_final = L.build_call func actuals_arr "sfoldaccumfinal" builder in
    (builder, accum_final)

| SFunCall((_,SId("lfold")), [(_,f);(_,a);(_,l)]) ->
    let (builder, arg_func) = expr parent_func builder f in
    let (builder, a') = expr parent_func builder a in
    let (builder, addr) = expr parent_func builder l in
    let f_name_suf = (str_of_ltyp (L.type_of a')) ^ (str_of_ltyp (L.type_of addr))
        in
    let f_name = "lfold" ^ f_name_suf ^ (string_of_int (Array.length (L.params
        arg_func))) in

    let func = (try Hashtbl.find func_tbl f_name
        with Not_found -> (
            let func = make_lfold_func f_name arg_func a' addr in
```

```
            Hashtbl.add func_tbl f_name func;
            func
         )
      ) in

      let actuals = add_parent_vars_to_actuals (L.params func) [arg_func;a';addr] in
      let actuals_arr = Array.of_list actuals in
      let accum_final = L.build_call func actuals_arr "lfoldaccumfinal" builder in
      (builder, accum_final)

   | SFunCall((_,SId("dfold")), [(_,f);(_,a);(_,d)]) ->
      let (builder, arg_func) = expr parent_func builder f in
      let (builder, a') = expr parent_func builder a in
      let (builder, addr) = expr parent_func builder d in
      let f_name_suf = (str_of_ltyp (L.type_of a')) ^ (str_of_ltyp (L.type_of addr))
          in
      let f_name = "dfold" ^ f_name_suf ^ (string_of_int (Array.length (L.params
          arg_func))) in

      let func = (try Hashtbl.find func_tbl f_name
         with Not_found -> (
            let func = make_dfold_func f_name arg_func a' addr in
            Hashtbl.add func_tbl f_name func;
            func
         )
      ) in


      let actuals = add_parent_vars_to_actuals (L.params func) [arg_func;a';addr] in
      let actuals_arr = Array.of_list actuals in

      let accum_final = L.build_call func actuals_arr "dfoldaccumfinal" builder in
      (builder, accum_final)

   | SFunCall((_,SId("lmap")), [(_,f);(typlist_l,l)]) ->
      let (builder, arg_func) = expr parent_func builder f in
      let (builder, addr) = expr parent_func builder l in

      (* initiate empty list *)
      let typ = match (assc_typ_of_typlist typlist_l) with A.List(t) -> t | _ ->
          raise (Failure (internal_err ^ "lmap")) in
      let (builder, new_list_addr) = expr parent_func builder (SListLit(typ,[])) in

      let addr_typ = L.type_of addr in
      let arg_func_name = L.value_name arg_func in
      let ltyp = ltyp_of_typ typ in

      let wrapper_f_name = "foldwrapper" ^ arg_func_name in

      let wrapper_func = (try Hashtbl.find func_tbl wrapper_f_name
         with Not_found -> (
            let formals = [(addr_typ,"#a");(ltyp,"#e")] in
            let (wrapper_func, function_builder) = make_func wrapper_f_name (
                add_parent_vars_to_formals formals) addr_typ in

            init_params wrapper_func wrapper_f_name function_builder;

            let arg_func_params = L.params arg_func in

            let (function_builder, a') = expr wrapper_func function_builder (SId("#a
                ")) in
            let (function_builder, e') = expr wrapper_func function_builder (SId("#e
                ")) in
```

```
            let ladd_f_name = "ladd" ^ str_of_ltyp addr_typ in
            let ladd_func = (try Hashtbl.find func_tbl ladd_f_name
                with Not_found -> (
                    let ladd_func = make_ladd_func ladd_f_name addr e' in
                    Hashtbl.add func_tbl ladd_f_name ladd_func;
                    ladd_func
                )
            ) in

            let actuals = add_parent_vars_to_actuals arg_func_params [e'] in
            let actuals_arr = Array.of_list actuals in
            let e' = L.build_call arg_func actuals_arr "e" function_builder in
            let addr' = L.build_call ladd_func [|a';e';max_int|] "ladd"
                 function_builder in
            ignore(L.build_ret addr' function_builder);

            cleanup_func_vars wrapper_func;
            Hashtbl.add func_tbl wrapper_f_name wrapper_func;

            wrapper_func
        )
    ) in

    let f_name_suf = (str_of_ltyp addr_typ) ^ (str_of_ltyp addr_typ) in
    let f_name = "lfold" ^ f_name_suf ^ (string_of_int (Array.length (L.params
        wrapper_func))) in

    let lfold_func = (try Hashtbl.find func_tbl f_name
        with Not_found -> (
            let lfold_func = make_lfold_func f_name wrapper_func new_list_addr addr
                 in
            Hashtbl.add func_tbl f_name lfold_func;
            lfold_func
        )
    ) in

    let actuals = add_parent_vars_to_actuals (L.params lfold_func) [wrapper_func;
        new_list_addr;addr] in
    let actuals_arr = Array.of_list actuals in
    let accum_final = L.build_call lfold_func actuals_arr "lmapaccumfinal" builder
        in
    (builder, accum_final)

| SFunCall((_,SId("dmap")), [(_,f);(typlist_d,d)]) ->
    let (builder, arg_func) = expr parent_func builder f in
    let (builder, addr) = expr parent_func builder d in

    (* initiate empty dict *)
    let (t1, t2) = match (assc_typ_of_typlist typlist_d) with A.Dict(t1,t2) -> (t1,
        t2) | _ -> raise (Failure (internal_err ^ "dmap")) in
    let (builder, new_dict_addr) = expr parent_func builder (SDictLit(t1,t2,[])) in

    let addr_typ = L.type_of addr in
    let arg_func_name = L.value_name arg_func in
    let ltyp1 = ltyp_of_typ t1 in
    let ltyp2 = ltyp_of_typ t2 in

    let wrapper_f_name = "foldwrapper" ^ arg_func_name in
    let wrapper_func = (try Hashtbl.find func_tbl wrapper_f_name
        with Not_found -> (
            let formals = [(addr_typ,"#a");(ltyp1,"#k");(ltyp2,"#v")] in
            let (wrapper_func, function_builder) = make_func wrapper_f_name (
```

```
                add_parent_vars_to_formals formals) addr_typ in

            init_params wrapper_func wrapper_f_name function_builder;

            let arg_func_params = L.params arg_func in

            let (function_builder, a') = expr wrapper_func function_builder (SId("#a
                ")) in
            let (function_builder, k') = expr wrapper_func function_builder (SId("#k
                ")) in
            let (function_builder, v') = expr wrapper_func function_builder (SId("#v
                ")) in

            let dadd_f_name = "dadd" ^ str_of_ltyp addr_typ in
            let dadd_func = (try Hashtbl.find func_tbl dadd_f_name
                with Not_found -> (
                    let dadd_func = make_dadd_func dadd_f_name addr k' v' in
                    Hashtbl.add func_tbl dadd_f_name dadd_func;
                    dadd_func
                )
            ) in

            let actuals = add_parent_vars_to_actuals arg_func_params [k';v'] in
            let actuals_arr = Array.of_list actuals in
            let v' = L.build_call arg_func actuals_arr "v" function_builder in
            let addr' = L.build_call dadd_func [|a';k';v'|] "dadd" function_builder
                in
            ignore(L.build_ret addr' function_builder);

            cleanup_func_vars wrapper_func;
            Hashtbl.add func_tbl wrapper_f_name wrapper_func;

            wrapper_func
        )
    ) in


    let f_name_suf = (str_of_ltyp addr_typ) ^ (str_of_ltyp addr_typ) in
    let f_name = "dfold" ^ f_name_suf ^ (string_of_int (Array.length (L.params
        wrapper_func))) in

    let dfold_func = (try Hashtbl.find func_tbl f_name
        with Not_found -> (
            let dfold_func = make_dfold_func f_name wrapper_func new_dict_addr addr
                in
            Hashtbl.add func_tbl f_name dfold_func;
            dfold_func
        )
    ) in

    let actuals = add_parent_vars_to_actuals (L.params dfold_func) [wrapper_func;
        new_dict_addr;addr] in
    let actuals_arr = Array.of_list actuals in
    let accum_final = L.build_call dfold_func actuals_arr "dmapaccumfinal" builder
        in
    (builder, accum_final)

| SFunCall((_,SId("dkeys")), [(typlist,d)]) ->
    let (builder, d_addr) = expr parent_func builder d in
    let t1 = match (assc_typ_of_typlist typlist) with A.Dict(t1,_) -> t1 | _ ->
        raise (Failure (internal_err ^ "dkeys")) in
    let addr_ht = L.build_in_bounds_gep d_addr [|zero;two|] "dictht" builder in
    let ht = L.build_load addr_ht "ht" builder in
```

```
    (* making empty list for keys *)
    let (builder, l_addr) = expr parent_func builder (SListLit(t1,[])) in
    let l_addr_head = L.build_in_bounds_gep l_addr [|zero;one|] "listhead" builder
        in

    let keys_list_head = L.build_call ht_keys_list_func [|ht|] "keyslisthead"
        builder in
    ignore(L.build_store keys_list_head l_addr_head builder);

    (builder, l_addr)

| SFunCall((_,SId("rematch")), [(_,r);(_,s)]) ->
    let (builder, regex) = expr parent_func builder r in
    let (builder, addr) = expr parent_func builder s in
    let out = L.build_call re_match_func [|regex;addr|] "rematch" builder in
    (builder, out)

| SFunCall((_,SId("resub")), [(_,r);(_,s);(_,t);(_,n)]) ->
    let (builder, regex) = expr parent_func builder r in
    let (builder, s_addr) = expr parent_func builder s in
    let (builder, t_addr) = expr parent_func builder t in
    let (builder, n') = expr parent_func builder n in
    let new_s_addr = L.build_call re_sub_func [|regex;s_addr;t_addr;n'|] "resub"
        builder in
    (builder, new_s_addr)

| SAssign(v, (_,e)) ->
    let (builder, e') = expr parent_func builder e in
    L.set_value_name v e';
    Hashtbl.add var_tbl v (e', parent_func);
    (builder, e')

| SId(id) ->
    let e' = try
           let (e', _) = Hashtbl.find var_tbl id in
           e'
       with Not_found -> (
           match L.lookup_function id the_module with
               Some f -> f
               | None -> raise (Failure ("ID not found: " ^ id))
       )
    in
    (builder, e')

| SMatch(m) ->
    let merge_bb = L.append_block context "merge" parent_func in
    let build_br_merge = L.build_br merge_bb in
    let out_addr = L.build_alloca (ltyp_of_typ m.smtyp) "out" builder in

    let make_block (blocks, i) (_, block) =
        let bb = L.append_block context ("block" ^ string_of_int i) parent_func in
        let (_, bb_builder, bb_out) = List.fold_left stmt (parent_func, L.
            builder_at_end context bb, zero) block in
        ignore(L.build_store bb_out out_addr bb_builder);
        ignore(build_br_merge bb_builder);
        (bb::blocks, i + 1)
    in

    let make_expr_cond_block (blocks, i) (sexpr_or_def, _) then_block =
        let cond_bb = L.append_block context ("cond" ^ string_of_int i) parent_func
            in
        let cond_builder = L.builder_at_end context cond_bb in
```

```
        ignore(match sexpr_or_def with
            SExprMatch e ->
                let (cond_builder, cond) = expr parent_func cond_builder (SBinop(e,
                    A.Equal, m.sminput)) in
                ignore(L.build_cond_br cond then_block (List.hd blocks) cond_builder
                    );

            | SDefaultExpr ->
                ignore(L.build_br then_block cond_builder);
        );
        (cond_bb::blocks, i - 1)
    in

    let make_typ_cond_block (blocks, i) (typ_or_def, _) then_block =
        let cond_bb = L.append_block context ("cond" ^ string_of_int i) parent_func
             in
        let cond_builder = L.builder_at_end context cond_bb in
        ignore(match typ_or_def with
            A.TypMatch t ->
                let (cond_builder, cond) = expr parent_func cond_builder (STypComp(m
                    .sminput, t)) in
                ignore(L.build_cond_br cond then_block (List.hd blocks) cond_builder
                    );

            | A.DefaultTyp ->
                ignore(L.build_br then_block cond_builder);
        );
        (cond_bb::blocks, i - 1)
    in

    let cond_blocks = (match m.smatchlist with
        SValMatchList(l)->
            let (blocks, _) = List.fold_left make_block ([], 0) l in
            (* blocks is in reverse order *)
            let (cond_blocks, _) = List.fold_left2 make_expr_cond_block ([], (List.
                length blocks - 1)) (List.rev l) blocks
            in
            cond_blocks

        | STypMatchList(l) ->
            let (blocks, _) = List.fold_left make_block ([], 0) l in
            (* blocks is in reverse order *)
            let (cond_blocks, _) = List.fold_left2 make_typ_cond_block ([], (List.
                length blocks - 1)) (List.rev l) blocks
            in
            cond_blocks
    ) in

    ignore(L.build_br (List.hd cond_blocks) builder);
    let builder = L.builder_at_end context merge_bb in
    let out = L.build_load out_addr "matchout" builder in
    (builder, out)

| SIfElse(i) ->
    let (builder, cond) = expr parent_func builder (snd i.sicond) in
    let merge_bb = L.append_block context "merge" parent_func in
    let build_br_merge = L.build_br merge_bb in
    let out_addr = L.build_alloca (ltyp_of_typ i.sityp) "out" builder in

    let then_bb = L.append_block context "then" parent_func in
    let (_, if_builder, ifout) = List.fold_left stmt (parent_func, L.builder_at_end
         context then_bb, zero) i.sifblock in
    ignore(L.build_store ifout out_addr if_builder);
```

```
    ignore(build_br_merge if_builder);

    let else_bb = L.append_block context "else" parent_func in
    let (_, else_builder, elseout) = List.fold_left stmt (parent_func, L.
        builder_at_end context else_bb, zero) i.selseblock in
    ignore(L.build_store elseout out_addr else_builder);
    ignore(build_br_merge else_builder);

    ignore(L.build_cond_br cond then_bb else_bb builder);
    let builder = L.builder_at_end context merge_bb in
    let out = L.build_load out_addr "ifelseout" builder in
    (builder, out)

| SWhile(w) ->
    let ltyp = ltyp_of_typ w.swtyp in
    let out_addr = L.build_alloca ltyp "out" builder in

    let cond_bb = L.append_block context "while" parent_func in
    let (cond_builder, cond) = expr parent_func (L.builder_at_end context cond_bb)
        (snd w.swcond) in

    let body_bb = L.append_block context "while_body" parent_func in
    let (_, body_builder, body_out) = List.fold_left stmt (parent_func, L.
        builder_at_end context body_bb, zero) w.swblock in
    ignore(L.build_store body_out out_addr body_builder);
    ignore(L.build_br cond_bb body_builder);

    (* body runs once without checking cond *)
    ignore(L.build_br body_bb builder);

    let merge_bb = L.append_block context "merge" parent_func in
    ignore(L.build_cond_br cond body_bb merge_bb cond_builder);
    let builder = L.builder_at_end context merge_bb in
    let out = L.build_load out_addr "whileout" builder in
    (builder, out)

| SFunLit(f) ->
    let f_name = "fun" ^ (string_of_int !fun_name_i) in
    fun_name_i := !fun_name_i + 1;
    let formals = List.map (fun (t,n) -> (ltyp_of_typ t, n)) f.sformals in
    let ret_typ = ltyp_of_typ f.sftyp in

    let (func, function_builder) = make_func f_name (add_parent_vars_to_formals
        formals) ret_typ in

    init_params func f_name function_builder;

    let (_, function_builder, function_out) = List.fold_left stmt (func,
        function_builder, zero) f.sfblock in
    ignore(L.build_ret function_out function_builder);

    cleanup_func_vars func;

    (builder, func)

| SFunCall((_,e), l) ->
    let make_actuals (builder, actuals) (_, e) =
       let (builder, e') = expr parent_func builder e in
       (builder, e'::actuals)
    in
    let (builder, actuals) = List.fold_left make_actuals (builder, []) l in
    let (builder, func) = expr parent_func builder e in
```

```
    let actuals = add_parent_vars_to_actuals (L.params func) (List.rev actuals) in
    let actuals_arr = Array.of_list (actuals) in
    let out = L.build_call func actuals_arr "funcall" builder in
    (builder, out)

| SCast(t, (_,e)) ->
    let (builder, e') = expr parent_func builder e in
    let to_string fmt =
        let len = L.build_call snprintf_func [|L.const_null string_t;zero;fmt;e'|]
            "" builder
        in
        let len = L.build_add len one "" builder in
        let (builder, addr) = make_safe_malloc (L.build_array_malloc i8_t len "
            strcast") i8_t builder parent_func in
        ignore(L.build_call snprintf_func [|addr;len;fmt;e'|] "" builder);
        (builder, addr)
    in
    let e_ltyp = L.type_of e' in
    let c_ltyp = ltyp_of_typ (List.hd t) in
    let (builder, e_cast) =
        if c_ltyp = i32_t then (builder, (
            if e_ltyp = float_t then
                L.build_fptosi e' c_ltyp "intcast" builder
            else e'
        ))
        else if c_ltyp = float_t then (builder, (
            if e_ltyp = i32_t then
                L.build_sitofp e' c_ltyp "floatcast" builder
            else e'
        ))
        else if c_ltyp = string_t then (
            if e_ltyp = i32_t then
                to_string i32_format
            else if e_ltyp = i1_t then
                to_string i32_format
            else if e_ltyp = float_t then
                to_string float_format
            else (builder, e')
        ) else (
            (builder, e')
        )
    in
    (builder, e_cast)

| STypDefAssign(t, v, l) ->
    let t' = assc_typ_of_typlist [t] in
    let ut = match t' with A.UserTypDef(ut) -> ut | _ -> raise (Failure ((
        internal_err ^ "usertypdef"))) in
    let (utd_typ, name_pos) = Hashtbl.find utd_typs ut in
    let name = match L.struct_name utd_typ with Some n -> n | None -> raise (
        Failure ((internal_err ^ "no struct name assign"))) in
    let (builder, addr) = make_safe_malloc (L.build_malloc utd_typ (name ^ v))
        utd_typ builder parent_func in
    let fill_struct builder (n, (_,e)) =
        let (builder, e') = expr parent_func builder e in
        let pos = Hashtbl.find name_pos n in
        let addr_pos = L.build_in_bounds_gep addr [|zero;L.const_int i32_t pos|] ("
            tdassign_" ^ n) builder
        in
        ignore(L.build_store e' addr_pos builder);
        builder
    in
    let builder = List.fold_left fill_struct builder l in
```

```
          Hashtbl.add var_tbl v (addr, parent_func);
          (builder, addr)

      | SChildAcc((_,e), s) ->
          let (builder, e') = expr parent_func builder e in
          let utd_typ = L.element_type (L.type_of e') in
          let name = match L.struct_name utd_typ with Some n -> n | None -> raise (
              Failure (internal_err ^ "no struct name access")) in
          let (_, name_pos) = Hashtbl.find utd_typs name in
          let pos = Hashtbl.find name_pos s in
          let addr_pos = L.build_in_bounds_gep e' [|zero;L.const_int i32_t pos|] (name ^
              "." ^ s) builder in
          let out = L.build_load addr_pos ("load" ^ s) builder in
          (builder, out)

  and stmt (func, builder, out) = function
      SExprStmt(e) ->
          let (builder', out') = expr func builder (snd e) in
          (func, builder', out')

      | STypDecl(_, l) ->
          let make_typ (n, t) =
              Hashtbl.add ut_typs n (ltyp_of_typ t);
              ignore(L.define_global n (L.const_int i32_t !ut_i) the_module);
              ut_i := !ut_i + 1;
          in
          List.iter make_typ l;
          (func, builder, out)

      | STypDefDecl(v, l) ->
          let td = L.named_struct_type context v in
          let arr = Array.of_list (List.map (fun (t,_) -> ltyp_of_typ t) l) in
          let name_pos : (string, int) Hashtbl.t = Hashtbl.create 10 in
          List.iteri (fun i (_,n) -> Hashtbl.add name_pos n i) l;
          L.struct_set_body td arr false;
          Hashtbl.add utd_typs v (td, name_pos);
          (func, builder, out)
  in

  (* make stdin list *)
  let (builder, stdin_addr) = expr main builder (SListLit(A.String,[])) in
  Hashtbl.add var_tbl "stdin" (stdin_addr, main);
  L.set_value_name "stdin" stdin_addr;
  let stdin_addr_head = L.build_in_bounds_gep stdin_addr [|zero;one|] "stdinlisthead"
      builder in
  let stdin_head_node = L.build_call ll_of_stdin_func [||] "stdinheadnode" builder in
  ignore(L.build_store stdin_head_node stdin_addr_head builder);

  let (_, builder, _) = List.fold_left stmt (main, builder, zero) sast in
  ignore(L.build_call free_malloc_addrs_func [||] "" builder);
  ignore(L.build_ret (L.const_int i32_t 0) builder);

  the_module
```

8.1.7 cll.mll

```
(* Top level of the CLL compiler *)
(* Author: Annalise Mariottini (aim2120) *)

type action = Ast | Sast | LLVM_IR | Compile

let () =
        let action = ref Compile in
```

```
let set_action a () = action := a in
let speclist = [
    ("-a", Arg.Unit (set_action Ast), "Print the AST");
    ("-s", Arg.Unit (set_action Sast), "Print the SAST");
    ("-l", Arg.Unit (set_action LLVM_IR), "Print the generated LLVM IR");
    ("-c", Arg.Unit (set_action Compile), "Compile the program");
] in
let usage_msg = "usage: ./cll.native [-a|-s|-l] [file.cll]" in
let channel = ref stdin in
let filename = ref "" in
Arg.parse speclist (fun file -> filename := file; channel := open_in file;)
    usage_msg;
let ast = try
        let lexbuf = Lexing.from_channel !channel in
        Parser.program Scanner.token lexbuf
    with Parsing.Parse_error ->
        let s = ("!!!ERROR!!! line " ^ string_of_int !Scanner.line_num ^ ": parsing
             error")
        in
        raise (Failure (s))
    | Failure(msg) ->
        let s = ("!!!ERROR!!! line " ^ string_of_int !Scanner.line_num ^ ": " ^ msg
            )
        in
        raise (Failure (s))
in
match !action with
    Ast -> print_string (Ast.string_of_program ast)
    |_ ->
        let (env, sast) = try Semant.check_ast ast
            with Failure(msg) ->
                let file_out = !filename ^ ".ast" in
                let log = open_out file_out in
                Printf.fprintf log "%s" (Ast.string_of_program ast);
                close_out log;
                let s = (msg ^ "\n(Check " ^ file_out ^ " for line numbers)")
                in
                raise (Failure (s))
        in
        match !action with
        Ast -> ()
        | Sast -> print_string (Sast.string_of_sprogram sast)
        | LLVM_IR -> print_string (Llvm.string_of_llmodule (Codegen.translate env
            sast))
        | Compile -> let m = Codegen.translate env sast in
                Llvm_analysis.assert_valid_module m;
                print_string (Llvm.string_of_llmodule m)
```

8.2 C Lib Files

8.2.1 find_prime.c

```
/*
 * Author: Annalise Mariottini (aim2120)
 */
#include "find_prime.h"

const int primes[] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61,
    67, 71,
73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167,
    173,
```

179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271, 277, 281,
283, 293, 307, 311, 313, 317, 331, 337, 347, 349, 353, 359, 367, 373, 379, 383, 389, 397, 401, 409,
419, 421, 431, 433, 439, 443, 449, 457, 461, 463, 467, 479, 487, 491, 499, 503, 509, 521, 523, 541,
547, 557, 563, 569, 571, 577, 587, 593, 599, 601, 607, 613, 617, 619, 631, 641, 643, 647, 653, 659,
661, 673, 677, 683, 691, 701, 709, 719, 727, 733, 739, 743, 751, 757, 761, 769, 773, 787, 797, 809,
811, 821, 823, 827, 829, 839, 853, 857, 859, 863, 877, 881, 883, 887, 907, 911, 919, 929, 937, 941,
947, 953, 967, 971, 977, 983, 991, 997, 1009, 1013, 1019, 1021, 1031, 1033, 1039, 1049, 1051, 1061, 1063, 1069,
1087, 1091, 1093, 1097, 1103, 1109, 1117, 1123, 1129, 1151, 1153, 1163, 1171, 1181, 1187, 1193, 1201, 1213, 1217, 1223,
1229, 1231, 1237, 1249, 1259, 1277, 1279, 1283, 1289, 1291, 1297, 1301, 1303, 1307, 1319, 1321, 1327, 1361, 1367, 1373,
1381, 1399, 1409, 1423, 1427, 1429, 1433, 1439, 1447, 1451, 1453, 1459, 1471, 1481, 1483, 1487, 1489, 1493, 1499, 1511,
1523, 1531, 1543, 1549, 1553, 1559, 1567, 1571, 1579, 1583, 1597, 1601, 1607, 1609, 1613, 1619, 1621, 1627, 1637, 1657,
1663, 1667, 1669, 1693, 1697, 1699, 1709, 1721, 1723, 1733, 1741, 1747, 1753, 1759, 1777, 1783, 1787, 1789, 1801, 1811,
1823, 1831, 1847, 1861, 1867, 1871, 1873, 1877, 1879, 1889, 1901, 1907, 1913, 1931, 1933, 1949, 1951, 1973, 1979, 1987,
1993, 1997, 1999, 2003, 2011, 2017, 2027, 2029, 2039, 2053, 2063, 2069, 2081, 2083, 2087, 2089, 2099, 2111, 2113, 2129,
2131, 2137, 2141, 2143, 2153, 2161, 2179, 2203, 2207, 2213, 2221, 2237, 2239, 2243, 2251, 2267, 2269, 2273, 2281, 2287,
2293, 2297, 2309, 2311, 2333, 2339, 2341, 2347, 2351, 2357, 2371, 2377, 2381, 2383, 2389, 2393, 2399, 2411, 2417, 2423,
2437, 2441, 2447, 2459, 2467, 2473, 2477, 2503, 2521, 2531, 2539, 2543, 2549, 2551, 2557, 2579, 2591, 2593, 2609, 2617,
2621, 2633, 2647, 2657, 2659, 2663, 2671, 2677, 2683, 2687, 2689, 2693, 2699, 2707, 2711, 2713, 2719, 2729, 2731, 2741,
2749, 2753, 2767, 2777, 2789, 2791, 2797, 2801, 2803, 2819, 2833, 2837, 2843, 2851, 2857, 2861, 2879, 2887, 2897, 2903,
2909, 2917, 2927, 2939, 2953, 2957, 2963, 2969, 2971, 2999, 3001, 3011, 3019, 3023, 3037, 3041, 3049, 3061, 3067, 3079,
3083, 3089, 3109, 3119, 3121, 3137, 3163, 3167, 3169, 3181, 3187, 3191, 3203, 3209, 3217, 3221, 3229, 3251, 3253, 3257,
3259, 3271, 3299, 3301, 3307, 3313, 3319, 3323, 3329, 3331, 3343, 3347, 3359, 3361, 3371, 3373, 3389, 3391, 3407, 3413,
3433, 3449, 3457, 3461, 3463, 3467, 3469, 3491, 3499, 3511, 3517, 3527, 3529, 3533, 3539, 3541, 3547, 3557, 3559, 3571,
3581, 3583, 3593, 3607, 3613, 3617, 3623, 3631, 3637, 3643, 3659, 3671, 3673, 3677, 3691, 3697, 3701, 3709, 3719, 3727,
3733, 3739, 3761, 3767, 3769, 3779, 3793, 3797, 3803, 3821, 3823, 3833, 3847, 3851, 3853, 3863, 3877, 3881, 3889, 3907,
3911, 3917, 3919, 3923, 3929, 3931, 3943, 3947, 3967, 3989, 4001, 4003, 4007, 4013, 4019, 4021, 4027, 4049, 4051, 4057,
4073, 4079, 4091, 4093, 4099, 4111, 4127, 4129, 4133, 4139, 4153, 4157, 4159, 4177, 4201, 4211, 4217, 4219, 4229, 4231,
4241, 4243, 4253, 4259, 4261, 4271, 4273, 4283, 4289, 4297, 4327, 4337, 4339, 4349, 4357, 4363, 4373, 4391, 4397, 4409,
4421, 4423, 4441, 4447, 4451, 4457, 4463, 4481, 4483, 4493, 4507, 4513, 4517, 4519, 4523, 4547, 4549, 4561, 4567, 4583,
4591, 4597, 4603, 4621, 4637, 4639, 4643, 4649, 4651, 4657, 4663, 4673, 4679, 4691, 4703, 4721, 4723, 4729, 4733, 4751,
4759, 4783, 4787, 4789, 4793, 4799, 4801, 4813, 4817, 4831, 4861, 4871, 4877, 4889, 4903, 4909, 4919, 4931, 4933, 4937,
4943, 4951, 4957, 4967, 4969, 4973, 4987, 4993, 4999, 5003, 5009, 5011, 5021, 5023, 5039,

```
    5051, 5059, 5077, 5081, 5087,
5099, 5101, 5107, 5113, 5119, 5147, 5153, 5167, 5171, 5179, 5189, 5197, 5209, 5227, 5231,
    5233, 5237, 5261, 5273, 5279,
5281, 5297, 5303, 5309, 5323, 5333, 5347, 5351, 5381, 5387, 5393, 5399, 5407, 5413, 5417,
    5419, 5431, 5437, 5441, 5443,
5449, 5471, 5477, 5479, 5483, 5501, 5503, 5507, 5519, 5521, 5527, 5531, 5557, 5563, 5569,
    5573, 5581, 5591, 5623, 5639,
5641, 5647, 5651, 5653, 5657, 5659, 5669, 5683, 5689, 5693, 5701, 5711, 5717, 5737, 5741,
    5743, 5749, 5779, 5783, 5791,
5801, 5807, 5813, 5821, 5827, 5839, 5843, 5849, 5851, 5857, 5861, 5867, 5869, 5879, 5881,
    5897, 5903, 5923, 5927, 5939,
5953, 5981, 5987, 6007, 6011, 6029, 6037, 6043, 6047, 6053, 6067, 6073, 6079, 6089, 6091,
    6101, 6113, 6121, 6131, 6133,
6143, 6151, 6163, 6173, 6197, 6199, 6203, 6211, 6217, 6221, 6229, 6247, 6257, 6263, 6269,
    6271, 6277, 6287, 6299, 6301,
6311, 6317, 6323, 6329, 6337, 6343, 6353, 6359, 6361, 6367, 6373, 6379, 6389, 6397, 6421,
    6427, 6449, 6451, 6469, 6473,
6481, 6491, 6521, 6529, 6547, 6551, 6553, 6563, 6569, 6571, 6577, 6581, 6599, 6607, 6619,
    6637, 6653, 6659, 6661, 6673,
6679, 6689, 6691, 6701, 6703, 6709, 6719, 6733, 6737, 6761, 6763, 6779, 6781, 6791, 6793,
    6803, 6823, 6827, 6829, 6833,
6841, 6857, 6863, 6869, 6871, 6883, 6899, 6907, 6911, 6917, 6947, 6949, 6959, 6961, 6967,
    6971, 6977, 6983, 6991, 6997,
7001, 7013, 7019, 7027, 7039, 7043, 7057, 7069, 7079, 7103, 7109, 7121, 7127, 7129, 7151,
    7159, 7177, 7187, 7193, 7207,
7211, 7213, 7219, 7229, 7237, 7243, 7247, 7253, 7283, 7297, 7307, 7309, 7321, 7331, 7333,
    7349, 7351, 7369, 7393, 7411,
7417, 7433, 7451, 7457, 7459, 7477, 7481, 7487, 7489, 7499, 7507, 7517, 7523, 7529, 7537,
    7541, 7547, 7549, 7559, 7561,
7573, 7577, 7583, 7589, 7591, 7603, 7607, 7621, 7639, 7643, 7649, 7669, 7673, 7681, 7687,
    7691, 7699, 7703, 7717, 7723,
7727, 7741, 7753, 7757, 7759, 7789, 7793, 7817, 7823, 7829, 7841, 7853, 7867, 7873, 7877,
    7879, 7883, 7901, 7907, 7919};
int find_prime(int n) {
    int len = sizeof(primes);
    for (int i = 0; i < len; i++) {
        if (n <= primes[i]) {
            return primes[i];
        }
    }
    // shouldn't reach here
    return 0;
}
```

8.2.2 hash_table.c

```
/*
 * Author: Annalise Mariottini (aim2120)
 * Sourced from https://gist.github.com/tonious/1377667 with many changes made
 */

/*
 * Simple hash table implementation
 */

/* Read this comment first: https://gist.github.com/tonious/1377667#gistcomment-2277101
 * 2017-12-05
 *
 *  -- T.
 */

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

```
#include <limits.h>
#include <string.h>
#include <stdbool.h>
#include "hash_table.h"
#include "find_prime.h"
#include "linked_list.h"
#include "malloc_manager.h"

struct entry_s {
    char *key;
    char *value;
    struct entry_s *next;
};

typedef struct entry_s entry_t;

struct hashtable_s {
    int size;
    int filled;
    struct entry_s **table;
    bool key_is_string;
};

typedef struct hashtable_s hashtable_t;


/* Create a new hashtable. */
hashtable_t *ht_create( int size, bool key_is_string ) {

    hashtable_t *hashtable = NULL;
    int i;

    if( size < 1 ) size = 1;

     size = find_prime(size * 2);

    /* Allocate the table itself. */
    if( ( hashtable = malloc( sizeof( hashtable_t ) ) ) == NULL ) {
        exit(1);
    }

    add_malloc_addr((char *)hashtable);

    /* Allocate pointers to the head nodes. */
    if( ( hashtable->table = malloc( sizeof( entry_t * ) * size ) ) == NULL ) {
        exit(1);
    }

    add_malloc_addr((char *)hashtable->table);


    for( i = 0; i < size; i++ ) {
        hashtable->table[i] = NULL;
    }

    hashtable->filled = 0;
    hashtable->size = size;
    hashtable->key_is_string = key_is_string;

    return hashtable;
}

/* Copy old table into new larger table */
```

```
hashtable_t *ht_grow( hashtable_t *hashtable ) {
    int new_size;

    new_size = hashtable->filled;
    new_size = find_prime(new_size * 2);

    hashtable_t* new_hashtable = ht_create(new_size, hashtable->key_is_string);

    for (int i = 0; i < hashtable->size; i++) {
        entry_t *pair = hashtable->table[i];
        while ( pair != NULL && pair->key != NULL ) {
            ht_add(new_hashtable, pair->key, pair->value);
            pair = pair->next;
        }
    }

    return new_hashtable;
}


/* Hash a string for a particular hash table. */
int ht_hash( hashtable_t *hashtable, char *key ) {

    unsigned long int hashval = 5381;
    int i = 0;

    if (hashtable->key_is_string) {
        /* Convert our string to an integer */
        /* djb2 hash function */
        while( hashval < ULONG_MAX && i < strlen( key ) ) {
            hashval += key[ i ];
            hashval = hashval << 5;
            i++;
        }
    }
    else {
        /* https://stackoverflow.com/questions/664014/
         * what-integer-hash-function-are-good-that-accepts-an-integer-hash-key */
        hashval += key[0];
        hashval = ((hashval >> 16) ^ hashval) * 0x45d9f3b;
        hashval = ((hashval >> 16) ^ hashval) * 0x45d9f3b;
        hashval = (hashval >> 16) ^ hashval;
    }

    return hashval % hashtable->size;
}

/* Create a key-value pair. */
entry_t *ht_newpair( char *key, char *value ) {
    entry_t *newpair;

    if( ( newpair = malloc( sizeof( entry_t ) ) ) == NULL ) {
        exit(1);
    }
    add_malloc_addr((char *)newpair);

    if( ( newpair->key = malloc( sizeof( char * ) ) ) == NULL ) {
        exit(1);
    }
    add_malloc_addr((char *)newpair->key);
    memcpy( newpair->key, key, sizeof ( char * ) );

    if( ( newpair->value = malloc( sizeof( char * ) ) ) == NULL ) {
```

```
        exit(1);
    }
    add_malloc_addr((char *)newpair->value);
    memcpy( newpair->value, value, sizeof ( char * ) );

    newpair->next = NULL;

    return newpair;
}

/* Insert a key-value pair into a hash table. */
hashtable_t *ht_add( hashtable_t *hashtable, char *key, char *value ) {
    int bin = 0;
    bool kis = hashtable->key_is_string;
    entry_t *newpair = NULL;
    entry_t *next = NULL;
    entry_t *last = NULL;

    char *to_hash = key;
    if (kis) {
        to_hash = *(char **)key;
    }
    bin = ht_hash( hashtable, to_hash );

    next = hashtable->table[ bin ];

    char **key_ptr;
    char *key_;
    char **nextkey_ptr;
    char *nextkey_;

    if (kis) {
        key_ptr = (char **) key;
        key_ = *key_ptr;
    }

    bool addr_cmp, str_cmp = false;
    while( next != NULL && next->key != NULL ) {
        addr_cmp = !kis && memcmp( key, next->key, 1) == 0;
        if (kis) {
            nextkey_ptr = (char **) (next->key);
            nextkey_ = *nextkey_ptr;
            str_cmp = strcmp( key_, nextkey_ ) == 0;
        }
        if (addr_cmp || str_cmp) {
            break;
        }

        last = next;
        next = next->next;
    }

    /* There's already a pair. Let's replace that string. */
    if( next != NULL && next->key != NULL && (addr_cmp || str_cmp) ) {

        memcpy( next->value, value, sizeof( char * ) );

    /* Nope, could't find it. Time to grow a pair. */
    } else {
        hashtable->filled++;

        newpair = ht_newpair( key, value );
```

```
        /* We're at the start of the linked list in this bin. */
        if( next == hashtable->table[ bin ] ) {
            newpair->next = next;
            hashtable->table[ bin ] = newpair;

        /* We're at the end of the linked list in this bin. */
        } else if ( next == NULL ) {
            last->next = newpair;
        /* We're in the middle of the list. */
        } else {
            newpair->next = next;
            last->next = newpair;
        }
    }

     /* Growing the table if the size is < 1.3 * filled */
    if ((hashtable->filled * 1.3) > (float)hashtable->size) {
        hashtable = ht_grow(hashtable);
    }

    return hashtable;
}

hashtable_t *ht_remove(hashtable_t *hashtable, char *key) {
    int bin = 0;
    bool kis = hashtable->key_is_string;
    entry_t *newpair = NULL;
    entry_t *next = NULL;
    entry_t *last = NULL;

    char *to_hash = key;
    if (kis) {
        to_hash = *(char **)key;
    }
    bin = ht_hash( hashtable, to_hash );

    next = hashtable->table[ bin ];

    char **key_ptr;
    char *key_;
    char **nextkey_ptr;
    char *nextkey_;

    if (kis) {
        key_ptr = (char **) key;
        key_ = *key_ptr;
    }

    bool addr_cmp, str_cmp = false;
    while( next != NULL && next->key != NULL ) {
        addr_cmp = !kis && memcmp( key, next->key, 1) == 0;
        if (kis) {
            nextkey_ptr = (char **) (next->key);
            nextkey_ = *nextkey_ptr;
            str_cmp = strcmp( key_, nextkey_ ) == 0;
        }
        if (addr_cmp || str_cmp) {
            break;
        }

        last = next;
        next = next->next;
    }
```

```
    /* found the key to remove */
    if( next != NULL && (addr_cmp || str_cmp) ) {

        if (last != NULL) {
            last->next = next->next;
        } else {
            hashtable->table[ bin ] = next->next;
        }

        hashtable->filled--;

    } /* else -> couldn't find key, return hashtable unchanged */

    return hashtable;
}

bool ht_mem( hashtable_t *hashtable, char *key ) {
    if (ht_get(hashtable, key) == NULL) {
        return false;
    } else {
        return true;
    }
}

/* Retrieve a key-value pair from a hash table. */
char *ht_get( hashtable_t *hashtable, char *key ) {
    int bin = 0;
    bool kis = hashtable->key_is_string;
    entry_t *pair;

    char *to_hash = key;
    if (kis) {
        to_hash = *(char **)key;
    }
    bin = ht_hash( hashtable, to_hash );

    /* Step through the bin, looking for our value. */
    pair = hashtable->table[ bin ];

    char **key_ptr;
    char *key_;
    char **pairkey_ptr;
    char *pairkey_;

    if (kis) {
        key_ptr = (char **) key;
        key_ = *key_ptr;
    }

    bool addr_cmp, str_cmp = false;

    while( pair != NULL && pair->key != NULL ) {
        if (kis) {
            pairkey_ptr = (char **) (pair->key);
            pairkey_ = *pairkey_ptr;
            str_cmp = strcmp( key_, pairkey_ ) == 0;
        }
        addr_cmp = !kis && memcmp( key, pair->key, 1) == 0;
        if (addr_cmp || str_cmp) {
            break;
        }
```

```
        pair = pair->next;
    }

    /* Did we actually find anything? */
    if( pair == NULL || pair->key == NULL || !(addr_cmp || str_cmp) ) {
        exit(1);

    } else {
        return pair->value;
    }
}

char **ht_keys(hashtable_t *hashtable) {
    char **keys;

    if ((keys = malloc((sizeof (char**)) * hashtable->filled)) == NULL) {
        exit(1);
    }
    add_malloc_addr((char *)keys);

    int j = 0;

    for (int i = 0; i < hashtable->size; i++) {
        entry_t *pair = hashtable->table[i];
        while (pair != NULL) {
            keys[j] = pair->key;
            pair = pair->next;
            j++;
        }
    }

    for (int i = 0; i < hashtable->filled; i++) {
    };

    return keys;
};

ll_node *ht_keys_list(hashtable_t *hashtable) {
    ll_node *head_key = NULL;

    for (int i = 0; i < hashtable->size; i++) {
        entry_t *pair = hashtable->table[i];
        while (pair != NULL) {
            head_key = ll_add(head_key, pair->key, 0);
            pair = pair->next;
        }
    }

    return head_key;
};

// for debugging, not pretty
int ht_print (hashtable_t *hashtable) {
    int i = 0;
    for (int i = 0; i < hashtable->size; i++) {
        entry_t *pair = hashtable->table[i];
        printf("(%d) ", i);
        while ( pair != NULL) {
            if (hashtable->key_is_string) {
                printf("%s : ", *(char**) pair->key);
            }
            else {
                printf("%lu : ", (unsigned long) pair->key);
```

```
            }
            printf("%lu; ", (unsigned long) pair->value);
            pair = pair->next;
        }
        printf("\n");
    }
    return i;
}

int ht_size (hashtable_t *hashtable) {
    return hashtable->filled;
}
```

8.2.3 linked_list.c

```
/*
 * Author: Annalise Mariottini (aim2120)
 */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <stdbool.h>
#include "linked_list.h"
#include "malloc_manager.h"

struct ll_node_s {
    char *data;
    struct ll_node_s *next;
};

typedef struct ll_node_s ll_node;

ll_node *ll_create(char *data) {
    ll_node *new_node;

    if ((new_node = malloc(sizeof (ll_node))) == NULL) {
        exit(1);
    }
    add_malloc_addr((char *)new_node);

    if ((new_node->data = malloc(sizeof (char *))) == NULL) {
        exit(1);
    }
    add_malloc_addr((char *)new_node->data);

    memcpy(new_node->data, data, sizeof (char *));

    new_node->next = NULL;
    return new_node;
}

ll_node *ll_add(ll_node *head, char *data, int n) {
    ll_node *curr = head;
    ll_node *prev = NULL;
    ll_node *new_node;

    if (curr == NULL) {
        return ll_create(data);
    }

    for (int i = 0; i < n && curr != NULL; i++) {
        prev = curr;
```

```
        curr = curr->next;
    }

    new_node = ll_create(data);

    if (new_node == NULL) {
        exit(1);
    }

    if (prev != NULL) {
        prev->next = new_node;
        new_node->next = curr;
    } else {
        new_node->next = head;
        head = new_node;
    }

    return head;
}

ll_node *ll_append(ll_node *head, ll_node *to_append) {
    ll_node *curr = head;

    if (curr == NULL) {
        return to_append;
    }

    while (curr->next != NULL) {
        curr = curr->next;
    }

    curr->next = to_append;

    return head;
};

int ll_mem(ll_node *head, char *data, bool is_string) {
    ll_node *curr = head;
    char *data_;
    char *currdata_;
    int n = -1;
    int i = 0;
    bool addr_cmp = false, str_cmp = false;

    if (is_string) {
        data_ = *(char **)data;
    }

    while (curr != NULL) {
        addr_cmp = memcmp( data, curr->data, 1) == 0;

        if (is_string) {
            currdata_ = *(char **)(curr->data);
            str_cmp = strcmp(data_, currdata_) == 0;
        }

        if (addr_cmp || str_cmp) {
            n = i;
            break;
        }
        curr = curr->next;
        i++;
    }
```

```
    return n;
}

/* returns new head node */
ll_node *ll_remove(ll_node *head, int n) {
    ll_node *curr = head;
    ll_node *prev = NULL;

    if (curr == NULL) {
        exit(1);
    }

    for (int i = 0; i < n && curr->next != NULL; i++) {
        prev = curr;
        curr = curr->next;
    }

    if (prev != NULL) {
        prev->next = curr->next;
    } else {
        head = head->next;
    }

    return head;
}

ll_node *ll_next(ll_node *node) {
    if (node == NULL) {
        exit(1);
    }

    return node->next;
}

char *ll_get(ll_node *head, int n) {
    ll_node *curr = head;

    if (curr == NULL) {
        exit(1);
    }

    for ( int i = 0; i < n && curr->next != NULL; i++ ) {
        curr = curr->next;
    }
    return curr->data;
}

// for debugging, not pretty
int ll_print(ll_node *head) {
    int i = 0;
    ll_node *curr = head;
    while(curr != NULL) {
        if (curr->data != NULL) {
            printf("%lu ",(unsigned long)curr->data);
            i++;
        }
        curr = curr->next;
    }
    printf("\n");
    return i;
}
```

```
int ll_size(ll_node *head) {
    int i = 0;
    ll_node *curr = head;
    while(curr != NULL) {
        i++;
        curr = curr->next;
    }
    return i;
}
```

8.2.4 malloc_manager.c

```
/*
 * Author: Annalise Mariottini (aim2120)
 */

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

#define INITSIZE 4096

char **malloc_addr;
int malloc_addr_size;
int addr_i;

void init_malloc_addr() {
    malloc_addr = malloc(sizeof(char *) * INITSIZE);
    if (malloc_addr == NULL) {
        exit(1);
    }
    memset(malloc_addr, 0, sizeof(char *) * INITSIZE);
    malloc_addr_size = INITSIZE;
    addr_i = 0;
}

void add_malloc_addr(char *addr) {
    malloc_addr[addr_i] = addr;
    addr_i++;

    if (addr_i == malloc_addr_size) {

        int old_malloc_addr_size = malloc_addr_size;
        malloc_addr_size *= 2;
        char **new_malloc_addr = malloc(sizeof(char *) * malloc_addr_size);
        memset(new_malloc_addr, 0, sizeof(char *) * malloc_addr_size);
        memcpy(new_malloc_addr, malloc_addr, sizeof(char *) * old_malloc_addr_size);
        malloc_addr = new_malloc_addr;
    }
}

void free_malloc_addrs() {
    for (int i = 0; i < malloc_addr_size; i++) {
        if (malloc_addr[i] == NULL) {
            break;
        }
        free(malloc_addr[i]);
    }
}
```

8.2.5 regex.c

```
/*
 * Author: Annalise Mariottini (aim2120)
 */

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

regex_t *re_create(char *r) {
    regex_t *regex;
    if ((regex = malloc (sizeof(regex_t))) == NULL) {
        return NULL;
    }
    if (regcomp(regex, r, REG_EXTENDED)) return NULL;
    return regex;
}

bool re_match(regex_t *r, char *s) {
    return regexec(r, s, 0, NULL, 0) == 0;
}

char *re_sub(regex_t *r, char *s, char *t, int n) {
    char *new_s;
    char *s_ = s;
    int s_len, t_len, match_len, new_s_len;
    int match_start, match_end;
    int ret;

    while (1) {
        regmatch_t buf[n+1];

        if (regexec(r, s_, n+1, buf, 0)) return s_;

        match_start = buf[n].rm_so;
        match_end = buf[n].rm_eo;
        if (match_start < 0) {
            break;
        }

        s_len = strlen(s_);
        t_len = strlen(t);
        match_len = match_end-match_start;

        new_s_len = (s_len - match_len) + t_len + 1;

        if ((new_s = malloc (sizeof(char *) * new_s_len)) == NULL) {
            exit(1);
        }

        memmove(new_s, s_, match_start);
        memmove(&new_s[match_start], t, t_len);
        memmove(&new_s[match_start+t_len], &s_[match_end], s_len-match_end);
        new_s[new_s_len-1] = '\0';

        s_ = new_s;
    }

    return s_;
}
```

8.2.6 stdin.c

```
/*
 * Author: Annalise Mariottini (aim2120)
 */

#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include <stdio.h>
#include <limits.h>
#include "linked_list.h"
#include "malloc_manager.h"

#define LRGBUFSIZE 4096

ll_node *ll_of_stdin() {
    char buf[1];
    char largebuf[LRGBUFSIZE];
    int strlen = 0;
    ll_node *ll_head = NULL;

    fseek (stdin, 0, SEEK_END);
    if (ftell (stdin) == 0) {
        return NULL;
    }

    fseek(stdin, 0, SEEK_SET);
    memset(largebuf, 0, sizeof(char) * (LRGBUFSIZE - 1));

    while(read(0, buf, sizeof(buf))>0) {
        char c = *buf;
        char *s;
        char **s_addr;
        bool is_whitespace = (c == ' ' || c == '\n' || c == '\t' || c == '\r');

        if (strlen > 0 && is_whitespace) {
            s = malloc(sizeof(char) * (strlen + 1));
            if (s == NULL) {
                exit(1);
            }
            add_malloc_addr(s);

            s_addr = malloc(sizeof(char *));
            if (s_addr == NULL) {
                exit(1);
            }
            add_malloc_addr((char *)s_addr);

            memcpy(s, largebuf, sizeof(char) * strlen);
            s[strlen] = '\0';
            memcpy(s_addr, &s, sizeof(char *));

            // remember: ll data is janky universal char * addresses, not values
            ll_head = ll_add(ll_head, (char *)s_addr, INT_MAX);

            strlen = 0;
            memset(largebuf, 0, sizeof(char) * (LRGBUFSIZE - 1));
        } else if (!is_whitespace) {
            largebuf[strlen] = c;
            strlen++;

            if (strlen == LRGBUFSIZE) {
```

```
                exit(1);
            }
        }
    }

    return ll_head;
}
```