DRRTY Programming Final Report

Programming Languages and Translators Fall 2021

Dylan Bamirny (db3381)	**Manager**
Richard Lopez (rl3020)	**Language Guru**
Rania Alshafie (rta2114@barnard.edu)	**Tester**
Trinity Sazo (ts3185)	**System Architect**

1 Introduction

DRRTY is an imperative, scripting language that offers back-end programmers a way to code front-end HTML pages. The language aims to integrate HTML and backend logic into one coding environment, and it enables developers the ability to build more complex HTML modules similar to the functionality of the React Javascript library. DRRTY's syntax closely follows that of Java and Python.

2 Language Tutorial

2.1 Environment Setup

To set up the DRRTY compiler, the language requires that OCaml, GCC, Clang, LLVM, and LLVM module bitreader be installed on your local machine. Alternatively, Docker can be installed and invoked using a terminal command, which is our preferred method for setting up the environment.

2.2 Compilation Guide

Enter the `/drrty` directory to build the compiler. Once in the directory, invoke Docker using: `docker run --rm -it -v pwd:/home/drrty -w=/home/drrty columbiasedwards/plt`.

Inside Docker, run `make clean` to remove any pre-generated files that may be present. Next, build the compiler and run all tests by entering the `make` command.

Once the environment is set up, source code can be written in files with a .drrt extension, and all code must follow rules outlined in the LRM (see Section 3). With a source code file, run `./run.sh <filename>.drrt` to compile and produce an HTML file.

2.3 Language Tutorial

After you compile DRRTY, you are ready to begin programming! The following steps outline a basic guide for developing programs using DRRTY. As an example, we can write a simple helloWorld.drrt program, which compiles and outputs an HTML file with a basic header: "<h1> hello world </h1>". In our helloWorld.drrt file, we have the following code:

```
def int main{
```

```
    str helloWorld;
    helloWorld = “Hello, World!”;
    createHTMLDocument("myCSS.css");
    makeHeader(helloWorld);
    return 0;
}
```

In the Drrty program defined above, there are a few key pieces of code to pay particular attention to. The first is the manner in which functions are defined. We utilize a mix syntax seen in the programming languages Python and Java to initialize functions.

```
def int main{ ... }
```

The first code snippet "def" illustrattes that a function is being declared, "int" is the return type, and "main" is the name of the function. "main" is an especially unique key word, as it defines the entry point into the written program.

Variable declarations follow the same pattern "type nameOfVariable ;".

```
    str helloWorld;
    helloWorld = “Hello, World!”;
```

The name and type of variable must be defined before providing the variable with a value, and only after can a user assign a value to the variable.

```
createHTMLDocument("myCSS.css");
makeHeader(helloWorld);
```

Next, we utilize functions defined in our HTML library to create and print HTML. The first step is to build the template HTML using the function "createHTMLDocument('myCSS.css')". This builds the necessary template to create HTML pages and view these pages on the browser. Finally, "makeHeader(argument1)" prints the string passed to it through the argument. This in turn will lead to a header tag, with its inner content being the <h1> “Hello world”</h1> we defined earlier. The final return statement indicates success if the return value is 0. Anything other than 0 indicates an error.

To compile, run, and create the HTML page, run the following command:
`./run.sh helloWorld.drrt`

3 Language Reference Manual

3.1 Lexical Structure

3.1.1 Comments

Comments are notes put in by the user that is ignored by the compiler. We signify them to be any text placed between a ‘/*’ and ‘*/’.

```
/* This is a comment */
/* They can also extend to
    be multiple lines */
```

3.1.2 Identifiers

3.1.2.1 Variable Declarations

There are specific ways to declare a valid variable name to store information in DRRTY. Variable names can begin with a lowercase letter, followed by a series of alphabetic or numeric characters. They cannot begin with a number or capital letters but can include underscores. They cannot include special characters and cannot be a DRRTY reserved keyword.

```
/* Variables must be declared first,
followed by an assignment */
int x;
x = 4;

str string1;
string1 = “Hello, World!”;
```

3.1.2.2 Function Declarations

Function declarations are signified by the keyword "def", followed by the function name and any parameters that the function might take placed in parentheses. The beginning and end of a function's code to execute is bound by two curly braces.

```
/* Variables must be declared */
def plus(int a, int b){
      return a + b;
}
```

3.1.3 Keywords

There are a total of 15 keywords in the DRRTY language:

`int`	`float`	`bool`
`str`	`list`	`void`
`for`	`while`	`def`
`True`	`False`	`return`
`if`	`else`	`end`

3.1.4 Separators

There are curly braces, parentheses, semicolons, colons, commas, and brackets present as syntactical separators in the DRRTY language:

`(`	`{`	`[`
`)`	`}`	`]`
`;`	`,`	`:`

3.1.5 Operators

There are 14 operators in DRRTY:

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

3.1.5.1 Assignment

The assignment operator, signified by an equal sign (=), is used to store information or different values of any type into a variable. The assignment operator is used to set any variable name equal to a value of any data type.

```
int x;
x = 5; /* assigns 5 to int x */

str greeting;
greeting = “Hello!”; /* assigns “Hello!” to greeting */
```

```
list[int] nums;
nums = [1, 2, 3, 4, 5]; /* assigns sequence to nums */
```

3.1.5.2 Arithmetic

The arithmetic operators are used to remain consistent with the standard use of arithmetic operators used in any other constant. We implement the 4 main operators: addition, subtraction, multiplication, and division.

```
int x;
x = 10;

int y;
y = 5;

int z;
z = x + y; /* 15 */
z = x - y; /* 10 */
z = x * y; /* 50 */
z = x / y; /* 2 */
```

3.1.5.3 Logic

Logic operators are implemented as comparative checks between different values of variables and return booleans of whether these checks are true or false. The ones implemented are greater than, less than, greater than/equal to, less than/equal to, equal, and not equal. Below are examples of all of them in use.

```
int x;
x = 10;

int y;
y = 5;

bool z;
z = x > y; /* True */
z = x < y; /* False */
z = x >= y; /* True */
z = x <= y; /* False */
z = x <= y; /* False */
z = x != y; /* True */
```

3.2 Types and Variables

DRRTY supports three primitive types: `int` (number, 4 bytes), `bool` (true or false, 1 byte), and `float` (float number, 8 bytes). Variables cannot be initialized with a value and must be assigned a value in a later line.

3.2.1 `int`

An `int` represents an integer value that contains a sequence of one or more digits from 0-9. An example of a declaration is as follows:

```
int a;
a = 1;

int b;
b = 2;

int c
c = 3;
```

3.2.2 `float`

A `float` data type is a sequence of digits followed by a single decimal point and an optional sequence of digits. The regular expression is ['0' - '9']+ '.' ['0' - '9']* (['e' 'E'] ['+' '-']? ['0' - '9']+)?.

```
float num;
num = 11.777;
```

3.2.3 `bool`

The `bool` data type is a boolean that can be one of two values: `True` or `False`. When evaluating a boolean expression, the result will also be a boolean.

```
bool finished;
finished = True;
```

3.2.4 `str`

A string, `str`, is a sequence of alphanumeric characters enclosed in double quotes.

```
str helloWorld;
helloWorld = “Hello, World!”;
```

3.2.5 `list`

The `list` implementation in DRRTY is a mutable, ordered, singly-linked list. Lists allow users to store data of the same type within an iterable container. While and for loops are often used to iterate through lists. The start and end of a list are bound by square brackets [], and elements are separated by commas.

```
list[str] names;
names = [“Dylan”, “Richard”, “Rania”, “Trinity”];

/* sets index 1 to “Rania” */
names.set(1, “Rania”);

/* adds “Prof” to the end of names */
names.add(“Prof”);

/* gets value at index 1 and prints resulting string */
prints(names.get(1));
```

3.3 Control Flow

3.3.1 `if/else` Statements

Conditional statements in DRRTY are implemented through if/else checks. If the first conditional statement is evaluated to a boolean true, then the code that follows is completed. If the “if” block is false, the "else" block is executed. Each block of code corresponding to the conditional statements is bound by curly braces.

```
int x;
x = 11;

if(x < 100){
      prints(“You are now enrolled in PLT.”);
      print(x);
} else { /* if x > 100 code below runs */
      prints(“Sorry, the class is full.”);
}
```

3.3.2 `for` loops

For loops are used to iterate through a data structure, such as a list, and execute a block of code on each individual element of that structure. The start and end of a for loop are signified by curly braces.

```
list[int] nums;
nums = [1, 2, 3];

int i;
for(i = 0; i < nums.length(); i = i+1 ){
      print(nums.get(i));
}
```

3.3.3 `while` loops

While loops are used to execute a block of code until a boolean statement that is defined in the parentheses following the keyword "while" no longer evaluates to True. The beginning and end of the while loop code will be signified by single curly braces.

```
int i;
i = 0;

while(i < 10){
      nums.add(i);
      i = i + 1;
}
```

3.4 Output

3.4.1 `return`

`return` is a keyword used to send output from the function to the main program and signifies the complete execution of that function. It is analogous to the return keyword in Python and Java.

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

3.5 Built-In Functions

3.5.1 `list` functions

3.5.1.1 add(value: <list_type>)

`add` is a static function that appends an item to the end of a list. It only accepts data types that match the type of elements contained in the list to which it is appending. It does not return anything.

```
list[int] nums;
nums = [1, 2, 3];
nums.add(4); /* nums is now [1, 2, 3, 4] */
```

3.5.1.2 get(index: int)

`get` is a static function that accepts a key and returns its corresponding value. A KeyError will occur if a key is not present in the list and a user attempts to retrieve a value using the nonexistent key.

```
def int get2(){
      list[int] nums;
      nums = [1, 2, 3];
      return nums.get(1); /* returns 2 */
}
```

3.5.1.3 set(index: int, value: <list_type>)

`set` is a static function that replaces a value at a given index with another value inputted by the user. `set` takes in two inputs: an `int` representing the index, and a valid user input to replace the indexed value. It only accepts items of the same data type that exist in the list. It does not return anything.

```
list[int] nums;
nums = [1, 2, 3];
nums.set(0, 3); /* nums is now [3, 2, 3] */
nums.set(1, 3); /* nums is now [3, 3, 3] */
```

3.5.1.4 length(list: <list_type>)

`length` is a static function that returns the number of elements in a given list. The returned number is an `int`.

```
list[int] nums;
nums = [1, 2, 3];
nums.length(); /* prints out 3 */
```

3.5.2 HTML functions

We are implementing built-in HTML functions that will be able to create different HTML elements and set their values upon function call. Below are the different built-in functions.

3.5.2.1 createHTMLDocument(cssFile: String)

`createHTMLDocument` is a static function that creates the main body of an HTML document. The function requires a string input, representing a CSS file that the user might want to apply to the outputted *.html document. If there's no CSS file, an empty string is acceptable input. This function is required to be called in the main if the user wants to attach a CSS file for use in their program.

```
def int main(){
      /* printbig.css is a valid CSS file */
      createHTMLDocument("printbig.css");
      return 0;
}
```

3.5.2.2 createHTML(HTML: String)

`createHTML` is a static function that allows DRRTY users to input any HTML into the code. This function helps create nested HTML tags. The `createHTML` requires a string input, representing HTML code. The HTML written within the function will be output directly to the HTML file. In order for the HTML to come out as desired, the user would need to format the tags as needed.

```
def int main(){
      /* User inputs their own HTML */
      createHTML("<b><p>This is bold text!</p></b>");
      return 0;
}
```

3.5.2.3 createElement(HTMLTagName: String, innerHTML: String,cssClassName: String)

`createElement` is a static function that allows DRRTY users to easily create properly formatted HTML tags. The function takes in a string representing a tag name, a string representing a CSS class name, and what the user would like embedded in the specified tag

(inner HTML), which is also in the form of a string. Unlike in `createHTML`, `createElement` does not require the user to write their own HTML and format the tags for them. If the user does not have a predefined CSS file, then an empty string can be passed in to suggest null argument for the third parameter.

```
def int main(){
      /* Creates HTML document with CSS sheet attached */
      createHTMLDocument("printbig.css");

      /* Creates div tags with CSS class newel and the third string put between
      the div tags. */
      createElement("div","newel","This element will have a different color");
}
```

3.5.2.4 makeHeader(innerHTML: String)

`makeHeader` is a static function that allows DRRTY users to create a header tag. The function takes in a string representing what the user would like embedded in the header (innerHTML).

```
def int main(){
      /* Creates <h1>We are DRRTY!</h1> */
      makeHeader("We are DRRTY!");
}
```

3.5.2.5 makeText(innerHTML: String)

`makeText` is a static function that allows DRRTY users to create a paragraph tag – a general body text tag. The function takes in a string representing what the user would like embedded in the paragraph tag (innerHTML).

```
def int main(){
      /* Creates <p>Lorem ipsum is boring. Try cat ipsum next time.</p> */
      makeText("Lorem ipsum is boring. Try cat ipsum next time.");
}
```

3.5.2.6 makeImage(sourceName: String)

`makeImage` is a static function that allows DRRTY users to input an image into an HTML file. The function takes in a string representing the local or public path to the image they want to be displayed.

```
def int main(){
      /* Creates <img src="/drrty/drrty.jpeg"> */
```

```
      makeImage("/drrty/drrty.jpeg");
}
```

3.5.2.6 makeInput(className: String)

`makeInput` is a static function that allows DRRTY users to attach an input field on an HTML page that users can interact with on the generated HTML page. The function takes in a string representing a CSS class name.

```
def int main(){
      /* Creates <input class="class1"> */
      makeInput("class1");
}
```

3.6 Syntax and Style

- Indentation is required for style purposes and readability but is not required syntactically.
- Maximum 80 characters per line.
- All opening symbols must have a corresponding closing symbol: (), {}, [].
- All lines not ending in opening/closing symbols must end in a semicolon.

4 Project Plan

4.1 Planning

For our DRRTY project, the group aimed to meet in person or over Zoom at least once a week throughout the semester to check in and work on the project together. During these weekly meetings, we would work on the next deadline, resolve any confusion together, and split up tasks accordingly to ensure that we were able to complete what we needed to in a timely manner. We also reviewed lectures and any changes made individually as a cohort to ensure that each member understood how the program was modified and how to achieve the same result themselves.

Team communication occurred over text and Zoom meetings outside of our weekly in-person meetings. Meetings and consultations with Professor Edwards occurred after classes on Friday, over Zoom, or if necessary over email.

4.2 Specification

The goal of DRRTY is to create frontend engineering for backend engineers. We wanted to bring elements of HTML into the Python language, similar to Jenga. We implemented some basic elements of C/Java – semicolons, variable declarations, brackets – and mixed it with some Python elements such as lists. DRRTY incorporates the basic functionality of creating HTML elements with built-in functions as well as display output from user-generated functions in an HTML format. Initially, we planned to implement the HTML format by creating a new HTML type, but opted for our language to encapsulate the output in HTML tags and format the output as regular text in a webpage through built in HTML functions. After turning in our LRM, adjustments were made to clarify syntax and expand on built-in functions.

4.3 Development

The project started with a couple of meetings finalizing what we wanted our language to look like: fine-tuning what the purpose of our language was and a sample program that followed our logic. From here, we focused on defining the syntax and clarifying any holes that we came across in our design. The development of our LRM helped guide us through the construction of our compiler starting with the scanner, parser, and AST. Our Hello World program had a basic sast.ml file and codegen.ml file, enough to print out “Hello World” in an HTML header tag. Once the compiler was up and running successfully, the final weeks creating and linking our DRRTY C library to output a functional binary executable, building our HTML library and its functions that are used to generate HTML, and fleshing out components we included in our original LRM.

4.4 Testing

As we implement each of the data types and built in functions to generate html, we create tests to check the implementation of each individual feature. With two types of tests, successful and failures, we cover a broad array of error checking for booth syntax and semantics, as well as logic and functionality of the features we implement. All of these amount to every function, whether it be implemented through our c-libraries or simply arithmetic functions, being tested in an assortment of example programs that a DRRTY user might write.

4.5 Roles and Responsibilities

Team Member	**Role**
Dylan Bamirny	Manager

Richard Lopez	Language Guru
Rania Alshafie	Tester
Trinity Sazo	System Architect

Responsibilities and tasks were split amongst more than one person more times than not, and many times we used a Visual Studio Code extension called Live Share to work on the project simultaneously. Below shows a breakdown of the tasks, but keep in mind that every member has assisted in the development of each.

HTML	Richard, Dylan
Lists	Dylan, Richard
Strings	Trinity, Rania
Print Functions	Dylan, Richard, Rania, Trinity
For/While loops	Dylan, Richard
If/else	Dylan, Richard
Built-In Functions	Dylan, Richard, Rania, Trinity
Testing	Rania, Trinity

4.6 Project Timeline / Log

Date	**Milestone**
9/29	Initial Meeting + DRRTY Proposal Idea
10/7	DRRTY Proposal Finalized
10/15	Professor Edwards Meeting
10/20	LRM Draft Started
10/22	Professor Edwards Meeting
10/27	Addressed Edward’s Notes + Continue LRM Draft + Gumbo exploration
10/29	Continue Gumbo Exploration

10/31	Completed LRM
11/3	Scanner/Parser Work
11/5	Professor Edwards Meeting
11/10	Add Strings to compiler + “Hello World”
11/17	Debugging
12/7	Professor Edwards Meeting + Started List type integration
12/19	Integrated List and HTML C libraries
12/20	Finalized List and HTML functions
12/22	Final Presentation, final project report, final compiler

As we all worked on VSCode Live Share, many of our files have been worked on by the group as a whole.

4.7 Software Development Environment

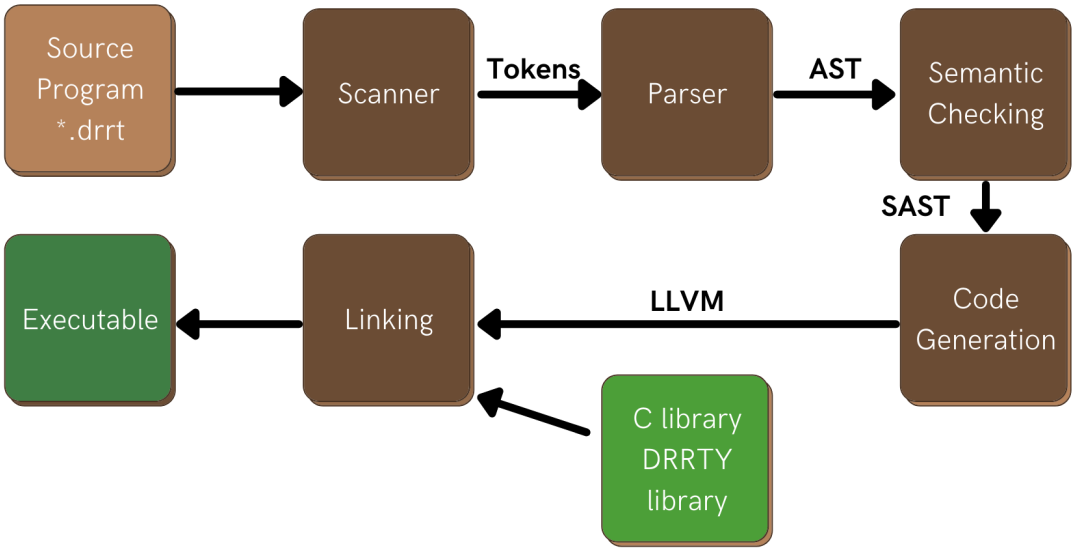
OCaml with the LLVM library was the main programming language that we used to build DRRTY. The OCamlyacc and OCamllex extensions were used for compiling the scanner and parser front end. We used the micro-C language provided by Professor Stephen Edwards to build some of the built-in functions and GCC was used to compile and link the C files. Each member used Visual Studio Code with the OCaml Platform and Live Share extensions.

4.8 Programming Style Guide

While we programmed, we generally followed the formatting style shown to us in class and required of us from homework 1. We kept to OCaml’s editing and formatting style throughout all of our files and ensured our indentation made the code not only readable but also functional. We broke up the code based on what they implemented and commented on our files based on the functionality of what we added. Naming conventions stuck to using underscores (similar to Python).

5 Architectural Design

5.1 Block Diagram



5.2 Scanner

Our DRRTY scanner takes in the DRRTY source file *.drrt and tokenizes the stream of ASCII characters into keywords, identifiers, operators, and all of our other language components. Spaces, newlines, and any character inside comment blocks will be ignored and removed during this stage. Any character not identified by our scanner – an illegal character – will throw a lexing error. These tokens are then sent to our parser file.

5.3 Parser and Semantic Checker

The parser evaluates the token stream from the scanner following DRRTY's grammar as specified in the language reference manual. The parser uses the ast.ml file to generate an Abstract Syntax Tree following the rules in our grammar. Moreover, our parser is derived from micro-C and the file has been built upon to implement strings and lists. If any mismatching is detected (syntax error), the parser error will be thrown.

Semantic checking takes place in the semant.ml file and takes the Abstract Syntax Tree to return a syntactically checked AST. Each identifier, each type of any given expression, and each type of

any statements in functions are all checked for consistency in their typing. Additionally, semantic checking ensures variables are in the proper scope, functions are not redefined, and more. This is especially useful in our implementation of lists, which can only hold elements of the same type. In reference to our list library and html library, semantic checking ensures that types and arguments are also specified for the use of those libraries. If a type error occurs, semant.ml will output an error message. Otherwise, a SAST will be passed into the codegen.ml file.

5.4 Code Generation

The DRRTY code generator takes in the SAST and constructs LLVM code for a DRRTY binary executable. The code generator is where all functions are defined both in the C libraries and from micro-C. The codegen.ml file generates the built-in HTML functions and explicitly links our written C libraries with the semantically checked AST from the scanner, parser, and semant.

5.5 C Libraries

To implement HTML functions and list functions, we created two C files to provide HTML generation and list manipulation to our users.

6 Test Plan

Tests were created iteratively during development in order to ensure that the functionalities that we were implementing were working properly. We created two types of tests: ones that were intended to fail and ones that were intended to succeed. There is a naming convention to distinguish between the two types of tests and make sure they are compared against the right type of file in order to test correctly.

The first type of test, whose filenames begin with “test”, is meant to succeed. Running this type of test would produce 2 executable files: .out and a .html file. The .out file was used to check against the .drrt file in order to make sure that the .drrt file was outputting correctly. It required at least one successful run to produce the .out file and use it for further testing. The .html file was used to ensure that the HTML output of the programs is functioning correctly, where we would open it up in the browser.

The second type of test whose filenames begin with “fail” was meant to fail. Running this test would not produce the two executable files because they would result in errors such as a parsing error that would halt the compilation process, preventing the production of the executables. Instead, the main file is produced as a .err suffix, where it would log the error that was found when running that specific test. These tests also need at least one “successful” run, where the test does fail, in order to use the .err file to compare future errors.

7 Lessons Learned

7.1 Dylan Bamirny

Among all the computer science courses I have taken at Columbia, Programming Languages and Translators has been one of the most challenging and also one of the most rewarding. Developing a sophisticated programming language in less than a semester was no easy feat. I leave this course and project not only as a more seasoned developer, but also as someone with a deeper appreciation for the beauty and complexity of computer programming. Besides the steep learning curve of the subject material, one of the most challenging aspects of this project was organizing and coordinating a four-person team to work on a code repository with co-dependent modules. Initially, we struggled to arrange meetings that could accommodate everyone's busy schedules; on some weeks, we were all only available to meet in person for a single hour. However, over the course of the semester, we developed various strategies to maintain progress towards completing our programming language: delegating work according to language features as opposed to language files, incorporating version control to merge independent workstreams, and regularly checking in on each other by virtual means. And, when we could meet in person, we prepared rigorous agendas to maximize the value of the time spent working together. I am glad—and also quite lucky—to have worked with team members who were good communicators, and that made all the difference.

7.2 Richard Lopez

Some of the most important lessons I learned during the completion of the project include the importance of communicating with team members, being clear about current workloads and capacity, and the importance of using a version control system. Communication amongst our team was positive and always consistent. During moments in which I was having difficulty implementing a specific language feature, I was able to openly communicate this with my team and they would provide assistance. This form of open and honest communication led to everyone having an awareness of what was being currently worked on, what features were completed, and which features were causing the most headache. Throughout the project, I also learned the importance of breaking tasks down into smaller subtasks. While the task of writing a compiler and an external C-library is daunting, breaking the overarching goal into smaller goals and tasks made the work much more manageable. Specifically, when we had one working feature, we were able to build off of the previous work and modify it so that future features would be built on top of working and functional code. This made it much easier to understand the cause of our errors. Finally, using a version control system allowed our team to share and merge code much more efficiently by the time we started to use it. It also allowed us to make errors on a separate branch that was independent of a working compiler. This approach made making errors much safer as the errors would not affect the working version of our language.

7.3 Rania Alshafie

Some of the lessons I learned throughout the course of this project was the importance of being proactive, communicating, and asking for help in a group environment. I think one of the main things I wished we had done differently was come to group meetings with better preparation or understanding of the content of the files and the code we want to implement, so being proactive in learning these things would have been a huge asset and saved a lot of time that we would have been able to put towards implementing other features or even figuring out Gumbo. Regardless, I learned that communicating both what I was able to accomplish and what I wasn't able to do by a certain deadline was important in avoiding overlap and staying on track. When I was struggling with a specific function (for example with how the testing files were going to work in terms of comparing the .out and .drrt files), asking for help was so important and saved so much time. Discussing it with the rest of my team clarified our goals and made it easy to move forward and build on the tests we're creating. It allowed us to work productively in our live share sessions, where we coded together in our sessions in order to maximize everyone's understanding. Overall, I learned so much about compilers, and being part of the building process has helped me gain so many more skills than I had anticipated. Although the project was difficult, it was definitely worth all the information I learned.

7.4 Trinity Sazo

As this project comes to a close, I find myself thankful that the group and I have been able to both complete our tasks but also how we've been able to stay in such high spirits the whole time. There have been so many lessons throughout the semester from our time management, to our communication, to our coding process and more. Looking back on how much we have accomplished, I find all of it quite rewarding. The project is not an easy one, especially with the limited time we were given this semester due to the class starting a couple weeks late and having to jump into the project with little to no background knowledge on how to code in OCaml, let alone create a compiler from scratch. For me, I definitely would have spent more time in office hours with both Professor Edwards and the TAs. I found that the longer I avoided asking for help because I did not know what questions to ask, the bigger a hole I dug for myself. However, being with this group in particular made me feel comfortable in my confusion as we would always be there for each other in catching everyone up to speed. One thing I wish the group would have done differently would be to meet more frequently but for less time using Zoom and other means available to us. Meeting up more than once a week would have ensured that we planned better for meetings with Edwards on top of continuously moving the project forward. While all of our members were present and consistent, we still fell behind due to being confused, which is where reaching out for help would have been extremely beneficial in terms of time management for the project. Regardless of any struggles, I am amazed at how much I've been able to take away from the class and more deeply understand the intricacies of building a language from the scanner to the parser to the semantic checking to the code generation.

8 Appendix

8.1 scanner.mll

```
(* Ocamllex scanner for DRRTY *)

{
 open Drrtyparse

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

let digit = ['0' - '9']
let digits = digit+
let ascii = ([' '-'!' '#'-'[' ']'-'~'])
let escape = '\\' ['\\' ''' '"' 'n' 'r' 't']
let string = '"' ( (ascii | escape)* as s) '"'

rule token = parse
 [' ' '\t' '\r' '\n'] { token lexbuf } (* Whitespace *)
| "/*"     { comment lexbuf }           (* Comments *)
| '('      { LPAREN }
| ')'      { RPAREN }
| '{'      { LBRACE }
| '}'      { RBRACE }
| '['      { LBRACKET }
| ']'      { RBRACKET }
| ';'      { SEMI }
| ':'      { COLON }
| ','      { COMMA }
| '+'      { PLUS }
| '-'      { MINUS }
| '*'      { TIMES }
| '/'      { DIVIDE }
| '='      { ASSIGN }
| '\n'     { EOL }
```

```
| "=="     { EQ }
| "!="     { NEQ }
| "<"      { LT }
| "<="     { LEQ }
| ">"      { GT }
| ">="     { GEQ }
| "&&"     { AND }
| "||"     { OR }
| "!"      { NOT }
| "def"    { FUNCTION }
| "end"    { END }
| "if"     { IF }
| "else"   { ELSE }
| "for"    { FOR }
| "while"  { WHILE }
| "return" { RETURN }
| "int"    { INT }
| "bool"   { BOOL }
| "float"  { FLOAT }
| "void"   { VOID }
| "str"    { STRING }
| ".get"    { LISTGET}
| ".set"    { LISTSET }
| ".add"    { LISTADD }
| ".length" { LISTLENGTH }
| "list"    { LIST }
| "True"   { BLIT(true)  }
| "False"  { BLIT(false) }
| digits as lxm { LITERAL(int_of_string lxm) }
| digits '.'  digit* ( ['e' 'E'] ['+' '-']? digits )? as lxm { FLIT(lxm) }
| ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']*     as lxm { ID(lxm) }
| string            { STRING_LITERAL( (unescape s) ) }
| eof { EOF }
| _ as char { raise (Failure("illegal character " ^ Char.escaped char)) }

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

8.2 drrtyparse.mly

```
/* Ocamlyacc parser for drrty */

%{
open Ast
%}

%token SEMI COLON LPAREN RPAREN LBRACE RBRACE LBRACKET RBRACKET COMMA
%token PLUS MINUS TIMES DIVIDE ASSIGN
%token NOT EQ NEQ LT LEQ GT GEQ AND OR EOL
%token FUNCTION END RETURN IF ELSE FOR WHILE INT BOOL FLOAT LIST VOID STRING
%token LIST LISTLENGTH LISTGET LISTSET LISTADD
%token <int> LITERAL
%token <bool> BLIT
%token <string> ID FLIT STRING_LITERAL
%token EOF

%start program
%type <Ast.program> program

%nonassoc NOELSE
%nonassoc ELSE
%right ASSIGN
%left OR
%left AND
%left EQ NEQ
%left LT GT LEQ GEQ
%left PLUS MINUS
%left TIMES DIVIDE
%right NOT
%right LISTLENGTH LISTGET LISTSET LISTADD
%left LBRACKET RBRACKET
```

```
%%

program:
 decls EOF { $1 }

decls:
  /* nothing */ { ([], [])               }
| decls vdecl { (($2 :: fst $1), snd $1) }
| decls fdecl { (fst $1, ($2 :: snd $1)) }

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

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

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

typ:
   INT   { Int }
 | BOOL  { Bool }
 | FLOAT { Float }
 | VOID  { Void }
 | STRING { String }
 | LIST LBRACKET typ RBRACKET { List($3) }


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

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

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

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



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

expr:
   LITERAL          { Literal($1)            }
 | FLIT             { Fliteral($1)           }
 | BLIT             { BoolLit($1)            }
 | STRING_LITERAL   { StringLit($1) }
 | ID               { Id($1)                 }
 | expr PLUS   expr { Binop($1, Add,   $3)   }
 | expr MINUS  expr { Binop($1, Sub,   $3)   }
 | expr TIMES  expr { Binop($1, Mult,  $3)   }
 | expr DIVIDE expr { Binop($1, Div,   $3)   }
 | expr EQ     expr { Binop($1, Equal, $3)   }
```

```
 | expr NEQ    expr { Binop($1, Neq,    $3)   }
 | expr LT     expr { Binop($1, Less,   $3)   }
 | expr LEQ    expr { Binop($1, Leq,    $3)   }
 | expr GT     expr { Binop($1, Greater, $3)  }
 | expr GEQ    expr { Binop($1, Geq,    $3)   }
 | expr AND    expr { Binop($1, And,    $3)   }
 | expr OR     expr { Binop($1, Or,     $3)   }
 | MINUS expr %prec NOT      { Unop(Neg, $2) }
 | NOT expr                  { Unop(Not, $2) }
 | ID ASSIGN expr            { Assign($1, $3) }
 | ID LPAREN args_opt RPAREN { Call($1, $3)  }
 | LPAREN expr RPAREN { $2 }

 | LBRACKET args_opt RBRACKET                    { ListLit($2) }
 | expr LISTLENGTH LPAREN RPAREN                 { ListLength($1) }
 | expr LISTGET LPAREN expr RPAREN               { ListGet($1, $4) }
 | expr LISTSET LPAREN expr COMMA expr RPAREN    { ListSet($1, $4, $6) }
 | expr LISTADD LPAREN expr RPAREN               { ListAdd($1, $4) }

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

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

8.3 ast.ml

```
(* Abstract Syntax Tree *)
```

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

type uop = Neg | Not

type typ = Int | Bool | Float | Void | String | List of typ

type bind = typ * string

type expr =
   Literal of int
 | Fliteral of string
 | BoolLit of bool
 | Id of string
 | StringLit of string
 | Binop of expr * op * expr
 | Unop of uop * expr
 | Assign of string * expr
 | Call of string * expr list
 | ListLength of expr
 | ListGet of expr * expr
 | ListSet of expr * expr * expr
 | ListAdd of expr * expr
 | ListLit of expr list
 | Noexpr

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

type func_decl = {
   typ : typ;
   fname : string;
   formals : bind list;
```

```
    locals : bind list;
    body : stmt list;
  }

type program = bind list * func_decl list

(* Pretty-printing functions *)

let string_of_op = function
    Add -> "+"
  | Sub -> "-"
  | Mult -> "*"
  | Div -> "/"
  | Equal -> "=="
  | Neq -> "!="
  | Less -> "<"
  | Leq -> "<="
  | Greater -> ">"
  | Geq -> ">="
  | And -> "&&"
  | Or -> "||"

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

let rec string_of_expr = function
    Literal(l) -> string_of_int l
  | Fliteral(l) -> l
  | StringLit s -> "\"" ^ s ^ "\""
  | Id(s) -> s
  | BoolLit(true) -> "True"
  | BoolLit(false) -> "False"
  | Binop(e1, o, e2) -> string_of_expr e1 ^ " " ^ string_of_op o ^ " " ^
string_of_expr e2
  | Unop(o, e) -> string_of_uop o ^ string_of_expr e
  | Assign(v, e) -> v ^ " = " ^ string_of_expr e
```

```
 | Call(f, el) -> f ^ "(" ^ String.concat ", " (List.map string_of_expr el) ^
")"
 | ListLit(l) -> "[" ^ String.concat "," (List.map string_of_expr l) ^ "]"
 | ListGet(l, idx) -> string_of_expr l ^ ".get(" ^ string_of_expr idx ^ ")"
 | ListLength(l) -> string_of_expr l ^ ".length()"
 | ListAdd(l, e) -> string_of_expr l ^ ".add(" ^ string_of_expr e ^ ")"
 | ListSet(l, idx, e) -> string_of_expr l ^ ".set(" ^ string_of_expr idx ^ ","
^ string_of_expr e ^ ")"
 | Noexpr -> ""

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

let rec string_of_typ = function
   Int -> "int"
 | Bool -> "bool"
 | Float -> "float"
 | Void -> "void"
 | String -> "str"
 | List(t) -> "list[" ^ string_of_typ t ^ "]"

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

let string_of_fdecl fdecl =
 "def " ^ string_of_typ fdecl.typ ^ " " ^
 fdecl.fname ^ "(" ^ String.concat ", " (List.map snd fdecl.formals) ^
 ")\n:\n" ^
 String.concat "" (List.map string_of_vdecl fdecl.locals) ^
```

```
 String.concat "" (List.map string_of_stmt fdecl.body) ^
 "\n" ^ "end"

let string_of_program (vars, funcs) =
 String.concat "" (List.map string_of_vdecl vars) ^ "\n" ^
 String.concat "\n" (List.map string_of_fdecl funcs)
```

8.4 sast.ml

```
(* Semantically-checked Abstract Syntax Tree *)

open Ast

type sexpr = typ * sx
and sx =
   SLiteral of int
 | SFliteral of string
 | SBoolLit of bool
 | SId of string
 | SStringLit of string
 | SBinop of sexpr * op * sexpr
 | SUnop of uop * sexpr
 | SAssign of string * sexpr
 | SCall of string * sexpr list
 | SListLit of sexpr list
 | SListLength of sexpr
 | SListGet of sexpr * sexpr
 | SListSet of sexpr * sexpr * sexpr
 | SListAdd of sexpr * sexpr
 | SNoexpr

type sstmt =
   SBlock of sstmt list
 | SExpr of sexpr
 | SReturn of sexpr
 | SIf of sexpr * sstmt * sstmt
```

```
 | SFor of sexpr * sexpr * sexpr * sstmt
 | SWhile of sexpr * sstmt

type sfunc_decl = {
   styp : typ;
   sfname : string;
   sformals : bind list;
   slocals : bind list;
   sbody : sstmt list;
 }

type sprogram = bind list * sfunc_decl list

(* Pretty-printing functions *)

let rec string_of_sexpr (t, e) =
 "(" ^ string_of_typ t ^ " : " ^ (match e with
   SLiteral(l) -> string_of_int l
 | SBoolLit(true) -> "True"
 | SBoolLit(false) -> "False"
 | SFliteral(l) -> l
 | SStringLit(s) -> s
 | SListLit(l) -> "[" ^ String.concat "," (List.map string_of_sexpr l) ^ "]"
 | SListLength(l) -> string_of_sexpr l ^ ".length()"
 | SListGet(l, idx) -> string_of_sexpr l ^ ".get(" ^ string_of_sexpr idx ^ ")"
 | SListSet(l, idx, e) -> string_of_sexpr l ^ ".set(" ^ string_of_sexpr idx ^
"," ^ string_of_sexpr e ^ ")"
 | SListAdd(l, e) -> string_of_sexpr l ^ ".add(" ^ string_of_sexpr e ^ ")"
 | SId(s) -> s
 | SBinop(e1, o, e2) ->
     string_of_sexpr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_sexpr e2
 | SUnop(o, e) -> string_of_uop o ^ string_of_sexpr e
 | SAssign(v, e) -> v ^ " = " ^ string_of_sexpr e
 | SCall(f, el) ->
     f ^ "(" ^ String.concat ", " (List.map string_of_sexpr el) ^ ")"
 | SNoexpr -> ""
       ) ^ ")"
```

```
let rec string_of_sstmt = function
   SBlock(stmts) ->
     "{\n" ^ String.concat "" (List.map string_of_sstmt stmts) ^ "}\n"
 | SExpr(expr) -> string_of_sexpr expr ^ ";\n";
 | SReturn(expr) -> "Return " ^ string_of_sexpr expr ^ ";\n";
 | SIf(e, s, SBlock([])) -> "if (" ^ string_of_sexpr e ^ ")\n" ^
string_of_sstmt s
 | SIf(e, s1, s2) ->  "if (" ^ string_of_sexpr e ^ ")\n" ^ string_of_sstmt s1 ^
"else\n" ^ string_of_sstmt s2
 | SFor(e1, e2, e3, s) -> "for (" ^ string_of_sexpr e1  ^ " ; " ^
string_of_sexpr e2 ^ " ; " ^ string_of_sexpr e3  ^ ") " ^ string_of_sstmt s
 | SWhile(e, s) -> "while (" ^ string_of_sexpr e ^ ") " ^ string_of_sstmt s

let string_of_sfdecl fdecl =
 "def " ^ string_of_typ fdecl.styp ^ " " ^
 fdecl.sfname ^ "(" ^ String.concat ", " (List.map snd fdecl.sformals) ^
 ")\n:\n" ^
 String.concat "" (List.map string_of_vdecl fdecl.slocals) ^
 String.concat "" (List.map string_of_sstmt fdecl.sbody) ^
 "\n"

let string_of_sprogram (vars, funcs) =
 String.concat "" (List.map string_of_vdecl vars) ^ "\n" ^
 String.concat "\n" (List.map string_of_sfdecl funcs)
```

8.5 semant.ml

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

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 global variable, then check each function *)

let get_typ(t, _) = t
 let first_elem (l) = match l with
     [] -> Void
   | e :: _ -> get_typ(e)

let check_list_typ m =
 let (t, _) = m in
 match t with
 List(ty) -> ty
 | _ -> raise (Failure ("Invalid list type: " ^ string_of_typ t))

(* Checks whether sexpr is an acceptable list element*)
let check_elem_typ = function
       (Void,_) -> raise(Failure("Invalid element type"))
 | _ -> ()

let check_typ (e, t) =
 if e = t then () else raise (Failure ("Expression and type incompatible"))

let check (globals, functions) =

 (* Verify a list of bindings has no none types or duplicate names *)
 let check_binds (kind : string) (binds : bind list) =
   List.iter (function
 (Void, b) -> raise (Failure ("illegal void " ^ kind ^ " " ^ b))
     | _ -> ()) binds;
   let rec dups = function
       [] -> ()
     | ((_,n1) :: (_,n2) :: _) when n1 = n2 ->
   raise (Failure ("duplicate " ^ kind ^ " " ^ n1))
     | _ :: t -> dups t
   in dups (List.sort (fun (_,a) (_,b) -> compare a b) binds)
```

```
in

let check_list_binds (binds : sexpr list) =
  List.iter check_elem_typ binds;

  let rec typ_match = function
    [] -> ()
  | ((t1,_) :: (t2,_) :: _) when t1 != t2 ->
          raise (Failure ("Type of elements in list incosistent"))
  | _ :: t -> typ_match t

  in typ_match (List.sort (fun (a,_) (b,_) -> compare a b) binds);

in

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

check_binds "global" globals;

(**** Check functions ****)

(* Collect function declarations for built-in functions: no bodies *)
let built_in_decls =
  let add_bind map (name, ty) = StringMap.add name {
    typ = Void;
    fname = name;
    formals = [(ty, "x")];
    locals = []; body = [] } map
  in List.fold_left add_bind StringMap.empty [ ("print", Int);
                               ("printb", Bool);
                               ("printf", Float);
                               ("prints", String);
                               ("createHTMLDocument", String);
                               ("createHTML", String);
                               ("printl", List(Int));
                               ("printl", List(Float));
                               ("printl", List(String));
                               ("makeHeader", String);
```

```
                                ("makeText", String);
                                ("makeImage", String);
                                ("makeInput", String)]
in

let built_in_decls =
  StringMap.add "createElement" {
    typ = String;
    fname = "createElement";
    formals = [(String, "str1"); (String, "str2");(String, "str3")];
    locals = [];
    body = [] } built_in_decls
 in

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

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

(* Return a function from our symbol table *)
let find_func s =
  try StringMap.find s function_decls
  with Not_found -> raise (Failure ("unrecognized function " ^ s))
in

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

```
 let check_function func =
   (* Make sure no formals or locals are none or duplicates *)
   check_binds "formal" func.formals;
   check_binds "local" func.locals;

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

   (* Build local symbol table of variables for this function *)
   let symbols = List.fold_left (fun m (ty, name) -> StringMap.add name ty m)
                 StringMap.empty (globals @ func.formals @ func.locals )
   in

   (* Return a variable from our local symbol table *)
   let type_of_identifier s =
     try StringMap.find s symbols
     with Not_found -> raise (Failure ("undeclared identifier " ^ s))
   in

   (* Return a semantically-checked expression, i.e., with a type *)
   let rec expr =
     let check_list m =
       let (t, _) = expr m in
       match t with
       List(_) -> ()
       | _ -> raise (Failure ("Invalid list, received: " ^ string_of_typ t)) in

     let check_list_type m =
       let (t, _) = expr m in
       match t with
        List(ty) -> ty
       |_ -> raise (Failure ("Invalid list type, received: " ^ string_of_typ
t)) in

     function
```

```
  Literal  l -> (Int, SLiteral l)
| Fliteral l -> (Float, SFliteral l)
| BoolLit l  -> (Bool, SBoolLit l)
| Noexpr     -> (Void, SNoexpr)
| StringLit s -> (String, SStringLit s)
| Id s       -> (type_of_identifier s, SId s)
| Assign(var, e) as ex ->
    let lt = type_of_identifier var
    and (rt, e') = expr e in
    let err = "illegal assignment " ^ string_of_typ lt ^ " = " ^
      string_of_typ rt ^ " in " ^ string_of_expr ex
    in (check_assign lt rt err, SAssign(var, (rt, e')))
| Unop(op, e) as ex ->
    let (t, e') = expr e in
    let ty = match op with
      Neg when t = Int || t = Float -> t
    | Not when t = Bool -> Bool
    | _ -> raise (Failure ("illegal unary operator " ^
                           string_of_uop op ^ string_of_typ t ^
                           " in " ^ string_of_expr ex))
    in (ty, SUnop(op, (t, e')))
| Binop(e1, op, e2) as e ->
    let (t1, e1') = expr e1
    and (t2, e2') = expr e2 in
    (* All binary operators require operands of the same type *)
    let same = t1 = t2 in
    (* Determine expression type based on operator and operand types *)
    let ty = match op with
        Add | Sub | Mult | Div when same && t1 = Int   -> Int
      | Add | Sub | Mult | Div when same && t1 = Float -> Float
      | Add                          when same && t1 = String -> String
    | Equal | Neq            when same               -> Bool
    | Less | Leq | Greater | Geq
               when same && (t1 = Int || t1 = Float) -> Bool
    | And | Or when same && t1 = Bool -> Bool
    | _ -> raise (
      Failure ("illegal binary operator " ^
          string_of_typ t1 ^ " " ^ string_of_op op ^ " " ^
```

```
            string_of_typ t2 ^ " in " ^ string_of_expr e))
        in (ty, SBinop((t1, e1'), op, (t2, e2')))
  | ListLength l -> check_list(l);
    (Int, SListLength(expr l))

  | ListGet(l, idx) -> check_list(l);
    check_typ(get_typ(expr idx), Int);
    (check_list_type(l), SListGet(expr l, expr idx))

  | ListSet(l, idx, e) -> check_list(l);
    check_typ(get_typ(expr idx), Int);
    check_elem_typ(expr e);
    (check_list_type(l), SListSet (expr l, expr idx, expr e))

  | ListAdd(l, e) -> check_list(l);
    check_elem_typ(expr e);
    (Void, SListAdd(expr l, expr e))

  | ListLit l  -> check_list_binds (List.map expr l);
    (List(first_elem(List.map expr l)), SListLit (List.map expr l))

  | Call(fname, args) as call ->
      let fd = find_func fname in
      let param_length = List.length fd.formals in
      if List.length args != param_length then
        raise (Failure ("expecting " ^ string_of_int param_length ^
                        " arguments in " ^ string_of_expr call))
      else let check_call (ft, _) e =
        let (et, e') = expr e in
        let err = "illegal argument found " ^ string_of_typ et ^
          " expected " ^ string_of_typ ft ^ " in " ^ string_of_expr e
        in (check_assign ft et err, e')
      in
      let args' = List.map2 check_call fd.formals args
      in (fd.typ, SCall(fname, args'))
in

let check_bool_expr e =
```

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

  in

  (* Return a semantically-checked statement i.e. containing sexprs *)
  let rec check_stmt = function
      Expr e -> SExpr (expr e)
    | If(p, b1, b2) -> SIf(check_bool_expr p, check_stmt b1, check_stmt b2)
    | For(e1, e2, e3, st) -> SFor(expr e1, check_bool_expr e2, expr e3,
check_stmt st)
    | While(p, s) -> SWhile(check_bool_expr p, check_stmt s)
    | Return e -> let (t, e') = expr e in
      if t = func.typ then SReturn (t, e')
      else raise (
        Failure ("output gives " ^ string_of_typ t ^ " expected " ^
     string_of_typ func.typ ^ " in " ^ string_of_expr e))

    (* A block is correct if each statement is correct and nothing
       follows any Return statement.  Nested blocks are flattened. *)
    | Block sl ->
        let rec check_stmt_list = function
            [Return _ as s] -> [check_stmt s]
          | Return _ :: _   -> raise (Failure "nothing may follow a output")
          | Block sl :: ss  -> check_stmt_list (sl @ ss) (* Flatten blocks *)
          | s :: ss         -> check_stmt s :: check_stmt_list ss
          | []              -> []
        in SBlock(check_stmt_list sl)

  in (* body of check_function *)
  { styp = func.typ;
    sfname = func.fname;
    sformals = func.formals;
    slocals  = func.locals;
    sbody = match check_stmt (Block func.body) with
 SBlock(sl) -> sl
    | _ -> raise (Failure ("internal error: block didn't become a block?"))
```

```
  }
 in (globals, List.map check_function functions)
```

8.6 codegen.ml

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

LLVM tutorial: Make sure to read the OCaml version of the tutorial

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

Detailed documentation on the OCaml LLVM library:

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

*)

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

module StringMap = Map.Make(String)

let get_typ(t, _) = t

(* translate : Sast.program -> Llvm.module *)
let translate (globals, functions) =
 let context    = L.global_context () in

 (* Load library *)
 let llmem_lib = L.MemoryBuffer.of_file "listlibrary.bc" in
 let llm_lib = Llvm_bitreader.parse_bitcode context llmem_lib in
```

```
 (* Create the LLVM compilation module into which
    we will generate code *)
 let the_module = L.create_module context "DRRTY" in

 (* Get types from the context *)
 let i32_t      = L.i32_type    context
 and i8_t       = L.i8_type     context
 and i1_t       = L.i1_type     context
 and float_t    = L.double_type context
 and string_t   = L.pointer_type (L.i8_type context)
 and void_t     = L.void_type   context
 and void_ptr_t = L.pointer_type (L.i8_type context)
 and list_t     = L.pointer_type (match L.type_by_name llm_lib "struct.list"
with
 None -> raise (Failure "struct list not found in library")
| Some t -> t)

 in

 (* Return the LLVM type for a DRRTY type *)
 let ltype_of_typ = function
     A.Int   -> i32_t
   | A.Bool  -> i1_t
   | A.Float -> float_t
   | A.Void  -> void_t
   | A.String -> string_t
   | A.List _ -> list_t
  in

 let str_of_typ t = match t with
     A.Int -> "int"
   | A.Bool -> "bool"
   | A.Float -> "float"
   | A.String -> "str"
   | _ -> raise (Failure "Invalid type")

 in
```

```
let check_list_typ m =
  let (t, _) = m in
  match t with
  List(ty) -> ty
  | _ -> raise (Failure ("Invalid list type: " ^ str_of_typ t))
 in

(* Create a map of global variables after creating each *)
let global_vars : L.llvalue StringMap.t =
  let global_var m (t, n) =
    let init = match t with
        A.Float -> L.const_float (ltype_of_typ t) 0.0
      | _ -> L.const_int (ltype_of_typ t) 0
    in StringMap.add n (L.define_global n init the_module) m in
  List.fold_left global_var StringMap.empty globals in

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

(* C-Library HTML Functions *)

let createHTMLDocument_t : L.lltype =
  L.function_type i32_t [| string_t |] in
let createHTMLDocument_func : L.llvalue =
  L.declare_function "createHTMLDocument" createHTMLDocument_t the_module in
 let createElement_t : L.lltype =
  L.function_type i32_t [| string_t; string_t; string_t |] in
let createElement_func : L.llvalue =
    L.declare_function "createElement" createElement_t the_module in

let createHTML_t : L.lltype =
  L.function_type i32_t [| string_t |] in
let createHTML_func : L.llvalue =
    L.declare_function "createHTML" createHTML_t the_module in

let makeHeader_t : L.lltype =
```

```
  L.function_type i32_t [| string_t |] in
let makeHeader_func : L.llvalue =
    L.declare_function "makeHeader" makeHeader_t the_module in

let makeText_t : L.lltype =
  L.function_type i32_t [| string_t |] in
let makeText_func : L.llvalue =
    L.declare_function "makeText" makeText_t the_module in

let makeImage_t : L.lltype =
  L.function_type i32_t [| string_t |] in
let makeImage_func : L.llvalue =
    L.declare_function "makeImage" makeImage_t the_module in

let makeInput_t : L.lltype =
  L.function_type i32_t [| string_t |] in
let makeInput_func : L.llvalue =
    L.declare_function "makeInput" makeInput_t the_module in

(* List Functions *)
let list_init_t = L.function_type list_t [||] in
let list_init_func = L.declare_function "list_init" list_init_t the_module in

let list_size_t = L.function_type i32_t [| list_t |] in
let list_size_func = L.declare_function "list_size" list_size_t the_module in

let list_get_t = L.function_type void_ptr_t [| list_t; i32_t |] in
let list_get_func = L.declare_function "list_get" list_get_t the_module in

let list_add_t = L.function_type i32_t [| list_t; void_ptr_t |] in
let list_add_func = L.declare_function "list_add" list_add_t the_module in

(* printl() does not yet support all list data types *)
let printl_t : L.lltype =
  L.function_type list_t [| list_t |] in
let printl_func : L.llvalue =
  L.declare_function "printl" printl_t the_module in
```

```
  (* Type casting for use with linked list*)

 (* From Int *)
 let list_set_int_t = L.function_type i32_t [| list_t; i32_t; i32_t |] in
 let list_set_int_func = L.declare_function "list_set_int" list_set_int_t
the_module in

 let list_add_int_t = L.function_type i32_t [| list_t; i32_t |] in
 let list_add_int_func = L.declare_function "list_add_int" list_add_int_t
the_module in

 (* From String *)
 let list_set_str_t = L.function_type string_t [| list_t; i32_t; string_t |] in
 let list_set_str_func = L.declare_function "list_set_str" list_set_str_t
the_module in

 let list_add_str_t = L.function_type i32_t [| list_t; string_t |] in
 let list_add_str_func = L.declare_function "list_add_str" list_add_str_t
the_module in

 (* From Float *)
 let list_set_float_t = L.function_type float_t [| list_t; i32_t; float_t |] in
 let list_set_float_func = L.declare_function "list_set_float" list_set_float_t
the_module in

 let list_add_float_t = L.function_type i32_t [| list_t; float_t |] in
 let list_add_float_func = L.declare_function "list_add_float" list_add_float_t
the_module in

 (* End of List Functions *)

 (* Define each function (arguments and return type) so we can
    call it even before we've created its body *)
 let function_decls : (L.llvalue * sfunc_decl) StringMap.t =
   let function_decl m fdecl =
     let name = fdecl.sfname
     and formal_types =
       Array.of_list (List.map (fun (t,_) -> ltype_of_typ t) fdecl.sformals)
```

```
    in let ftype = L.function_type (ltype_of_typ fdecl.styp) formal_types in
    StringMap.add name (L.define_function name ftype the_module, fdecl) m in
  List.fold_left function_decl StringMap.empty functions in

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

  let int_format_str = L.build_global_stringptr "%d<br>\n" "fmt" builder
  and float_format_str = L.build_global_stringptr "%g<br>\n" "fmt" builder
  and string_format_str = L.build_global_stringptr "<span>%s</span><br>\n"
"fmt" builder
  in

  (* Construct the function's "locals": formal arguments and locally
     declared variables.  Allocate each on the stack, initialize their
     value, if appropriate, and remember their values in the "locals" map *)
  let local_vars =
    let add_formal m (t, n) p =
      L.set_value_name n p;
      let local = L.build_alloca (ltype_of_typ t) n builder in
      ignore (L.build_store p local builder);
      StringMap.add n local m

    (* Allocate space for any locally declared variables and add the
     * resulting registers to our map *)
    and add_local m (t, n) =
      let local_var = L.build_alloca (ltype_of_typ t) n builder
      in StringMap.add n local_var m
    in

    let formals = List.fold_left2 add_formal StringMap.empty fdecl.sformals
        (Array.to_list (L.params the_function)) in
    List.fold_left add_local formals fdecl.slocals
  in

  (* Return the value for a variable or formal argument.
```

```
      Check local names first, then global names *)
    let lookup n = try StringMap.find n local_vars
      with Not_found -> StringMap.find n global_vars
    in

    (* Construct code for an expression; return its value *)
    let rec expr builder ((styp, e) : sexpr) = match e with
        SLiteral i  -> L.const_int i32_t i
      | SBoolLit b  -> L.const_int i1_t (if b then 1 else 0)
      | SStringLit s -> L.build_global_stringptr s "str" builder
      | SFliteral l -> L.const_float_of_string float_t l
      | SNoexpr     -> L.const_int i32_t 0
      | SId s       -> L.build_load (lookup s) s builder
      | SAssign (s, e) -> let e' = expr builder e in
        ignore(L.build_store e' (lookup s) builder); e'

      | SBinop ((A.Float,_ ) as e1, op, e2) ->
        let e1' = expr builder e1
        and e2' = expr builder e2 in
        (match op with
          A.Add     -> L.build_fadd
        | A.Sub     -> L.build_fsub
        | A.Mult    -> L.build_fmul
        | A.Div     -> L.build_fdiv
        | A.Equal   -> L.build_fcmp L.Fcmp.Oeq
        | A.Neq     -> L.build_fcmp L.Fcmp.One
        | A.Less    -> L.build_fcmp L.Fcmp.Olt
        | A.Leq     -> L.build_fcmp L.Fcmp.Ole
        | A.Greater -> L.build_fcmp L.Fcmp.Ogt
        | A.Geq     -> L.build_fcmp L.Fcmp.Oge
        | A.And | A.Or ->
          raise (Failure "internal error: semant should have rejected and/or on
float")
        ) e1' e2' "tmp" builder

      | SBinop (e1, op, e2) ->
        let e1' = expr builder e1
        and e2' = expr builder e2 in
```

```
      (match op with
         A.Add     -> L.build_add
       | A.Sub     -> L.build_sub
       | A.Mult    -> L.build_mul
       | A.Div     -> L.build_sdiv
       | A.And     -> L.build_and
       | A.Or      -> L.build_or
       | A.Equal   -> L.build_icmp L.Icmp.Eq
       | A.Neq     -> L.build_icmp L.Icmp.Ne
       | A.Less    -> L.build_icmp L.Icmp.Slt
       | A.Leq     -> L.build_icmp L.Icmp.Sle
       | A.Greater -> L.build_icmp L.Icmp.Sgt
       | A.Geq     -> L.build_icmp L.Icmp.Sge
      ) e1' e2' "tmp" builder

    | SUnop(op, ((t, _) as e)) ->
      let e' = expr builder e in
      (match op with
         A.Neg when t = A.Float -> L.build_fneg
       | A.Neg                  -> L.build_neg
       | A.Not                  -> L.build_not) e' "tmp" builder

    | SListLength (l) -> let l' = expr builder l in
       L.build_call list_size_func [|l'|] "list_size" builder;

    | SListGet(l, idx) ->
      let ltype = ltype_of_typ styp in
      let lst = expr builder l in
      let index = expr builder idx in
      let value = L.build_call list_get_func [| lst; index |] "index" builder
in
      (match styp with
        A.List _ | A.String -> L.build_bitcast value ltype "value" builder
      | _ -> let value = L.build_bitcast value (L.pointer_type ltype) "value"
builder in
        L.build_load value "value" builder)

    | SListSet(l, idx, e) ->
```

```
      let m = (match check_list_typ(l) with
        A.Int -> let l' = expr builder l and idx' = expr builder idx and e' =
expr builder e in
        L.build_call list_set_int_func [|l'; idx'; e'|] "list_set_int"
builder;
      | A.String -> let l' = expr builder l and idx' = expr builder idx and e'
= expr builder e in
        L.build_call list_set_str_func [|l'; idx'; e'|] "list_set_str"
builder;
      | A.Float -> let l' = expr builder l and idx' = expr builder idx and e'
= expr builder e in
        L.build_call list_set_float_func [|l'; idx'; e'|] "list_set_float"
builder;
      | _ -> raise(Failure("Invalid type of list literal"))) in m

    | SListAdd(l, e) -> let m = (match get_typ(e) with
     A.Int -> let l' = expr builder l and e' = expr builder e in
      L.build_call list_add_int_func [|l'; e'|] "list_add_int" builder;
    | A.String -> let l' = expr builder l and e' = expr builder e in
      L.build_call list_add_str_func [|l'; e'|] "list_add_str" builder;
    | A.Float -> let l' = expr builder l and e' = expr builder e in
      L.build_call list_add_float_func [|l'; e'|] "list_add_float" builder;
    | _ -> raise(Failure("Invalid type of list literal"))) in m

    | SListLit l -> let rec fill lst = (function
        [] -> lst
        | sx :: rest ->
          let (t, _) = sx in
            let value = (match t with
              A.List _ | A.String -> expr builder sx
              | _ -> let value = L.build_malloc (ltype_of_typ t) "value"
builder in
                let llvm = expr builder sx in
                ignore(L.build_store llvm value builder); value) in
                let value = L.build_bitcast value void_ptr_t "value" builder
in
                ignore(L.build_call list_add_func [| lst; value |] "list_add"
builder); fill lst rest) in
```

```
                let m = L.build_call list_init_func [||] "list_init" builder
in
                fill m l

    | SCall ("print", [e]) | SCall ("printb", [e]) ->
      L.build_call printf_func [| int_format_str ; (expr builder e) |]
"printf" builder
    | SCall ("prints", [e]) ->
      L.build_call printf_func [| string_format_str ; (expr builder e) |]
"printf" builder
    | SCall ("printf", [e]) ->
      L.build_call printf_func [| float_format_str ; (expr builder e) |]
"printf" builder

    (* HTML Library Function Calls *)
    | SCall ("createHTMLDocument", [e] ) ->
      L.build_call createHTMLDocument_func [| (expr builder e) |]
"createHTMLDocument" builder

    | SCall ("createElement", [e1;e2;e3] ) ->
      L.build_call createElement_func [| (expr builder e1); (expr builder
e2);(expr builder e3) |] "createElement" builder

    | SCall ("createHTML", [e] ) ->
      L.build_call createHTML_func [| (expr builder e) |] "createHTML" builder

    | SCall ("makeHeader", [e1]) ->
      L.build_call makeHeader_func [| (expr builder e1) |] "makeHeader"
builder
     | SCall ("makeText", [e1]) ->
      L.build_call makeText_func [| (expr builder e1) |] "makeText" builder

    | SCall ("makeImage", [e1]) ->
      L.build_call makeImage_func [| (expr builder e1) |] "makeImage" builder

    | SCall ("makeInput", [e]) ->
      L.build_call makeInput_func [| (expr builder e) |] "makeInput" builder
```

```
     (* List function printl() does not yet fully support all DRRTY list data
types *)
     | SCall ("printl", [e]) ->
         L.build_call printl_func [| (expr builder e) |] "printl" builder

     | SCall (f, args) ->
       let (fdef, fdecl) = StringMap.find f function_decls in
       let llargs = List.rev (List.map (expr builder) (List.rev args)) in
       let result = (match fdecl.styp with
             A.Void -> ""
           | _ -> f ^ "_result") in
       L.build_call fdef (Array.of_list llargs) result builder
    in

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


   (* Build the code for the given statement; return the builder for
      the statement's successor (i.e., the next instruction will be built
      after the one generated by this call) *)

   let rec stmt builder = function
       SBlock sl -> List.fold_left stmt builder sl
     | SExpr e -> ignore(expr builder e); builder
     | SReturn e -> ignore(match fdecl.styp with
         (* Special "return nothing" instr *)
           A.Void -> L.build_ret_void builder
         (* Build return statement *)
         | _ -> L.build_ret (expr builder e) builder );
       builder
     | SIf (predicate, then_stmt, else_stmt) ->
```

```
      let bool_val = expr builder predicate in
      let merge_bb = L.append_block context "merge" the_function in
      let build_br_merge = L.build_br merge_bb in (* partial function *)

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

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

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

    | SWhile (predicate, body) ->
      let pred_bb = L.append_block context "while" the_function in
      ignore(L.build_br pred_bb builder);

      let body_bb = L.append_block context "while_body" the_function in
      add_terminal (stmt (L.builder_at_end context body_bb) body)
        (L.build_br pred_bb);

      let pred_builder = L.builder_at_end context pred_bb in
      let bool_val = expr pred_builder predicate in

      let merge_bb = L.append_block context "merge" the_function in
      ignore(L.build_cond_br bool_val body_bb merge_bb pred_builder);
      L.builder_at_end context merge_bb


    (* Implement for loops as while loops *)
    | SFor (e1, e2, e3, body) -> stmt builder
                      ( SBlock [SExpr e1 ; SWhile (e2, SBlock [body ; SExpr
e3]) ] )
  in

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

```
  let builder = stmt builder (SBlock fdecl.sbody) in

  (* Add a return if the last block falls off the end *)
  add_terminal builder (match fdecl.styp with
        A.Void -> L.build_ret_void
      | A.Float -> L.build_ret (L.const_float float_t 0.0)
      | t -> L.build_ret (L.const_int (ltype_of_typ t) 0))
  in

  List.iter build_function_body functions;
  the_module
```

8.7 drrty.ml

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

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),
     "Check and print the generated LLVM IR (default)");
 ] in
 let usage_msg = "usage: ./drrty.native [-a|-s|-l|-c] [file.drrt]" in
 let channel = ref stdin in
 Arg.parse speclist (fun filename -> channel := open_in filename) usage_msg;

 let lexbuf = Lexing.from_channel !channel in
 let ast = Drrtyparse.program Scanner.token lexbuf in
```

```
 match !action with
   Ast -> print_string (Ast.string_of_program ast)
 | _ -> let sast = Semant.check ast in
   match !action with
     Ast     -> ()
   | Sast    -> print_string (Sast.string_of_sprogram sast)
   | LLVM_IR -> print_string (Llvm.string_of_llmodule (Codegen.translate sast))
   | Compile -> let m = Codegen.translate sast in
 Llvm_analysis.assert_valid_module m;
 print_string (Llvm.string_of_llmodule m)
```

8.8 listlibrary.c

```
/* DRRTY standard library */

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

/* Linked list implementation */

struct node{
   void * value;
   struct node * next;
};

struct list{
   int size;
   struct node * head;
};
```

```
//Initialize list
struct list* list_init(){

   struct list *l;
   l = malloc(sizeof(struct list));

   if (l == NULL)
       return NULL;

   l->size = 0;
   l->head = 0;

   return l;
}

//Return length of list
int list_size(struct list *l){

   return l->size;
}

//Return element at index i of list
void * list_get(struct list *l, int i) {

   if (l->head == NULL || l->size <= i || i < 0)
       return NULL;

   struct node *current = l->head;

   int j = 0;
   while (j != i) {
       current = current->next;
               j++;
   }

   return current->value;
}
```

```
//Set value at index i of list
int list_set(struct list *l, int i, void *value){

   if (l->head == NULL || l->size <= i || i < 0)
       return 0;

   struct node *current = l->head;

   int j = 0;
   while (j != i) {
       current = current->next;
       j++;
   }

   current->value = value;
   return 1;
}

//Add element to end of list
int list_add(struct list *l, void *value){

   struct node *new = (struct node *)malloc(sizeof(struct node));

   if (new == NULL)
       return 0;

   new->value = value;
   new->next = NULL;

   if (l->head == NULL) {
       l->size +=  1;
       l->head = new;
       return 1;
   }

   struct node *current = l->head;
   while (current->next != NULL) {
       current = current->next;
```

```
    }

    current->next = new;
    l->size += 1;
    return 1;
}

//list_pop() not integrated into code generator
void * list_pop(struct list *l) {

    if (l->head == NULL)
        return NULL;

    if (l->head->next == NULL) {
        struct node *old = l->head;
        l->head = old->next;
        void *value = old->value;
        free(old);
        l->size -= 1;
        return value;
        }

    struct node *temp1 = l->head;
    struct node *temp2 = l->head->next;

    while (temp2->next != NULL) {
        temp1 = temp2;
        temp2 = temp2->next;
    }

    void *value = temp2->value;
    temp1->next = NULL;
    l->size -= 1;
    free(temp2);
    return value;
}

//printl() does not yet support all DRRTY list data types
```

```
void printl(struct list *l) {

   printf("[");

   struct node *current = l->head;
   while (current != NULL){
           if(current->next == NULL)
               printf("%d", *(int *) current->value);
           else
               printf("%d,", *(int *)current->value);

           current = current->next;
   }
   printf("]\n");
}

/* Type casting for use with linked list */

// From Integer
int list_add_int (struct list * l, int value)
{
   int * d = malloc(sizeof(int));
   *d = value;
   return list_add(l, d);
}

int list_get_int(struct list * l, int index)
{
   void * answer = list_get(l, index);
   return *(int *) answer;
}

int list_set_int(struct list * l, int index, int x)
{
   int answer = list_get_int(l, index);
   int * d = malloc(sizeof(int));
   *d = x;
   list_set(l, index, (void *) d);
```

```
    return answer;
}

int list_pop_int(struct list * l)
{
    void * answer = list_pop(l);
    return *(int *) answer;
}

// From Float
int list_add_float (struct list * l, double value)
{
    double * d = malloc(sizeof(double));
    *d = value;
    return list_add(l, d);
}

double list_get_float(struct list * l, int index)
{
    void * answer = list_get(l, index);
    return *(double *) answer;
}

double list_set_float(struct list * l, int index, double x)
{
    double answer = list_get_float(l, index);
    double * d = malloc(sizeof(double));
    *d = x;
    list_set(l, index, (void *) d);
    return answer;
}

double list_pop_float(struct list * l)
{
    void * answer = list_pop(l);
    return *(double *) answer;
}
```

```
// From String
int list_add_str (struct list * l, char * value)
{
   return list_add(l, (void *) value);
}

char * list_get_str(struct list * l, int index)
{
   void * answer = list_get(l, index);
   return (char *) answer;
}

char * list_set_str(struct list * l, int index, char * x)
{
   char * answer = list_get_str(l, index);

   list_set(l, index, (void *) x);
   return answer;
}

char * list_pop_str(struct list * l)
{
   void * answer = list_pop(l);
   return (char *) answer;
}
```

8.9 htmllibrary.c

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

/* HTML functions */
void createHTMLDocument(char *cssFile){
 printf("<!DOCTYPE html><html><head><link rel=\"stylesheet\"
href=\"%s\"></head><body>", cssFile);
}
```

```
void createElement(char *elementName, char *innerHtml, char *className){
 if( strcmp("img", elementName) == 0  || strcmp("input",elementName) == 0 ){
   printf("<%s class=\"%s\" src=\"%s\">\n", elementName, className, innerHtml);
// <img class="" src="">
 }else{
   printf("<%s class=\"%s\">%s</%s>\n", elementName, className, innerHtml,
elementName);
 }
}

void createHTML(char *html){
 printf("%s", html);
}

void makeHeader(char *innerHtml){
 printf("<h1>%s</h1>\n", innerHtml);
}

void makeText(char *innerHtml){
 printf("<p>%s</p>\n", innerHtml);
}

void makeImage(char *innerHtml){
 printf("<img src=\"%s\">\n", innerHtml);
}

void makeInput(char *className){
 printf("<input class=\"%s\">\n", className);
}

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

```
#endif
```

8.10 Makefile

```
# "make test" Compiles everything and runs the regression tests

.PHONY : test
test : all testall.sh
   ./testall.sh

.PHONY : all
all : drrty.native listlibrary.o htmllibrary.o

listlibrary.bc: listlibrary.c
      clang -emit-llvm -o listlibrary.bc -c listlibrary.c -Wno-varargs

listlibrary: listlibrary.c
      cc -o listlibrary -DBUILD_TEST listlibrary.c

htmllibrary : htmllibrary.c
      cc -o htmllibrary -DBUILD_TEST htmllibrary.c

drrty.native :
      opam config exec -- \
      rm -f *.o
      ocamlbuild -use-ocamlfind -pkgs llvm,llvm.analysis,llvm.bitreader
      drrty.native

# "make clean" removes all generated files

.PHONY : clean
clean :
   ocamlbuild -clean
   rm -rf testall.log *.diff drrty scanner.ml parser.ml parser.mli
   rm -rf *.cmx *.cmi *.cmo *.cmx *.ll *.html
```

```
   rm -rf *.err *.out *.exe *.s
   rm -f *.o


# Building the tarball

TESTS = \
      add1 arith1 arith2 arith3 fib float1 float2 float3 for1 for2 func1 func3
      func4 \
      func5 func6 func7 func8 func9 gcd gcd2 global1 global2 global3 hello
      hello2 \
      if1 if2 if3 if4 if5 if6 local1 local2 ops1 ops2 remainder stringconcat \
   stringconcat2 var1 var2

FAILS = \
      assign1 assign2 assign3 assign4 dead1 expr1 expr2 float1 for1 for2 for3
      func1  \
      func2 func3 func4 func5 func6 func7 global1 global2 if1 if2 if3 nomain \
      return1 return2 while1 while2


TESTFILES = $(TESTS:%=test-%.drrt) $(TESTS:%=test-%.out) \
       $(FAILS:%=fail-%.drrt) $(FAILS:%=fail-%.err)

TARFILES = ast.ml sast.ml codegen.ml Makefile _tags drrty.ml drrtyparse.mly \
   README scanner.mll semant.ml testall.sh \
arcade-font.pbm font2c \
   Dockerfile \
   $(TESTFILES:%=tests/%)

drrty.tar.gz : $(TARFILES)
   cd .. && tar czf drrty/drrty.tar.gz \
       $(TARFILES:%=drrty/%)
```

8.11 testall.sh

```
#!/bin/sh

# Regression testing script for DRRTY
# Step through a list of files
#  Compile, run, and check the output of each expected-to-work test
#  Compile and check the error of each expected-to-fail test

# Path to the LLVM interpreter
LLI="lli"
#LLI="/usr/local/opt/llvm/bin/lli"

# Path to the LLVM compiler
LLC="llc"
#LLC="/usr/local/opt/llvm/bin/llc"

# Path to the C compiler
CC="cc"
# CC="/usr/local/opt/llvm/bin/cc"

# Path to the drrty compiler.  Usually "./drrty.native"
# Try "_build/drrty.native" if ocamlbuild was unable to create a symbolic link.
DRRTY="./drrty.native"
#DRRTY="_build/drrty.native"

# Set time limit for all operations
ulimit -t 30

globallog=testall.log
rm -f $globallog
error=0
globalerror=0

keep=0

Usage() {
```

```
    echo "Usage: testall.sh [options] [.drrt files]"
    echo "-k    Keep intermediate files"
    echo "-h    Print this help"
    exit 1
}

SignalError() {
    if [ $error -eq 0 ] ; then
    echo "FAILED"
    error=1
    fi
    echo "  $1"
}

# Compare <outfile> <reffile> <difffile>
# Compares the outfile with reffile.  Differences, if any, written to difffile
Compare() {
    generatedfiles="$generatedfiles $3"
    echo diff -b $1 $2 ">" $3 1>&2
    diff -b "$1" "$2" > "$3" 2>&1 || {
    SignalError "$1 differs"
    echo "FAILED $1 differs from $2" 1>&2
    }
}

# Run <args>
# Report the command, run it, and report any errors
Run() {
    echo $* 1>&2
    eval $* || {
    SignalError "$1 failed on $*"
    return 1
    }
}

# RunFail <args>
# Report the command, run it, and expect an error
RunFail() {
```

```
    echo $* 1>&2
    eval $* && {
    SignalError "failed: $* did not report an error"
    return 1
    }
    return 0
}
```

8.12 demo.css

```
img{
    width: 250px;
    height: auto ;
}

.column{
    width:50%;
    float:left;
    font-size: 50px;
    padding:0;
}
.row{
    padding:0;
    width:50%;
    float:left;
    font-size: 50px;
}
```

8.13 printbig.css

```
@import
url('https://fonts.googleapis.com/css2?family=Varela+Round&display=swap');

body{
    background-color:#000000;
    color: white;
    padding-top: 10px;
}

h1{
    font-family: 'Varela Round', sans-serif;
```

```
}

.bigClass{
    font: 700;
    font-family: sans-serif;
    color: white;
    background-color: purple;
    width: 200px;
    height: 200px;
    display: inline-block;
}

.smallClass{
    font: 700;
    font-family: sans-serif;
    color: white;
    background-color: white;
    max-width: 200px;
    max-height: 200px;
    width: auto;
    height: auto;
    display: inline-block;
}

.lazy-or-stupid{
    font-family: 'Varela Round', sans-serif;
}

.column{
    width: 30%;

}
```

8.14 Testing Files

8.14.1 fail-arithtype.drrt

```
def int main(){
      /*Invalid operation between string and int type */
      int i = 5;
      str s = "6";
      print(i-s)

      return 0;
}
```

8.14.2 fail-assign1.drrt

```
def int main(){
	/*Assigning variables to wrong data type*/
	int x;
	bool y;
	x= True;
	y = 3;

	return 0;
}
```

8.14.3 fail-comment.drrt

```
def int main(){
	/*This comment will be incomplete
	return 0;
}
```

8.14.4 fail-compare.drrt

```
def int main(){
	/*Can't use compare operators on strings*/
	str name;
	str color;
	bool b;

	name="Rania";
	color="Orange";
	b = name > color;

	return 0;
}
```

8.14.5 fail-createEl.drrt

```
def int main(){
	/*int type instead of string for last parameter*/
	int m;
	m=9;

	createElement("div",int,"");
	return 0;
}
```

8.14.6 fail-forloop.drrt

```
def int main(){
/*Invalid placement of semicolon after for loop, incorrect use*/
	int i;
	for( i = 0; i < 10 ; i = i +1) ; {
		print(i);
	}
	return 0;
}
```

8.14.7 fail-funcdec.drrt

```
int counter(){ /*incorrect function declaration*/
	int i;
	for (i=0; i<5; i=i+1){
		print(i);
	}
}
def int main(){
	return 0;
}
```

8.14.8 fail-hello.drrt

```
def int main(){
	makeHeader(Hello!); /* invalid input string*/
	return 0;
}
```

8.14.9 fail-htmlfunctions.drrt

```
def int main(){ /*Incorrect input type, not a string*/
	makeHeader(Hello World);
	makeText(Hello There);

	return 0;
}
```

8.14.10 fail-if-else.drrt

```
def int main(){ /*No conditional in the if statement*/
	if (){
		prints("Yes");
	}else{
```

```
            prints("No");
      }
      return 0;
}
```

8.14.11 fail-if-else2.drrt

```
def int main(){ /*else without if*/
      int a = 3;
      else{
            a = a+1;
      }
      return 0;
}
```

8.14.12 fail-indexing.drrt

```
def int main(){ /*Cannot index with a string, invalid input*/
      list[int] numbers;
      numbers = [1,2,3,4,5];
      print(numbers.get("7"));
      return 0;
}
```

8.14.13 fail-listass.drrt

```
def int main(){ /*not valid strings in the list*/
      list[str] L1;
      L1 = [hi,hello,how are you];

      return 0;
}
```

8.14.14 fail-main-dec.drrt

```
def int main{ /*incorrect declaration, missing parentheses*/
      return 0;
}
```

8.14.15 fail-missing-semi.drrt

```
def int main(){ /*missing semicolon*/
      makeHeader("Hello World");
      makeText("My name is")

      return 0;
```

```
}
```

8.14.16 fail-nomain.drrt

```
/*No main function, empty file*/
```

8.14.17 fail-print-string.drrt

```
def int main(){ /*invalid input string*/
      prints(Hello);
      return 0;
}
```

8.14.18 fail-return.drrt

```
def int letter(){/*return type does not match what is defined*/
      str s = "hello"
      return s;
}
def int main(){ /*return type also does not match*/
      return letter();
}
```

8.14.19 fail-var-dec.drrt

```
def int main(){ /*no variable declaration*/
      a = "Hello there" ;
      prints(a);
}
```

8.14.20 fail-while.drrt

```
def int main(){ /*will go on forever, infinite loop*/
      int i = 0;
      while(i = 0){
            print(i);
      }
      return 0;
}
```

8.14.21 test-arithmetic.drrt

```
def void arithmetic( int x, int y){
/*test all of the arithmetic and logic operators*/
      print(x + y);
```

```
	print(x-y);
	print(x*y);
	print(x/y);
	printb(x == y);
	printb(x == x);
	printb(x!=y);
	printb(x!=x);
	printb(x>y);
	printb(x>=y);
	printb(x<y);
	printb(x<=y);
}

def int main(){
	arithmetic(100,10);
	arithmetic( 50,25);
	return 0;
}
```

8.14.22 test-assign1.drrt

```
def int main(){
/*assign and declare for each data type*/
	int x;
	bool b;
	str s;
	list[int] numb;
	list[str] names;

	x = 0;
	b = True;
	s = "Hello!";
	numb = [1,2,3];
	names = ["Rania","Richard","Dylan","Trinity"];

	return 0;
}
```

8.14.23 test-assign2.drrt

```
def int main(){
/*Assigning products of operators to a variable*/
	int z;
	int x;
	int y;
	bool on;

	x = 3;
```

```
	y = 2;
	z = x+y ;
	on = x>y;

	print(z);

	return 0;
}
```

8.14.24 test-comment.drrt

```
def int main(){
/*This is a comment and will not show in the output*/
	 return 0;
}
```

8.14.25 test-createElement-div.drrt

```
def int main(){
/*Create an html document, link a .css file, and create a div*/
	createHTMLDocument("printbig.css");
	createElement("div","newel","This element will have a different color");
	return 0;
}
```

8.14.26 test-createElement.drrt

```
def int main(){
/*Creating a horizontal line with HTML to separate headers*/
	makeHeader("This is the first separated header");
	createElement("hr", "","");
	makeHeader("This is the other separated header");

	return 0;
}
```

8.14.27 test-demo2.drrt

```
def int main(){
/*The code for the demo used in the presentation, makes a checkerboard*/

  int i;
  int j;
  list[str] images;

  createHTMLDocument("printbig.css");
  makeHeader("\"Umm we've been lazy so we haven't been using GitHub...\"");
```

```
  createHTML("<h2 class=\"lazy-or-stupid\">\"That's not lazy, that's stupid!\"
- Prof. Edwards</h2>");

  images = ["1.jpeg","2.jpeg","3.jpeg","4.jpeg"];

  for (i = 0; i < 28; i = i + 1){
      if( j == 0){
        createElement("div", "", "bigClass");
        j = 1;
      }else{
        if (i >= 0 && i <7){
            createElement("img", images.get(0), "smallClass");
        }if (i >= 7 && i < 14){
            createElement("img", images.get(1), "smallClass");
        }if (i >= 14 && i < 21){
            createElement("img", images.get(2), "smallClass");
        }if (i >= 21 && i < 28){
            createElement("img", images.get(3), "smallClass");
        }
        j = 0;
      }
  }
  return 0;
}
}
```

8.14.28 test-fib.drrt

```
def int fib(int x){ /*Fibonacci, testing recursion&arithmetic*/
      if (x<2){
            return 1;
      }
      return fib(x-1) + fib(x-2);
}
def int main() {
      print(fib(1));
      print(fib(5));

      return 0;
}
```

8.14.29 test-for.drrt

```
def int main(){
/*Iterating through 5 numbers with a for loop*/
      int i;
      for (i = 0; i < 5; i = i + 1){
        print(i);
```

```
    }
      return 0;
}
```

8.14.30 test-forloop.drrt

```
def int counter(int x){ /*test for loop and function calls*/
      int i;
      for (i = 0; i<x; i = i+1){
            print(i);
      }
      return 0;
}

def int main() {
      counter(1);
      counter(10);
      counter(100);

      return 0;
}
```

8.14.31 test-gcd.drrt

```
def int gcd (int x, int y) { /*testing gcd algorithm*/
      while (x!=y) {
            if (x>y) {
                  x = x - y;
            }else{
                  y = y-x;
            }
      }
      return x;
}

def int main(){
      print(gcd(4,20));
      print(gcd(10,100));
      return 0;
}
```

8.14.32 test-hello.drrt

```
def int main(){ /*testing makeHeader and makeText functions*/
      makeHeader("Hello!");
      makeText("DRRTY's first program!");
      return 0;
```

```
}
```

8.14.33 test-if-else.drrt

```
def int main(){ /*testing simple if-else statements*/
        if( False ){
              prints("Hello world");
          }
          else{
              prints("Sorry world");
          }
          return 0;
}
```

8.14.34 test-if1.drrt

```
def int main(){ /*testing simple if statements with booleans*/
         if( True ){
              prints("Hello world");
          }
          return 0;
}
```

8.14.35 test-index-list.drrt

```
def int main(){ /*using .get() function to index a list*/
      list[str] names;
      names = ["Trinity", "Dylan", "Richard", "Rania"];
      prints(names.get(2));

      return 0;
}
```

8.14.36 test-input-form.drrt

```
def int main(){ /*using makeInput() and createElement together*/
      str head;
      head = "This is the header for my form!";
      makeHeader(head);

      makeText("Please enter your name in here");
      makeInput("");/* Takes in class name*/

      createElement("button","Submit","");
      return 0;
}
```

8.14.37 test-length-list.drrt

```
def int main(){ /*using .length() for length of list*/
      list[int] numbers;
      numbers = [1,2,3,4,5,6,7,8,9];
      print(numbers.length());

      return 0;
}
```

8.14.38 test-replace.drrt

```
def int main(){ /*test replaces values in the list with .set()*/
      list[int] numbers;
      numbers = [1,2,3,4,5,6,7];
      print(numbers.get(3));

      numbers.set(3, 44);
      print(numbers.get(3));

      return 0;
}
```

8.14.39 test-list.drrt

```
def int main(){ /*use .add() and initialize a list*/
      list[str] names;
      names = ["Dylan", "Richard", "Rania", "Trinity"];
      names.add("Edwards");
      prints(names.get(4));
      return 0;
}
```

8.14.40 test-main.drrt

```
def int main(){ /*test the main function*/
      return 0;
}
```

8.14.41 test-makehtmlist.drrt

```
def int main(){ /*use for loop to create list in html*/
      int i;
      list[str] names;
      names =["Rania", "Trinity", "Dylan", "Richard"];
```

```
	for (i=0; i<names.length();i = i+1){
		createElement("li",names.get(i),"");
	}
	return 0;
}
```

8.14.42 test-makeimg.drrt

```
def int main(){ /*use makeImage function*/
	makeImage("1.jpeg");
	return 0;
}
```

8.14.43 test-meet-the-team.drrt

```
def int main() { /*use for loops to use all html func together*/
	list[str] team;
	int i;
	list[str] job;
	list[str] img;

	team = ["Dylan","Richard","Rania","Trinity"];
	job = ["Manager","Language Guru", "Tester", "System Architect"];

	img =
["https://cdn.pixabay.com/photo/2021/12/16/20/13/ducks-6875155__340.jpg","https
://cdn.pixabay.com/photo/2021/10/08/06/45/seagulls-6690361__340.jpg","https://c
dn.pixabay.com/photo/2021/11/06/22/05/camels-6774540__340.jpg","https://cdn.pix
abay.com/photo/2021/11/16/08/01/animal-6800387__340.jpg"];

	makeHeader("Meet the DRRTY Team!");

	for(i=0; i<team.length();i = i+1){
		makeText(team.get(i));
		createElement("h2",job.get(i),"");
		makeImage(img.get(i));
	}
	return 0;
}
```

8.14.44 test-meetus-col.drrt

```
def int main(){ /*use for loops and css page to */
	list[str] team;
	int i;
	list[str] job;
	list[str] img;
```

```
	createHTMLDocument("demo.css");

	team = ["Dylan","Richard","Rania","Trinity"];
	job = ["Manager","Language Guru", "Tester", "System Architect"];

	makeHeader("Meet the DRRTY Team!");

	for (i = 0; i<team.length();i=i+1){
		createElement("div",team.get(i),"row");
		createElement("div",job.get(i),"column");
	}
	return 0;
}
```

8.14.45 test-print-string.drrt

```
def int main(){ /*use prints() function*/
	prints("hello world!");
	return 0;
}
```

8.14.46 test-printbig.drrt

```
def int main(){ /*use .css file and create checkerboard*/
	 int i;
	 int j;
	 createHTMLDocument("printbig.css");
	 for (i = 0; i < 100; i = i + 1){
		 if( j == 0){
			createElement("div", "bigClass", "");
			j = 1;
		}else{
		 createElement("div", "smallClass", "");
			j = 0;
		 }
	  }
	  return 0;
}
```

8.14.47 test-printbimg.drrt

```
def int main(){ /*use createElement() to make an img element*/
	createElement("img","imgClass","https://as1.ftcdn.net/jpg/00/33/17/90/220
_F_33179059_DAZeqcO1edV3WBODcrL9RGQ39todSjBV.jpg");
	return 0;
}
```

8.14.48 test-profile.drrt

```
def void profile (str name,  str desc, str img){
      makeHeader(name);
      makeText(desc);
      makeImage(img);
}
def int main(){
/*use html functions in elsewhere and call them in the main*/

      profile(“Bob the Builder”, “Bob builds programs and uses github to track
his changes”, “2.jpeg”);

      return 0;
}
```

8.14.49 test-str.drrt

```
def int main(){ /*test string initialization and printing*/
      str s;
      s = "hello world";
      prints(s);
      return 0;
}
```

8.14.50 test-var1.drrt

```
def int letter(){/*test variable declaration outside of main*/
  int a;
  a = 18;
  return a;
}

def int main(){
  print(letter());
  makeHeader("Hello!");
  makeText("DRRTY's first program!");
  return 0;
}
```

8.14.51 test-var2.drrt

```
def int main(){
/*test variable declaration and function calls in same program*/
  int a;
  a = 20;
```

```
  print(a);
  makeHeader("Hello!");
  makeText("DRRTY's first program!");
  return 0;
}
```

8.14.52 test-var3.drrt

```
int a;
/*test variable declaration outside the main and initializing inside the main
method*/
def int main(){
  a = 30;
  print(a);
  makeHeader("Hello!");
  makeText("DRRTY's first program!");
  return 0;
}
```

8.14.53 test-while-form.drrt

```
def int main(){
/*Use while loops and if-else to make input forms*/
      int i;
      bool b;

      b = True;
      i = 0;

      makeHeader("Log the names and numbers below!");

      while(i<6){
            if(b == True){
                  makeText("Name");
                  makeInput("");
                  createElement("button","Submit","");
                  i = i+1;
                  b = False;
            }
            else{
                  makeText("Phone Number");
                  makeInput("");
                  createElement("button","Submit","");
                  i = i+1;
                  b = True;
            }
      }
      return 0;
```

```
}
```

8.14.54 test-var3.drrt

```
def int main(){/*use while loops*/
int x;
      x = 1;
      while(x<5){
            prints("Hello");
            x = x+1;
      }
      return 0;
}
```

8.15 run.sh

```
#!/bin/sh


# Path to the LLVM interpreter
LLI="lli"
#LLI="/usr/local/opt/llvm/bin/lli"

# Path to the LLVM compiler
LLC="llc"
#LLC="/usr/local/opt/llvm/bin/llc"

# Path to the C compiler
CC="cc"
# CC="/usr/local/opt/llvm/bin/cc"

# Path to the drrty compiler.  Usually "./drrty.native"
# Try "_build/drrty.native" if ocamlbuild was unable to create a symbolic link.
DRRTY="./drrty.native"
#DRRTY="_build/drrty.native"

SignalError() {
    if [ $error -eq 0 ] ; then
      echo "FAILED"
      error=1
    fi
    echo "  $1"
}


Run() {
```

```
    echo $* 1>&2
    eval $* || {
      SignalError "$1 failed on $*"
      return 1
    }
}

basename=`echo $1 | sed 's/.*\\///
                            s/.drrt//'`

generatedfiles=""

    generatedfiles="$generatedfiles ${basename}.ll ${basename}.s
${basename}.exe ${basename}.out ${basename}.html" &&
    Run "$DRRTY" "$1" ">" "${basename}.ll" &&
    Run "$LLC" "-relocation-model=pic" "${basename}.ll" ">" "${basename}.s" &&
    Run "$CC" "-o" "${basename}.exe" "${basename}.s" "htmllibrary.o"
"listlibrary.o" &&
    Run "./${basename}.exe" > "${basename}.html"
```
