QWEB: Language Reference Manual

Xabier Peralta, Tester
xp2159@columbia.edu

Ramisa Murshed, Language Guru
rm3508@barnard.edu

Kamrul Hossain, Systems Architect
kh2857@columbia.edu

Tamanna Hussain, Project Manager
th2704@barnard.edu

April 26, 2021

Contents

1 Introduction

QWEB is an object-oriented, pseudocode-style website language inspired by block-based visual programming languages such as Scratch and Snap, as well as interactive visualization languages such as Processing.js. The purpose of the language is to help both novel and experienced programmers design and develop websites to run in their browser.

Although popular web programming languages such as HTML and CSS are generally easy to learn and understand, QWEB fuses the two into one and produces a language that is more intuitive, maximizes human readability, and incorporates familiar programming constructs that are used in traditional high-level programming languages. By replacing repetitive aspects of HTML and CSS such as tags and selectors, QWEB makes use of control flow statements, variable declarations, and objects instead. In essence, QWEB is to HTML and CSS as Processing.js is to JavaScript.

2 QWEB Tutorial

2.1 The QWEB Compiler

The QWEB compiler takes a pseudocode-style syntax and compiles it into LLVM IR. It was coded in OCaml.

It needs the OCaml llvm library, which is most easily installed through opam.

Install LLVM and its development libraries, the m4 macro preprocessor, and opam, then use opam to install llvm.

The version of the OCaml llvm library must match the version of the LLVM system installed on your system.

testall.sh runs the QWEB executable on each testcase (.qwb file) to produce a .ll file, invokes "llc" (the LLVM compiler) to produce a .s (assembly) file, then invokes "cc" (the stock C compiler) to assemble the .s file, and generate an executable and html file. See testall.sh for details.

If you get errors about llvm.analysis not being found, it's probably because opam enviroment information is missing. Either run

```
1 eval $\$$(opam config env)
```

or run ocamlbuild like this:

```
opam config exec -- ocamlbuild <args>
```

2.2 Installation under Ubuntu 16.04

LLVM 3.8 is the default under 16.04. Install the matching version of the OCaml LLVM bindings:

```
sudo apt install ocaml llvm llvm-runtime m4 opam
opam init
opam install llvm.3.8
eval `opam config env`
```

```
make
./testall.sh
```

To run and test:

```
$\$$ make
ocamlbuild -use-ocamlfind -pkgs llvm,llvm.analysis -cflags -w,+a-4
    qweb.native
Finished, 22 targets (0 cached) in 00:00:01.
$\$$ ./testall.sh
test-arith1...OK
test-arith2...OK
test-arith3...OK
test-fib...OK
...
fail-while1...OK
fail-while2...OK
```

After running make, you can then view the *.html files generated in the root directory and run it in a web browser.

3 Lexical Conventions

3.1 Comments

Comments are explanatory notes embedded within the code that are ignored by the compiler. QWEB has both single-line and multi-line comments. Single-line comments begin with a # symbol and multi-line comments begin and end with triple double quotes.

```
# This is a single-line comment

"""
This is a multi-line comment
"""
```

3.2 Identifiers

Valid identifiers, or names, consist of a sequence of alphanumeric characters or a single alphabetic character. Identifiers can contain an underscore, but they cannot be a QWEB keyword. They are also case sensitive; in other words, identifiers that differ in uppercase and lowercase letters are considered distinct.

```
x = 14 # x is a valid identifier because it is a single alphabetic
    character

1a = 10 # 1a is an invalid identifier because it begins with a
    number
```

3.3 Operators

Relational operators are used to evaluate a boolean statement. QWEB uses the following reserved operators:

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

For example:

```
 a = 1
 b = 2

output a > b # false
output a < b # true
output a >= b # false

 a = 2

output a <= b # true
```

3.4 Keywords

The list of identifiers reserved as keywords are below:

```
and            or           not
if         otherwise if   otherwise
true          false        output
int            str          bool
float        FOR each       output
REPEAT until
```

3.5 Indentation

Sequence control is indicated by writing one action after another on separate lines and with the same indent. For control flow statements, however, QWEB relies on brackets to indicate grouping. In other words, control flow keywords like `IF, OTHERWISE, OTHERWISE IF, REPEAT until`, and `FOR each` must be followed by a bracket with subsequent lines indented with a tab or four spaces and enclosed with a closing bracket.

```
# Example using indentation and brackets in a for each loop
FOR each key in data {
    FOR each val in key {
        table.append(val)
    }
}
```

3.6 Literals

Literals represents strings or one of QWEB's primitive data types: int, char, float, and boolean.

3.6.1 String Literals

A string literal is a sequence of alphabetic characters enclosed in single or double quotation marks. Examples of string literals include:

```
"QWEB"
'Websites'
```

3.6.2 Int Literals

An integer literal is any sequence of integers between 0 and 9. Examples of integer literals include:

```
12
25
62
```

```
34
97
```

3.6.3 Char Literals

A character literal consists of a single letter from the alphabet. Examples of char literals include:

```
a
v
c
d
e
```

3.6.4 Boolean Literals

Boolean types are represented by the `True` and `False` keywords.

```
True
False
```

3.6.5 Float Literals

A float literal is a number with a whole number, optional decimal point, a fraction, and an exponent. Examples of float literals include:

```
10
11.5
5.25
6e+2
0.005
```

4 Data Types

4.1 Primitives

In QWEB, there are four primitive data types available: **`int`, `char`, `float`, `bool`,** and **`string`**.

4.1.1 `int`

`int` represents integers and consists of one or more characters in the range 0-9. Examples of declarations are:

```
int b
 b = 23
 int c = 45
```

4.1.2 `char`

`char` represents characters like symbols and letters (A-Z). In order to use `char`, the char element must be written in between single quotes and can be declared as the following:

```
char n
 n = 'X'
 char m = '$'
```

4.1.3 `float`

`float` represents floating point numbers and consists only of numbers that have a decimal point. Declaration is as follows:

```
 float a = 10.5
float b
 b = 50.0
```

4.1.4 `bool`

`bool` can only represent a boolean value of either true or false. An example of a declaration is:

```
 bool a = false
bool b
 b = true
```

4.1.5 `str`

`str` represents a sequence of characters stored in a character array. Declaration is as follows:

```
    str x = "language"
```

5 Statements and Expressions

5.1 Statements

QWEB supports the use of statements in order to iterate or perform repetitive actions. All statements are executed sequentially.

5.1.1 `if-else`

QWEB supports the use of conditional statements, including Pythonic `if-else` statements. In QWEB, *if* is denoted by an all-caps `IF`, *elif* is denoted by `OTHERWISE IF`, and *else* is denoted by `OTHERWISE`. Each statement is separated by curly braces to denote the end of the statement. If an `IF` statement evaluates to the boolean false, then `OTHERWISE IF` is evaluated. If this statement also evaluates to false, then `OTHERWISE` is executed.

```
    IF (x == 5) {
            output "x is 5"
    } OTHERWISE IF (x == 3) {
            output "x is 3"
    } OTHERWISE {
            output "x is not 5 or 3"
    }
```

5.1.2 `for`

The usage of `for` loops is also supported by QWEB. They serve a similar function as `for` loops in Python and are denoted by the keywords `FOR each`. These loops iterate through elements in a structure such as a list and allow for the repeated execution of a block of code, for which beginning and ending is indicated by curly braces. Iteration begins at 0.

```
    FOR each element in list {
        output element
    }
```

5.1.3 `while`

QWEB also supports the use of Pythonic `while` loops, which are indicated in QWEB by the keywords `REPEAT until`. These loops repeatedly execute a block

of code (contained in the curly braces) until the boolean statement that follows `REPEAT until` no longer evaluates to true.

```
1     int x = 0
2
3    REPEAT until (x > 6) {
4        output "Hello World"
5         x = x + 1
6    }
```

5.1.4 `output`

QWEB denotes `return` statements with the keyword `output`. This can be used in any function in QWEB and is not constrained to any particular type. Whenever an `output` statement is called, the program exits out of the current loop and returns the value denoted after the keyword `output`.

```
1    function int incrementX (int x) {
2         result = x + 1
3        output result
4    }
```

5.2 Expressions and Operators

5.2.1 Assignment Operator

The assignment operator in QWEB is denoted by the keyword `...=`, which assigns a value to a variable. The value directly after the keyword `=` is assigned to the variable indicated between the keywords and `=`.

```
1     a = 4
2
3     x = "Hello World"
```

5.2.2 Arithmetic Operators

QWEB supports the use of the arithmetic operators + (to add or concatenate values), - (to subtract values), * (to multiply values), / (to divide values), and % (to perform modulo operations).

5.2.3 Logical Operators

QWEB also makes use of the logical operators `and, or,` and `not`, providing an identical function to that of its usage in Python.

5.3 HTML Functions

5.3.1 `createHeader`

`createHeader` is a function that returns an HTML header. It takes in two values: the first being a string that contains the text to be put in the header, and the second being an optional parameter of type `int` from 1 to 6, inclusive, that specifies which size the header should be. If the second parameter is omitted, then it defaults to `<h1>`.

```
    str headerText = "This is a header."

    head = createHeader(headerText)
   output head #prints "This is a header." header in size h1
```

5.3.2 `createParagraph`

`createParagraph` is a function that returns an HTML paragraph. It takes in a string that contains the text to be put in the paragraph.

```
    str paragraphText = "This is a paragraph."

    par = createParagraph(paragraphText)
   output head #prints "This is a paragraph." paragraph
```

5.3.3 `createSubheader`

`createSubheader` is a function that returns an HTML subheader (h2). It takes in a string that contains the text to be put in the subheader.

```
    str subheaderText = "This is a subheader."

    subhead = createSubheader(subheaderText)
   output subhead #prints "This is a subheader." header in size h2
```

5.3.4 `createImage`

`createImage` is a function that returns an image (¡img¿). It takes in a string that contains the link to be put in the image.

```
    str imageText = "https://source.unsplash.com/user/c_v_r"

    image = createImage(imageText)
   output image #prints the image from that link to the page.
```

5.3.5 createList

createList is a function that returns an HTML unordered list. It takes in a list that contains all of the values to be put into the unordered list.

```
    ulList = [a, b, c, d]

    ul to createList(ulList)
   output ul # prints out an unordered list of <li> elements
```

6 Sample Programs

The following programs illustrates an example of how to use QWEB:

6.1 Bio Portfolio Website

```
function int main(){
  createHeader("Stephen A. Edwards");
  createImage("./edwards.jpeg");
  createSubheader("Contact Information");
  prints("sedwards@cs.columbia.edu");
  prints("+1 212 939 7019");
  prints("+1 212 666 0140 (fax)");
  prints("462 Computer Science Building");
  prints("1214 Amsterdam Ave MC 0401 New York, NY 10027-7003");
  createSubheader("About");
  createParagraph("Stephen A. Edwards is a tenured associate professor
      in the Computer Science Department of Columbia University. He
      obtained his Ph.D from the University of California, Berkeley in
      1997, his MS from Berkeley in 1994, and his BS from the
      California Institute of Technology in 1992, all in Electrical
      Engineering. Before pursuing his academic career in 2001, he
      worked for two Electronic Design Automation (EDA) companies,
      Simplex Solutions, now part of Cadence, and Synopsys.");
  createParagraph("Professor Edwards and his group explore automating
      the creation of software for embedded systems:
      application-specific computers hiding in a growing number of
      industrial and consumer systems. They have developed numerous
      compilation techniques for the Esterel synchronous language for
      real-time control and are also developing domain-specific
      languages for device drivers and communication protocols.");
  createSubheader("Courses Taught");
  createList("Programming Languages and Translators");
  createList("Parallel Functional Programming");
  createList("Fundamentals of Computer Systems");
  createList("Embedded System Design");
```

```
  output 0;
}
```

In contrast, the following program in HTML would require users to manually populate and remove elements from the tags:

```
<h1>Stephen A. Edwards</h1>
<img src='./edwards.jpeg'>
<h2>Contact Information</h2>
<span>sedwards@cs.columbia.edu</span><br>
<span>+1 212 939 7019</span><br>
<span>+1 212 666 0140 (fax)</span><br>
<span>462 Computer Science Building</span><br>
<span>1214 Amsterdam Ave MC 0401 New York, NY 10027-7003</span><br>
<h2>About</h2>
<p>Stephen A. Edwards is a tenured associate professor in the Computer
    Science Department of Columbia University. He obtained his Ph.D
    from the University of California, Berkeley in 1997, his MS from
    Berkeley in 1994, and his BS from the California Institute of
    Technology in 1992, all in Electrical Engineering. Before pursuing
    his academic career in 2001, he worked for two Electronic Design
    Automation (EDA) companies, Simplex Solutions, now part of Cadence,
    and Synopsys.</p>
<p>Professor Edwards and his group explore automating the creation of
    software for embedded systems: application-specific computers
    hiding in a growing number of industrial and consumer systems. They
    have developed numerous compilation techniques for the Esterel
    synchronous language for real-time control and are also developing
    domain-specific languages for device drivers and communication
    protocols.</p>
<h2>Courses Taught</h2>
<li>Programming Languages and Translators</li>
<li>Parallel Functional Programming</li>
<li>Fundamentals of Computer Systems</li>
<li>Embedded System Design</li>
```

The program above will look like this in a web browser:

Stephen A. Edwards

Contact Information

sedwards@cs.columbia.edu
+1 212 939 7019
+1 212 666 0140 (fax)
462 Computer Science Building
1214 Amsterdam Ave MC 0401 New York, NY 10027-7003

About

Stephen A. Edwards is a tenured associate professor in the Computer Science Department of Columbia University. He obtained his Ph.D from the University of California, Berkeley in 1997, his MS from Berkeley in 1994, and his BS from the California Institute of Technology in 1992, all in Electrical Engineering. Before pursuing his academic career in 2001, he worked for two Electronic Design Automation (EDA) companies, Simplex Solutions, now part of Cadence, and Synopsys.

Professor Edwards and his group explore automating the creation of software for embedded systems: application-specific computers hiding in a growing number of industrial and consumer systems. They have developed numerous compilation techniques for the Esterel synchronous language for real-time control and are also developing domain-specific languages for device drivers and communication protocols.

Courses Taught

- Programming Languages and Translators
- Parallel Functional Programming
- Fundamentals of Computer Systems
- Embedded System Design

Figure 1: Image of the Bio Portfolio Website

6.2 Greatest Common Divisor Website

```
function int gcd(int a, int b){
  REPEAT until (a != b){
    if (a > b) a = a - b;
    otherwise b = b - a;
  }
  output a;
}

function int main(){
  createHeader("Demo Program");
  createSubheader("Greatest Common Divisor");
  createParagraph("The gcd function returns the greatest common divisor
      of two or more integers. The greatest common divisor is the
      largest integer that goes into all supplied numbers without a
      remainder. For example, gcd(60,36) returns 12.");
  createImage("./demo.png");
  createList("gcd(2,14)");
  print(gcd(2,14));
  createList("gcd(3,15)");
  print(gcd(3,15));
  createList("gcd(99,121)");
  print(gcd(99,121));
  output 0;
}
```

In contrast, the following program in HTML would require users to manually populate and remove elements from the tags:

```
<h1>Demo Program</h1>
<h2>Greatest Common Divisor</h2>
<p>The gcd function returns the greatest common divisor of two or more
    integers. The greatest common divisor is the largest integer that
    goes into all supplied numbers without a remainder. For example,
    gcd(60,36) returns 12.</p>
<img src='./demo.png'>
<li>gcd(2,14)</li>
2<br>
<li>gcd(3,15)</li>
3<br>
<li>gcd(99,121)</li>
11<br>
```

The program above will look like this in a web browser:

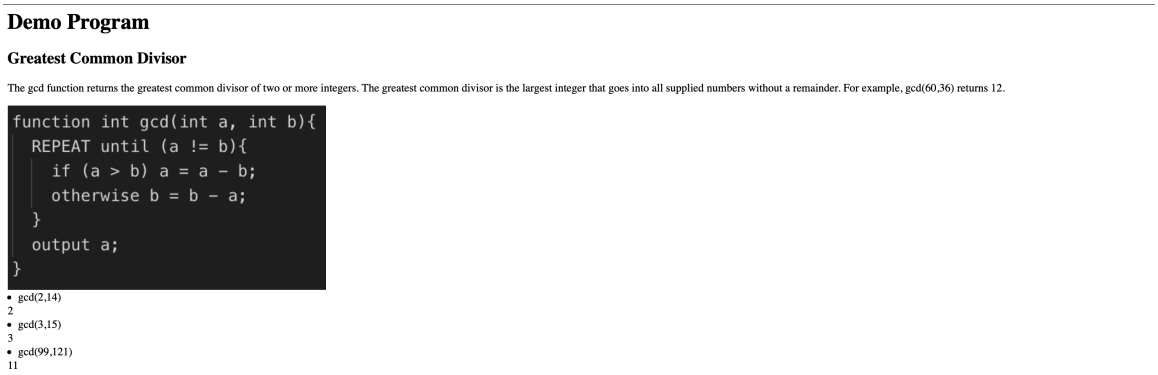

Figure 2: Image of the Greatest Common Divisor Website

7 Project Plan

7.1 Our Developmental Process

7.1.1 Planning

Seeing as this was the first time for all of us in both creating a language using OCaml, we created initial goals for our development but felt free to change them as we progress throughout the process. We very much followed an iterative process of planning by making sure to look back at past ideas and think about important questions about implementation and time constraints.

7.1.2 Specifications

Throughout our process, there was definitely a lot of change regarding specifications that we were going to implement within our language. Initially, we were planning on having our language include static scoping, weakly typed, object-oriented programming, and specific classes for web programming language. However, as we learned about these features and started to build our language, we proceeded to change or completely remove features which we published in our LRM. After creating our LRM, we had to also revisit those specifications and determine which of these features were feasible given our time constraints.

7.1.3 Development

In terms of development, we for the most followed the stages of the compiler architecture as in constructing the scanner first and completing the code generator last. Seeing as most of these components depended on each other, we could not necessarily work on two different aspects of the compiler at the same time. This meant that the pace at which we proceeded with the development of our language really depended on the speed at which we finished the earlier aspects of the compiler.

7.1.4 Testing

In order to test our language, we created test programs that tested key features of our language such as createHeader as they are the most likely to be used within our program. We tried to make our testing programs within our test suite very specific to each of these functions in order to see whether our compiler will both understand and output the correct HTML file. There are also tests within our test suite that combined multiple commands and syntax to see whether our compiler is able to parse and understand multiple web-related commands as most web pages consists of a layering of a lot of the commands that we have provide.

7.2 Software Development

In order to build the compiler, we used Ocaml version 4.12.0, Ocamlyacc, and Ocamllex. We all collectively chose to use Microsoft Visual Studio Code for our development environment.

7.3 Style Guide

We followed OCaml editing and formatting style.

7.4 Roles and Responsibilities

Name	Roles and Responsibilities
Xabier Peralta	Tester
Tamanna Hussain	Manager
Ramisa Murshed	Language Guru
Kamrul Hossain	Systems Architect

7.5 Project Timeline

Date	Milestone
2/03/21	Proposal
2/24/21	LRM and Parser Completed
3/24/21	Hello World
4/26/21	Project Report Due

7.6 Project Log

This project log shows a history of 62 commits starting from February 24th and ending April 26th.

```
commit 67f889a6f9440c04b6ab04b15f99cda67ad152b9
Author: Kamrul Hossain <kam.hossain5@gmail.com>
Date:  Mon Apr 26 13:10:54 2021 -0400

    added demo programs

commit f7410d13c50f5fc9096c3aa52ff0b05833225b8d
Author: Kamrul Hossain <kam.hossain5@gmail.com>
Date:  Mon Apr 26 10:46:08 2021 -0400

    Added image html

commit 462ba3ef4c486d1aafffee7fe26e344008c6e74e
Author: Kamrul Hossain <kam.hossain5@gmail.com>
Date:  Sun Apr 25 23:22:10 2021 -0400

    Added breaks to test outputs

commit c1c2ad3f8c8ee1564fca5a2a69ff7da52c827f17
Author: Kamrul Hossain <kam.hossain5@gmail.com>
Date:  Sat Apr 24 21:25:29 2021 -0400

    Edited test files

commit 005b1654473ee88b1694a449e1f8a455cd2c2c72
```

```
Author: Kamrul Hossain <kam.hossain5@gmail.com>
Date:  Sat Apr 24 13:21:30 2021 -0400

    Edited test files

commit 3d17047dbf6c0c4f6614d77604f7f316abfb26c8
Author: Kamrul Hossain <kam.hossain5@gmail.com>
Date:  Fri Apr 23 17:39:41 2021 -0400

    Modified codegen

commit 934d644500bc4c347bef79f7449f6ea6ad6e998f
Author: Kamrul Hossain <kam.hossain5@gmail.com>
Date:  Wed Apr 21 22:52:12 2021 -0400

    Updated codebase with html functions

commit a9179e84de237300ca569495427fadc46fc711d4
Author: TamannaH <thussain9997@bths.edu>
Date:  Tue Apr 20 18:42:28 2021 -0400

    Updated codegen.ml

commit 028087fab3c299e808db7b4cc3b1504bd0bd1c0a
Author: Kamrul Hossain <kam.hossain5@gmail.com>
Date:  Mon Apr 19 23:27:52 2021 -0400

    Updated testall.sh

commit c24f1f8cb4b85c0ec14499949bb2e790575bb5e6
Author: TamannaH <thussain9997@bths.edu>
Date:  Mon Apr 19 18:40:46 2021 -0400

    Updated testall.sh

commit 899dad24ecd6cea8533480de7ccc2160e3bfdcb1
Author: TamannaH <thussain9997@bths.edu>
Date:  Mon Apr 19 18:40:03 2021 -0400

    Updated scanner.mll

commit 58001ee7ef0e28b9d699592ae068ac6f3340cb6d
Author: TamannaH <thussain9997@bths.edu>
Date:  Mon Apr 19 18:39:30 2021 -0400

    Updated sast.ml

commit 63ebcb881f52039b40c0b5ea909384c7ca0075a0
Author: TamannaH <thussain9997@bths.edu>
Date:  Mon Apr 19 18:39:03 2021 -0400
```

```
76
77      Updated qwebparse.mly
78
79  commit a29329a7ba840d7f23f128b5957f9d142ebc51df
80  Author: TamannaH <thussain9997@bths.edu>
81  Date:   Mon Apr 19 18:38:39 2021 -0400
82
83      Delete parser.output
84
85  commit bd4655217123a89d3f99af2b8801eb4330bc777c
86  Author: TamannaH <thussain9997@bths.edu>
87  Date:   Mon Apr 19 18:38:29 2021 -0400
88
89      Deleted parser.mly
90
91  commit 94da787962f2ad885c64c7ba07625c45861c6386
92  Author: TamannaH <thussain9997@bths.edu>
93  Date:   Mon Apr 19 18:37:57 2021 -0400
94
95      Updated codegen.ml
96
97  commit f25ba85dfa8205aec96e37d16346dbce88ae859a
98  Author: TamannaH <thussain9997@bths.edu>
99  Date:   Mon Apr 19 18:37:51 2021 -0400
100
101     Updated codegen.ml
102
103 commit d0b0ef4042de322f0dfe22aea8e076cb2d8a3759
104 Author: TamannaH <thussain9997@bths.edu>
105 Date:   Sat Apr 17 15:44:41 2021 -0400
106
107     Updated test-hello.qwb
108
109 commit b2cb1ed31de27d1b6cd45215d1e0eadeb25e6414
110 Author: TamannaH <thussain9997@bths.edu>
111 Date:   Sat Apr 17 15:44:16 2021 -0400
112
113     Updated testall.sh
114
115 commit 22a673337a0d09560fa0d6a3c4d9495c692aecea
116 Author: TamannaH <thussain9997@bths.edu>
117 Date:   Sat Apr 17 15:43:50 2021 -0400
118
119     Updated semant.ml
120
121 commit 90f718b681c89d681a59c3f40f2b87658c55da21
122 Author: TamannaH <thussain9997@bths.edu>
123 Date:   Sat Apr 17 15:43:29 2021 -0400
124
125     Updated scanner.mll
```

```
126
127 commit 17d40c123e65256f2beba282f847f88b6c4df0f0
128 Author: TamannaH <thussain9997@bths.edu>
129 Date:  Sat Apr 17 15:43:28 2021 -0400
130
131     Updated scanner.mll
132
133 commit 963f88f150364764338ff6f777126060f57df305
134 Author: TamannaH <thussain9997@bths.edu>
135 Date:  Sat Apr 17 15:43:00 2021 -0400
136
137     Updated sast.ml
138
139 commit 762aab7e2daf02a5dd52debd2db18ee0bdac2f71
140 Author: TamannaH <thussain9997@bths.edu>
141 Date:  Sat Apr 17 15:42:37 2021 -0400
142
143     Updated qwebparse.mly
144
145 commit 7f46444d78eebc3b576c64273e20007cb8b28cf0
146 Author: TamannaH <thussain9997@bths.edu>
147 Date:  Sat Apr 17 15:41:32 2021 -0400
148
149     Updated Makefile
150
151 commit 5c93526282f2854a5cda709fc85bd043a6fb8be6
152 Author: TamannaH <thussain9997@bths.edu>
153 Date:  Sat Apr 17 15:41:03 2021 -0400
154
155     Updated codegen.ml
156
157 commit f228019fe0f80a8636750e649e728d38e9bdb012
158 Author: TamannaH <thussain9997@bths.edu>
159 Date:  Sat Apr 17 15:40:39 2021 -0400
160
161     Updated ast.ml
162
163 commit a201db1f4e878fcb22bcc7bb060ac9fd30fd6315
164 Author: TamannaH <thussain9997@bths.edu>
165 Date:  Wed Apr 14 20:47:44 2021 -0400
166
167     Updated codegen.ml
168
169 commit 045b47348a731d0361c81469875efaf205655867
170 Author: TamannaH <thussain9997@bths.edu>
171 Date:  Wed Apr 14 20:46:45 2021 -0400
172
173     Updated sast.ml
174
175 commit 7c3978484c2b8d52eef9c85a1fb21de584f7a0b9
```

```
Author: TamannaH <thussain9997@bths.edu>
Date:  Wed Apr 14 20:46:00 2021 -0400

    Updated scanner.mll

commit 3db2d6373bc1445bd9457cfd367529c05c99ec64
Author: TamannaH <thussain9997@bths.edu>
Date:  Wed Apr 14 20:45:33 2021 -0400

    Updated ast.ml

commit 2a0cddc45293b869995be47a02976f99fde923d7
Author: TamannaH <thussain9997@bths.edu>
Date:  Wed Apr 14 20:44:33 2021 -0400

    Commented out parts of parser

commit 6313ee83707271bcef0e5ff8d922ddf6ff7da4d2
Author: Kamrul Hossain <kam.hossain5@gmail.com>
Date:  Tue Apr 13 15:42:19 2021 -0400

    Modified qwebparse and scanner

commit 35790a7030db76bd2aaff4e87f7590f66a854f51
Author: TamannaH <thussain9997@bths.edu>
Date:  Mon Apr 12 21:46:39 2021 -0400

    Added file to tests

commit 1b007e452d1ec318e049231d61193933790720d1
Author: TamannaH <thussain9997@bths.edu>
Date:  Mon Apr 12 21:45:51 2021 -0400

    Created tests folder

commit 856efa4f947e926e44e0d1b1e001219bcaafee16
Author: TamannaH <thussain9997@bths.edu>
Date:  Sat Apr 10 22:41:57 2021 -0400

    Updated sast.ml

commit 784807aa005667cf30a3700e4979d3ac2205f59d
Author: TamannaH <thussain9997@bths.edu>
Date:  Sat Apr 10 22:41:19 2021 -0400

    Updated scanner.mll

commit c5b8dcecbdef780c097d0a5ad20a71dab546a580
Author: TamannaH <thussain9997@bths.edu>
Date:  Sat Apr 10 22:40:30 2021 -0400
```

```
226
227     Updated qweb.ml
228
229 commit fa6d58f5f664d6935fde3627458ecc01710f9c6e
230 Author: TamannaH <thussain9997@bths.edu>
231 Date:   Sat Apr 10 22:39:35 2021 -0400
232
233     Uploaded qwebparse.mly
234
235 commit 47903189e46a574e83ad5e382f5432d2d5fa061b
236 Author: TamannaH <thussain9997@bths.edu>
237 Date:   Sat Apr 10 22:33:56 2021 -0400
238
239     Updated ast.ml
240
241 commit 5ea9bb5515991b3d9d6fc1cfe2f14828df1558a8
242 Author: TamannaH <thussain9997@bths.edu>
243 Date:   Sat Apr 10 22:33:19 2021 -0400
244
245     Updated codegen.ml
246
247 commit be68eb73de1e59bfe11e3e89575f2ab097da66a1
248 Author: TamannaH <thussain9997@bths.edu>
249 Date:   Sat Apr 10 22:31:10 2021 -0400
250
251     Updated semant.ml
252
253 commit 173f4df796ca3b8840717ebdf4ec5394d61398ca
254 Author: TamannaH <thussain9997@bths.edu>
255 Date:   Wed Apr 7 20:31:39 2021 -0400
256
257     Minor change to str_format_str
258
259 commit bd25f71cd8e733a09a2bde4941159c5d378eefe0
260 Author: Kamrul Hossain <kam.hossain5@gmail.com>
261 Date:   Wed Apr 7 19:25:36 2021 -0400
262
263     Added basic functionalities in codegen file
264
265 commit 79442b2af8d43e68222d641711d5915a378c40a1
266 Author: TamannaH <thussain9997@bths.edu>
267 Date:   Tue Apr 6 00:17:53 2021 -0400
268
269     Added code for print string
270
271 commit 32269ace4fcd73ffa420368e6886b519d28f7f07
272 Author: TamannaH <thussain9997@bths.edu>
273 Date:   Tue Apr 6 00:16:41 2021 -0400
274
275     Added code for print string
```

```
commit 1378dfe0e99fd5f8587bbb3ebc9bcbf4b239ea91
Author: TamannaH <thussain9997@bths.edu>
Date:  Wed Mar 24 20:56:44 2021 -0400

    Updated codegen.ml

commit 6316fc06e261904de6548ca93918682068b7edd3
Author: TamannaH <thussain9997@bths.edu>
Date:  Wed Mar 24 20:49:53 2021 -0400

    Removed extra types

commit f2f43ec8f3cb23228240e239f648d0eb5ca2f57b
Author: TamannaH <thussain9997@bths.edu>
Date:  Wed Mar 24 20:48:32 2021 -0400

    Removed comment

commit e63e7fabb215e86dd125886fdcc594c82b951bd3
Author: TamannaH <thussain9997@bths.edu>
Date:  Wed Mar 24 20:45:07 2021 -0400

    Updated qweb.ml

commit e6e26ff13223e28d036f3a0f0a117beecaf69d8a
Author: TamannaH <thussain9997@bths.edu>
Date:  Wed Mar 24 20:41:55 2021 -0400

    Added initial version of semant.ml

commit e4ba5150d30b5a283eb532e842e9e7586009a301
Author: Ramisa Murshed <35822743+ramisamurshed@users.noreply.github.com>
Date:  Wed Mar 24 17:37:45 2021 -0400

    initial commit

commit 1fb0fbdabc364a0f6309c233c69456b677d49d1f
Author: TamannaH <thussain9997@bths.edu>
Date:  Tue Mar 23 17:28:45 2021 -0400

    Initial versions

commit 55f64d0b84b88960723ac05b16043d0eb6eee036
Author: Kamrul Hossain <kam.hossain5@gmail.com>
Date:  Tue Mar 23 15:32:48 2021 -0400

    Added sast.ml

commit 61dae224f6276ae465e5a99828aca31fe03fdebb
```

```
Author: TamannaH <thussain9997@bths.edu>
Date:  Sat Mar 20 20:37:23 2021 -0400

    Edited func_decl and removed semicolons

commit 8a1c900f2d5754c7840226705fe734ac99cb6a56
Author: TamannaH <thussain9997@bths.edu>
Date:  Sat Mar 20 20:35:24 2021 -0400

    Edited typ and literal

commit 8e1a2cf1fc140e17007bcb2599c0d980756b01ef
Author: TamannaH <thussain9997@bths.edu>
Date:  Fri Mar 19 21:37:55 2021 -0400

    Added ast.ml

commit b9687a3e35644e7437e56dd23a51f44e531c21f5
Author: Kamrul Hossain <kam.hossain5@gmail.com>
Date:  Wed Mar 17 19:40:51 2021 -0400

    Add files via upload

commit c0d55fe284439f3d19ebbc5a945f078af5221e95
Author: Kamrul Hossain <kam.hossain5@gmail.com>
Date:  Wed Feb 24 20:42:33 2021 -0500

    Ran ocamlyacc output

commit cfec1d562c264b063a6e1b0dc6f017a77a92d840
Author: Kamrul Hossain <kam.hossain5@gmail.com>
Date:  Wed Feb 24 20:40:47 2021 -0500

    Updated literal and types

commit 91e7c06f1073dadbeccb64305eb8992874061f5e
Author: Kamrul Hossain <kam.hossain5@gmail.com>
Date:  Wed Feb 24 20:14:53 2021 -0500

    Added Parser

commit a818035630179ada9aa5b07db664ca4949d478ca
Author: Kamrul Hossain <kam.hossain5@gmail.com>
Date:  Wed Feb 24 20:13:09 2021 -0500

    Initial commit
```

8 Architectural Design

8.1 Block Diagram

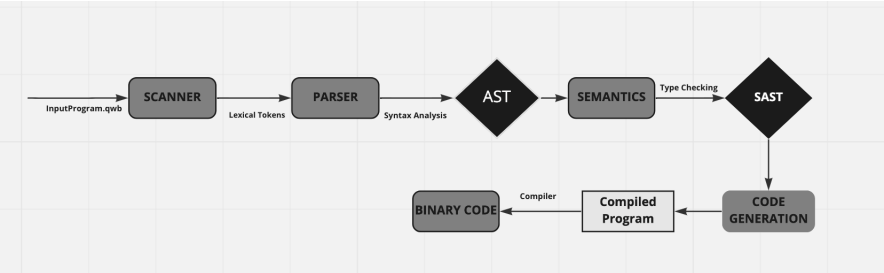

Figure 3: Architecture of QWEB Compiler.

8.2 Scanner

The scanner takes in a QWEB program and then proceeds to create tokens for keywords, identifiers, operators, and values as specified by QWEB's lexical conventions.

8.3 Parser and Semantic Checker

After the Scanner generates tokens based on the provided input file, the parser than takes in the recently created tokens and created an AST (Abstract Syntax Tree). The Semantic Checker then proceeds to recursively traverses this newly created AST and transform it into a SAST (Extended Abstract Syntax Tree).

8.4 Code Generation

QWEB's code generator uses post-order format to go through the SAST and generate code.

9 Testing

We wrote a test suite to test the functionality of our language as we implemented it. The test suite ensured that we did not break previously working features when we added new ones.

All of the tests should produce the correct output following the naming pattern of using test-*.qwb for the test program and then using the same name but different file extension of the form test-*.out for the expected output of the program. Similarly, all of the negative tests used fail-*.qwb for the test program

that should fail and used fail-*.err instead of .out for the expected error message.

A complete set of tests can be found in Appendix 11.3 Test Files.
These tests were implemented mostly in parallel with the functionality they tested and so they were chosen to test the implemented functionality and failure cases of the implemented functionality.

10 Lessons Learned

10.1 Xabier Peralta

I think that creating this project during a pandemic really taught me about how important effective communication is as everybody situation can be really tough so as a team, it is important to be mindful of that.

10.2 Tamanna Hussain

Through this project, I learned the importance of meeting project deadlines and creating realistic internal deadlines to slowly implement features. Our group had a difficult time completing the *Hello World* milestone, which made it even harder to add more features into our language. I would definitely recommend that future teams start working on the deliverable early on and complete the scanner/parser as soon as possible. Additionally, it is important for everyone in the team to hold each other accountable and to be transparent about their workloads/schedules in the beginning of the semester.

10.3 Ramisa Murshed

Working on this project gave me a better understanding of all of the complex pieces that go into programming languages. This project was my first time programming in OCaml, which I found extremely challenging at first, but learned to appreciate it a lot more as we got towards the end of our project. It was extremely interesting to see how much work went into implementing certain features, such as having function declarations denoted with a colon rather than a curly brace. As someone who works on a lot of frontend development, it was also fun to create a language that automates the process of adding common HTML elements to a webpage. Some advice I'd give to teams working on similar projects is to make sure you are able to have an organized schedule and set deadlines to reach smaller milestones even before the larger milestones are due, which requires good communication between team members.

10.4 Kamrul Hossain

I believe this project has been such a large and fruitful learning experience. I was exposed to more of functional programming and I gained knowledge in building a compiler using OCaml. I have always been passionate about web

development and creating a language that made it easier to develop websites has been exciting. My best piece of advice to future teams is to ensure that everyone is organized/communicative and make sure that all teammates are putting their equal share in the project.

11 Appendix

11.1 Optimization Files

11.1.1 Makefile

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

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

.PHONY : all
all : qweb.native

qweb.native :
 opam config exec -- \
 rm -f *.o
 ocamlbuild -use-ocamlfind -pkgs llvm.bitreader qweb.native
 gcc -c
 clang -emit-llvm -o -c -Wno-varargs

# "make clean" removes all generated files

.PHONY : clean
clean :
 ocamlbuild -clean
 rm -rf testall.log *.diff qweb scanner.ml parser.ml parser.mli
 rm -rf *.cmx *.cmi *.cmo *.cmx *.ll *.html
 rm -rf *.err *.out *.exe *.s
 rm -f *.o

# Testing the "printbig" example

# 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-%.qwb) $(TESTS:%=test-%.out) \
    $(FAILS:%=fail-%.qwb) $(FAILS:%=fail-%.err)

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

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

11.1.2 testall.sh

```
#!/bin/sh

# Regression testing script for QWEB
# 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 QWEB compiler. Usually "./qweb.native"
# Try "_build/qweb.native" if ocamlbuild was unable to create a symbolic
    link.
```

```
QWEB="./qweb.native"
#QWEB="_build/qweb.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] [.qwb 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
    }
}

```

```
71 # RunFail <args>
72 # Report the command, run it, and expect an error
73 RunFail() {
74     echo $* 1>&2
75     eval $* && {
76  SignalError "failed: $* did not report an error"
77  return 1
78     }
79     return 0
80 }
81
82 Check() {
83     error=0
84     basename=`echo $1 | sed 's/.*\\///
85                             s/.qwb//'`
86     reffile=`echo $1 | sed 's/.qwb$//'`
87     basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."
88
89     echo -n "$basename..."
90
91     echo 1>&2
92     echo "###### Testing $basename" 1>&2
93
94     generatedfiles=""
95
96     generatedfiles="$generatedfiles ${basename}.ll ${basename}.s
           ${basename}.exe ${basename}.out ${basename}.html" &&
97     Run "$QWEB" "$1" ">" "${basename}.ll" &&
98     Run "$LLC" "-relocation-model=pic" "${basename}.ll" ">"
           "${basename}.s" &&
99     Run "$CC" "-o" "${basename}.exe" "${basename}.s" &&
100    Run "./${basename}.exe" > "${basename}.out" &&
101    Run "./${basename}.exe" > "${basename}.html" &&
102    Compare ${basename}.out ${reffile}.out ${basename}.diff
103
104    # Report the status and clean up the generated files
105
106    if [ $error -eq 0 ] ; then
107 # if [ $keep -eq 0 ] ; then
108 #    rm -f $generatedfiles
109 # fi
110 echo "OK"
111 echo "###### SUCCESS" 1>&2
112    else
113 echo "###### FAILED" 1>&2
114 globalerror=$error
115    fi
116 }
117
118 CheckFail() {
```

```
119     error=0
120     basename=‘echo $1 | sed ’s/.*\\///
121                             s/.qwb//’‘
122     reffile=‘echo $1 | sed ’s/.qwb$//’‘
123     basedir="‘echo $1 | sed ’s/\/[^\/]*$//’‘/."
124
125     echo -n "$basename..."
126
127     echo 1>&2
128     echo "###### Testing $basename" 1>&2
129
130     generatedfiles=""
131
132     generatedfiles="$generatedfiles ${basename}.err ${basename}.diff" &&
133     RunFail "$QWEB" "<" $1 "2>" "${basename}.err" ">>" $globallog &&
134     Compare ${basename}.err ${reffile}.err ${basename}.diff
135
136     # Report the status and clean up the generated files
137
138     if [ $error -eq 0 ] ; then
139  if [ $keep -eq 0 ] ; then
140      rm -f $generatedfiles
141  fi
142  echo "OK"
143  echo "###### SUCCESS" 1>&2
144     else
145  echo "###### FAILED" 1>&2
146  globalerror=$error
147     fi
148 }
149
150 while getopts kdpsh c; do
151     case $c in
152  k) # Keep intermediate files
153      keep=1
154      ;;
155  h) # Help
156      Usage
157      ;;
158     esac
159 done
160
161 shift ‘expr $OPTIND - 1‘
162
163 LLIFail() {
164   echo "Could not find the LLVM interpreter \"$LLI\"."
165   echo "Check your LLVM installation and/or modify the LLI variable in
          testall.sh"
166   exit 1
167 }
```

```
168
169 which "$LLI" >> $globallog || LLIFail
170
171 # if [ ! -f printbig.o ]
172 # then
173 #     echo "Could not find printbig.o"
174 #     echo "Try \"make printbig.o\""
175 #     exit 1
176 # fi
177
178 if [ $# -ge 1 ]
179 then
180     files=$@
181 else
182     files="tests/test-*.qwb tests/fail-*.qwb"
183 fi
184
185 for file in $files
186 do
187     case $file in
188  *test-*)
189      Check $file 2>> $globallog
190      ;;
191  *fail-*)
192      CheckFail $file 2>> $globallog
193      ;;
194  *)
195      echo "unknown file type $file"
196      globalerror=1
197      ;;
198     esac
199 done
200
201 exit $globalerror
```

11.2 Source Files

11.2.1 ast.ml

```
1 (* Abstract Syntax Tree *)
2
3 type op = Add | Sub | Mult | Div | Equal | Neq | Less | Leq | Greater |
      Geq |
4           And | Or
5
6 type uop = Neg | Not
7
```

```
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
  (* | Seq of expr list *)
  | Noexpr

type stmt =
    Block of stmt list
  | Expr of expr
  | Output of expr
  | If of expr * stmt * stmt
  | For of expr * expr * expr * stmt
  | Repeat 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 -> "||"
```

```
58
59  let string_of_uop = function
60      Neg -> "-"
61    | Not -> "!"
62
63  let rec string_of_expr = function
64      Literal(l) -> string_of_int l
65    | Fliteral(l) -> l
66    | StringLit s -> "\"" ^ s ^ "\""
67    | BoolLit(true) -> "true"
68    | BoolLit(false) -> "false"
69    | Id(s) -> s
70    | Binop(e1, o, e2) ->
71        string_of_expr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_expr e2
72    | Unop(o, e) -> string_of_uop o ^ string_of_expr e
73    | Assign(v, e) -> v ^ " = " ^ string_of_expr e
74    | Call(f, el) ->
75      f ^ "(" ^ String.concat ", " (List.map string_of_expr el) ^ ")"
76    (* | Seq(a) -> string_of_list a *)
77    | Noexpr -> ""
78    and string_of_list = function
79        l -> "[" ^ (string_of_seq l) ^ "]"
80    and string_of_seq = function
81        x :: y :: a -> string_of_expr x ^ ", " ^ string_of_seq (y :: a)
82      | x :: _ -> string_of_expr x
83      | [] -> ""
84
85  let rec string_of_stmt = function
86      Block(stmts) ->
87        "{\n" ^ String.concat "" (List.map string_of_stmt stmts) ^ "}\n"
88    | Expr(expr) -> string_of_expr expr ^ ";\n";
89    | Output(expr) -> "output " ^ string_of_expr expr ^ ";\n";
90    | If(e, s, Block([])) -> "if (" ^ string_of_expr e ^ ")\n" ^
          string_of_stmt s
91    | If(e, s1, s2) -> "if (" ^ string_of_expr e ^ ")\n" ^
92        string_of_stmt s1 ^ "otherwise\n" ^ string_of_stmt s2
93    | For(e1, e2, e3, s) ->
94        "FOR each (" ^ string_of_expr e1 ^ " ; " ^ string_of_expr e2 ^ " ;
              " ^
95        string_of_expr e3 ^ ") " ^ string_of_stmt s
96    | Repeat(e, s) -> "REPEAT until (" ^ string_of_expr e ^ ") " ^
          string_of_stmt s
97
98  let rec string_of_typ = function
99      Int -> "int"
100   | Bool -> "bool"
101   | Float -> "float"
102   | Void -> "void"
103   | String -> "str"
104   | List(t) -> "List(" ^ string_of_typ t ^ ")"
```

```
105
106 let string_of_vdecl (t, id) = string_of_typ t ^ " " ^ id ^ ";\n"
107
108 let string_of_fdecl fdecl =
109   "function " ^ string_of_typ fdecl.typ ^ " " ^
110   fdecl.fname ^ "(" ^ String.concat ", " (List.map snd fdecl.formals) ^
111   ")\n:\n" ^
112   String.concat "" (List.map string_of_vdecl fdecl.locals) ^
113   String.concat "" (List.map string_of_stmt fdecl.body) ^
114   "\n" ^ "end"
115
116 let string_of_program (vars, funcs) =
117   String.concat "" (List.map string_of_vdecl vars) ^ "\n" ^
118   String.concat "\n" (List.map string_of_fdecl funcs)
```

11.2.2 codegen.ml

```
1  (* Code generation: translate takes a semantically checked AST and
2  produces LLVM IR
3
4  LLVM tutorial: Make sure to read the OCaml version of the tutorial
5
6  http://llvm.org/docs/tutorial/index.html
7
8  Detailed documentation on the OCaml LLVM library:
9
10 http://llvm.moe/
11 http://llvm.moe/ocaml/
12
13 *)
14
15 module L = Llvm
16 module A = Ast
17 open Sast
18
19 module StringMap = Map.Make(String)
20
21 (* translate : Sast.program -> Llvm.module *)
22 let translate (globals, functions) =
23   let context   = L.global_context () in
24
25   (* Create the LLVM compilation module into which
26      we will generate code *)
27   let the_module = L.create_module context "QWEB" in
28
29   (* Get types from the context *)
30   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
  in

(* Return the LLVM type for a QWEB type *)
let rec 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(t) -> L.pointer_type (ltype_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

(* let printbig_t : L.lltype =
  L.function_type i32_t [| i32_t |] in
let printbig_func : L.llvalue =
  L.declare_function "printbig" printbig_t the_module in *)

(* let createHeader_t : L.lltype =
  L.var_arg_function_type i32_t [| L.pointer_type i8_t |] in
let createHeader_func : L.llvalue =
  L.declare_function "createHeader" createHeader_t the_module in *)

(* 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)
```

```
79      in let ftype = L.function_type (ltype_of_typ fdecl.styp)
            formal_types in
80      StringMap.add name (L.define_function name ftype the_module,
            fdecl) m in
81    List.fold_left function_decl StringMap.empty functions in
82
83  (* Fill in the body of the given function *)
84  let build_function_body fdecl =
85    let (the_function, _) = StringMap.find fdecl.sfname function_decls in
86    let builder = L.builder_at_end context (L.entry_block the_function)
          in
87
88    let int_format_str = L.build_global_stringptr "%d<br>\n" "fmt"
          builder
89    and string_format_str = L.build_global_stringptr
          "<span>%s</span><br>\n" "fmt" builder
90    and header_format_str = L.build_global_stringptr "<h1>%s</h1>\n"
          "fmt" builder
91    and subheader_format_str = L.build_global_stringptr "<h2>%s</h2>\n"
          "fmt" builder
92    and paragraph_format_str = L.build_global_stringptr "<p>%s</p>\n"
          "fmt" builder
93    and image_format_str = L.build_global_stringptr "<img src='%s'>\n"
          "fmt" builder
94    and list_format_str = L.build_global_stringptr "<li>%s</li>\n" "fmt"
          builder
95    and float_format_str = L.build_global_stringptr "%g<br>\n" "fmt"
          builder in
96
97    (* Construct the function's "locals": formal arguments and locally
98       declared variables. Allocate each on the stack, initialize their
99       value, if appropriate, and remember their values in the "locals"
             map *)
100   let local_vars =
101     let add_formal m (t, n) p =
102       L.set_value_name n p;
103       let local = L.build_alloca (ltype_of_typ t) n builder in
104       ignore (L.build_store p local builder);
105       StringMap.add n local m
106
107     (* Allocate space for any locally declared variables and add the
108      * resulting registers to our map *)
109     and add_local m (t, n) =
110       let local_var = L.build_alloca (ltype_of_typ t) n builder
111       in StringMap.add n local_var m
112     in
113
114     let formals = List.fold_left2 add_formal StringMap.empty
            fdecl.sformals
115         (Array.to_list (L.params the_function)) in
```

```
116       List.fold_left add_local formals fdecl.slocals
117     in
118
119     (* Return the value for a variable or formal argument.
120        Check local names first, then global names *)
121     let lookup n = try StringMap.find n local_vars
122       with Not_found -> StringMap.find n global_vars
123     in
124
125     (* Construct code for an expression; return its value *)
126     let rec expr builder ((_, e) : sexpr) = match e with
127         SLiteral i -> L.const_int i32_t i
128       | SBoolLit b -> L.const_int i1_t (if b then 1 else 0)
129       | SStringLit s -> L.build_global_stringptr s "str" builder
130       | SFliteral l -> L.const_float_of_string float_t l
131       | SNoexpr     -> L.const_int i32_t 0
132       | SId s       -> L.build_load (lookup s) s builder
133       | SAssign (s, e) -> let e' = expr builder e in
134         ignore(L.build_store e' (lookup s) builder); e'
135       | SBinop ((A.Float,_ ) as e1, op, e2) ->
136         let e1' = expr builder e1
137         and e2' = expr builder e2 in
138         (match op with
139           A.Add     -> L.build_fadd
140         | A.Sub     -> L.build_fsub
141         | A.Mult    -> L.build_fmul
142         | A.Div     -> L.build_fdiv
143         | A.Equal   -> L.build_fcmp L.Fcmp.Oeq
144         | A.Neq     -> L.build_fcmp L.Fcmp.One
145         | A.Less    -> L.build_fcmp L.Fcmp.Olt
146         | A.Leq     -> L.build_fcmp L.Fcmp.Ole
147         | A.Greater -> L.build_fcmp L.Fcmp.Ogt
148         | A.Geq     -> L.build_fcmp L.Fcmp.Oge
149         | A.And | A.Or ->
150           raise (Failure "internal error: semant should have rejected
                and/or on float")
151       ) e1' e2' "tmp" builder
152       | SBinop (e1, op, e2) ->
153         let e1' = expr builder e1
154         and e2' = expr builder e2 in
155         (match op with
156           A.Add     -> L.build_add
157         | A.Sub     -> L.build_sub
158         | A.Mult    -> L.build_mul
159         | A.Div     -> L.build_sdiv
160         | A.And     -> L.build_and
161         | A.Or      -> L.build_or
162         | A.Equal   -> L.build_icmp L.Icmp.Eq
163         | A.Neq     -> L.build_icmp L.Icmp.Ne
164         | A.Less    -> L.build_icmp L.Icmp.Slt
```

```
165         | A.Leq    -> L.build_icmp L.Icmp.Sle
166         | A.Greater -> L.build_icmp L.Icmp.Sgt
167         | A.Geq    -> L.build_icmp L.Icmp.Sge
168       ) e1' e2' "tmp" builder
169     | SUnop(op, ((t, _) as e)) ->
170       let e' = expr builder e in
171       (match op with
172         A.Neg when t = A.Float -> L.build_fneg
173         | A.Neg                 -> L.build_neg
174         | A.Not                 -> L.build_not) e' "tmp" builder
175     | SCall ("print", [e]) | SCall ("printb", [e]) ->
176       L.build_call printf_func [| int_format_str ; (expr builder e) |]
              "printf" builder
177     (* | SCall ("printbig", [e]) ->
178       L.build_call printbig_func [| (expr builder e) |] "printbig"
              builder *)
179     | SCall ("prints", [e]) ->
180       L.build_call printf_func [| string_format_str ; (expr builder e)
              |] "printf" builder
181     | SCall ("createHeader", [e]) ->
182       L.build_call printf_func [| header_format_str ; (expr builder e)
              |] "printf" builder
183     | SCall ("createSubheader", [e]) ->
184       L.build_call printf_func [| subheader_format_str ; (expr builder
              e) |] "printf" builder
185     | SCall ("createParagraph", [e]) ->
186       L.build_call printf_func [| paragraph_format_str ; (expr builder
              e) |] "printf" builder
187     | SCall ("createImage", [e]) ->
188       L.build_call printf_func [| image_format_str ; (expr builder e)
              |] "printf" builder
189     | SCall ("createList", [e]) ->
190       L.build_call printf_func [| list_format_str ; (expr builder e)
              |] "printf" builder
191     | SCall ("printf", [e]) ->
192       L.build_call printf_func [| float_format_str ; (expr builder e)
              |] "printf" builder
193     | SCall (f, args) ->
194       let (fdef, fdecl) = StringMap.find f function_decls in
195       let llargs = List.rev (List.map (expr builder) (List.rev args))
              in
196       let result = (match fdecl.styp with
197           A.Void -> ""
198         | _ -> f ^ "_result") in
199       L.build_call fdef (Array.of_list llargs) result builder
200   in
201
202   (* LLVM insists each basic block end with exactly one "terminator"
203      instruction that transfers control. This function runs "instr
           builder"
```

```
204       if the current block does not already have a terminator. Used,
205       e.g., to handle the "fall off the end of the function" case. *)
206     let add_terminal builder instr =
207       match L.block_terminator (L.insertion_block builder) with
208         Some _ -> ()
209       | None -> ignore (instr builder) in
210
211     (* Build the code for the given statement; return the builder for
212       the statement's successor (i.e., the next instruction will be
              built
213       after the one generated by this call) *)
214
215     let rec stmt builder = function
216         SBlock sl -> List.fold_left stmt builder sl
217       | SExpr e -> ignore(expr builder e); builder
218       | SOutput e -> ignore(match fdecl.styp with
219           (* Special "return nothing" instr *)
220             A.Void -> L.build_ret_void builder
221           (* Build return statement *)
222           | _ -> L.build_ret (expr builder e) builder );
223         builder
224       | SIf (predicate, then_stmt, else_stmt) ->
225         let bool_val = expr builder predicate in
226         let merge_bb = L.append_block context "merge" the_function in
227         let build_br_merge = L.build_br merge_bb in (* partial function
              *)
228
229         let then_bb = L.append_block context "then" the_function in
230         add_terminal (stmt (L.builder_at_end context then_bb) then_stmt)
231           build_br_merge;
232
233         let else_bb = L.append_block context "else" the_function in
234         add_terminal (stmt (L.builder_at_end context else_bb) else_stmt)
235           build_br_merge;
236
237         ignore(L.build_cond_br bool_val then_bb else_bb builder);
238         L.builder_at_end context merge_bb
239
240       | SRepeat (predicate, body) ->
241         let pred_bb = L.append_block context "while" the_function in
242         ignore(L.build_br pred_bb builder);
243
244         let body_bb = L.append_block context "while_body" the_function in
245         add_terminal (stmt (L.builder_at_end context body_bb) body)
246           (L.build_br pred_bb);
247
248         let pred_builder = L.builder_at_end context pred_bb in
249         let bool_val = expr pred_builder predicate in
250
251         let merge_bb = L.append_block context "merge" the_function in
```

```
252         ignore(L.build_cond_br bool_val body_bb merge_bb pred_builder);
253         L.builder_at_end context merge_bb
254
255       (* Implement for loops as while loops *)
256       | SFor (e1, e2, e3, body) -> stmt builder
257                                    ( SBlock [SExpr e1 ; SRepeat (e2,
                                         SBlock [body ; SExpr e3]) ] )
258     in
259
260     (* Build the code for each statement in the function *)
261     let builder = stmt builder (SBlock fdecl.sbody) in
262
263     (* Add a return if the last block falls off the end *)
264     add_terminal builder (match fdecl.styp with
265           A.Void -> L.build_ret_void
266         | A.Float -> L.build_ret (L.const_float float_t 0.0)
267         | t -> L.build_ret (L.const_int (ltype_of_typ t) 0))
268   in
269
270   List.iter build_function_body functions;
271   the_module
```

11.2.3 qweb.ml

```
1  (* Top-level of the QWEB compiler: scan & parse the input,
2     check the resulting AST and generate an SAST from it, generate LLVM
          IR,
3     and dump the module *)
4
5  type action = Ast | Sast | LLVM_IR | Compile
6
7  let () =
8    let action = ref Compile in
9    let set_action a () = action := a in
10   let speclist = [
11     ("-a", Arg.Unit (set_action Ast), "Print the AST");
12     ("-s", Arg.Unit (set_action Sast), "Print the SAST");
13     ("-l", Arg.Unit (set_action LLVM_IR), "Print the generated LLVM IR");
14     ("-c", Arg.Unit (set_action Compile),
15       "Check and print the generated LLVM IR (default)");
16   ] in
17   let usage_msg = "usage: ./qweb.native [-a|-s|-l|-c] [file.qwb]" in
18   let channel = ref stdin in
19   Arg.parse speclist (fun filename -> channel := open_in filename)
          usage_msg;
20
21   let lexbuf = Lexing.from_channel !channel in
```

```
  let ast = Qwebparse.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)
```

11.2.4 qwebparse.mly

```
/* Ocamlyacc parser for QWEB */

%{
open Ast
%}

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

%start program
%type <Ast.program> program

%nonassoc NOOTHERWISE
%nonassoc OTHERWISE
%right ASSIGN
%left OR
%left AND
%left EQ NEQ
%left LT GT LEQ GEQ
%left PLUS MINUS
%left TIMES DIVIDE
%right NOT

%%

```

```
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             }
  | OUTPUT expr_opt SEMI                { Output $2           }
  | LBRACE stmt_list RBRACE                { Block(List.rev $2)  }
  | IF LPAREN expr RPAREN stmt %prec NOOTHERWISE { If($3, $5, Block([]))
      }
```

```
  | IF LPAREN expr RPAREN stmt OTHERWISE stmt { If($3, $5, $7)    }
  | FOR LPAREN expr_opt SEMI expr SEMI expr_opt RPAREN stmt
                                        { For($3, $5, $7, $9) }
  | REPEAT LPAREN expr RPAREN stmt       { Repeat($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               }

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

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

11.2.5 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
  | SNoexpr

type sstmt =
    SBlock of sstmt list
  | SExpr of sexpr
  | SOutput of sexpr
  | SIf of sexpr * sstmt * sstmt
  | SFor of sexpr * sexpr * sexpr * sstmt
  | SRepeat 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
  | 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 -> ""
      ) ^ ")"
```

```
54
55 let rec string_of_sstmt = function
56     SBlock(stmts) ->
57       "{\n" ^ String.concat "" (List.map string_of_sstmt stmts) ^ "}\n"
58   | SExpr(expr) -> string_of_sexpr expr ^ ";\n";
59   | SOutput(expr) -> "output " ^ string_of_sexpr expr ^ ";\n";
60   | SIf(e, s, SBlock([])) ->
61       "if (" ^ string_of_sexpr e ^ ")\n" ^ string_of_sstmt s
62   | SIf(e, s1, s2) -> "if (" ^ string_of_sexpr e ^ ")\n" ^
63       string_of_sstmt s1 ^ "otherwise\n" ^ string_of_sstmt s2
64   | SFor(e1, e2, e3, s) ->
65       "FOR each (" ^ string_of_sexpr e1 ^ " ; " ^ string_of_sexpr e2 ^ "
             ; " ^
66       string_of_sexpr e3 ^ ") " ^ string_of_sstmt s
67   | SRepeat(e, s) -> "REPEAT until (" ^ string_of_sexpr e ^ ") " ^
        string_of_sstmt s
68
69 let string_of_sfdecl fdecl =
70   "function " ^ string_of_typ fdecl.styp ^ " " ^
71   fdecl.sfname ^ "(" ^ String.concat ", " (List.map snd fdecl.sformals) ^
72   ")\n:\n" ^
73   String.concat "" (List.map string_of_vdecl fdecl.slocals) ^
74   String.concat "" (List.map string_of_sstmt fdecl.sbody) ^
75   "\n"
76
77 let string_of_sprogram (vars, funcs) =
78   String.concat "" (List.map string_of_vdecl vars) ^ "\n" ^
79   String.concat "\n" (List.map string_of_sfdecl funcs)
```

11.2.6 scanner.mll

```
1 (* Ocamllex scanner for QWEB *)
2
3 {
4   open Qwebparse
5
6   let unescape s =
7       Scanf.sscanf ("\"" ^ s ^ "\"") "%S%!" (fun x -> x)
8  }
9
10 let digit = ['0' - '9']
11 let digits = digit+
12 let ascii = ([' '-'!' '#'-'[' ']'-'~'])
13 let escape = '\\' ['\\' ''' '"' 'n' 'r' 't']
14 let string = '"' ( (ascii | escape)* as s) '"'
15
16 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 }
| "function"  { FUNCTION }
| "end"   { END }
| "if"    { IF }
| "otherwise" { OTHERWISE }
| "FOR each"  { FOR }
| "REPEAT until" { REPEAT }
| "output" { OUTPUT }
| "int"   { INT }
| "bool"  { BOOL }
| "float" { FLOAT }
| "void"  { VOID }
| "str"   { STRING }
| "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
  '\n' { token lexbuf }
```

```
| _    { comment lexbuf }
```

11.2.7 semant.ml

```
(* Semantic checking for the QWEB 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 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

  (**** 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);
                            ("createHeader", String);
                            ("createSubheader", String);
                            ("createParagraph", String);
                            ("createImage", String);
                            ("createList", String) ]
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

```

```
92    (* Return a variable from our local symbol table *)
93    let type_of_identifier s =
94      try StringMap.find s symbols
95      with Not_found -> raise (Failure ("undeclared identifier " ^ s))
96    in
97
98    (* Return a semantically-checked expression, i.e., with a type *)
99    let rec expr = function
100       Literal l -> (Int, SLiteral l)
101     | Fliteral l -> (Float, SFliteral l)
102     | BoolLit l -> (Bool, SBoolLit l)
103     | Noexpr    -> (Void, SNoexpr)
104     | StringLit s -> (String, SStringLit s)
105     | Id s      -> (type_of_identifier s, SId s)
106     | Assign(var, e) as ex ->
107        let lt = type_of_identifier var
108        and (rt, e') = expr e in
109        let err = "illegal assignment " ^ string_of_typ lt ^ " = " ^
110          string_of_typ rt ^ " in " ^ string_of_expr ex
111        in (check_assign lt rt err, SAssign(var, (rt, e')))
112     | Unop(op, e) as ex ->
113        let (t, e') = expr e in
114        let ty = match op with
115          Neg when t = Int || t = Float -> t
116        | Not when t = Bool -> Bool
117        | _ -> raise (Failure ("illegal unary operator " ^
118                               string_of_uop op ^ string_of_typ t ^
119                               " in " ^ string_of_expr ex))
120        in (ty, SUnop(op, (t, e')))
121     | Binop(e1, op, e2) as e ->
122        let (t1, e1') = expr e1
123        and (t2, e2') = expr e2 in
124        (* All binary operators require operands of the same type *)
125        let same = t1 = t2 in
126        (* Determine expression type based on operator and operand
             types *)
127        let ty = match op with
128            Add | Sub | Mult | Div when same && t1 = Int -> Int
129          | Add | Sub | Mult | Div when same && t1 = Float -> Float
130          | Add                          when same && t1 = String -> String
131        | Equal | Neq          when same               -> Bool
132        | Less | Leq | Greater | Geq
133                  when same && (t1 = Int || t1 = Float) -> Bool
134        | And | Or when same && t1 = Bool -> Bool
135        | _ -> raise (
136      Failure ("illegal binary operator " ^
137                    string_of_typ t1 ^ " " ^ string_of_op op ^ " " ^
138                    string_of_typ t2 ^ " in " ^ string_of_expr e))
139        in (ty, SBinop((t1, e1'), op, (t2, e2')))
140     | Call(fname, args) as call ->
```

```
141         let fd = find_func fname in
142         let param_length = List.length fd.formals in
143         if List.length args != param_length then
144           raise (Failure ("expecting " ^ string_of_int param_length ^
145                           " arguments in " ^ string_of_expr call))
146         else let check_call (ft, _) e =
147           let (et, e') = expr e in
148           let err = "illegal argument found " ^ string_of_typ et ^
149             " expected " ^ string_of_typ ft ^ " in " ^ string_of_expr e
150           in (check_assign ft et err, e')
151         in
152         let args' = List.map2 check_call fd.formals args
153         in (fd.typ, SCall(fname, args'))
154   in
155
156   let check_bool_expr e =
157     let (t', e') = expr e
158     and err = "expected Boolean expression in " ^ string_of_expr e
159     in if t' != Bool then raise (Failure err) else (t', e')
160   in
161
162   (* Return a semantically-checked statement i.e. containing sexprs *)
163   let rec check_stmt = function
164       Expr e -> SExpr (expr e)
165     | If(p, b1, b2) -> SIf(check_bool_expr p, check_stmt b1,
          check_stmt b2)
166     | For(e1, e2, e3, st) ->
167   SFor(expr e1, check_bool_expr e2, expr e3, check_stmt st)
168     | Repeat(p, s) -> SRepeat(check_bool_expr p, check_stmt s)
169     | Output e -> let (t, e') = expr e in
170       if t = func.typ then SOutput (t, e')
171       else raise (
172   Failure ("output gives " ^ string_of_typ t ^ " expected " ^
173     string_of_typ func.typ ^ " in " ^ string_of_expr e))
174
175     (* A block is correct if each statement is correct and nothing
176        follows any Return statement. Nested blocks are flattened. *)
177     | Block sl ->
178         let rec check_stmt_list = function
179             [Output _ as s] -> [check_stmt s]
180           | Output _ :: _ -> raise (Failure "nothing may follow a
                output")
181           | Block sl :: ss -> check_stmt_list (sl @ ss) (* Flatten
                blocks *)
182           | s :: ss       -> check_stmt s :: check_stmt_list ss
183           | []            -> []
184         in SBlock(check_stmt_list sl)
185
186   in (* body of check_function *)
187   { styp = func.typ;
```

```
188       sfname = func.fname;
189       sformals = func.formals;
190       slocals = func.locals;
191       sbody = match check_stmt (Block func.body) with
192  SBlock(sl) -> sl
193       | _ -> raise (Failure ("internal error: block didn't become a
              block?"))
194     }
195   in (globals, List.map check_function functions)
```

11.3 Test Files

11.3.1 test-*.qwb

test-add1.qwb

```
1 function int add(int x, int y){
2   output x + y;
3 }
4
5 function int main(){
6   print( add(15, 20) );
7 }
```

test-add1.out

```
1 35<br>
```

test-arith1.qwb

```
1 function int main(){
2   print(20 + 5);
3 }
```

test-arith1.out

```
1 25<br>
```

test-arith2.qwb

```
1 function int main(){
2   print(1 + 2 * 3 + 4);
3 }
```

test-arith12.out

```
11<br>
```

test-arith3.qwb

```
function int foo(int a){
  output a;
}

function int main(){
  int a;
  a = 50;
  a = a + 5;
  print(a);
}
```

test-arith3.out

```
55<br>
```

test-fib.qwb

```
function int fib(int x){
  if (x < 2){
    output 1;
  }
  output fib(x-1) + fib(x-2);
}

function int main(){
  print(fib(0));
  print(fib(1));
  print(fib(2));
  print(fib(3));
  print(fib(4));
  print(fib(5));
}
```

test-fib.out

```
1<br>
1<br>
2<br>
3<br>
5<br>
```

```
8<br>
```

test-float1.qwb

```
function int main(){
  float a;
  a = 5.123456789;
  printf(a);
}
```

test-float1.out

```
5.12346<br>
```

test-float2.qwb

```
function int main(){
  float a;
  float b;
  float c;
  a = 5.12345678;
  b = -2.12345;
  c = a + b;
  printf(c);
}
```

test-float2.out

```
3.00001<br>
```

test-float3.qwb

```
function void testfloat(float a, float b){
  printf(a + b);
  printf(a - b);
  printf(a * b);
  printf(a / b);
  printb(a == b);
  printb(a == a);
  printb(a != b);
  printb(a != a);
  printb(a > b);
  printb(a >= b);
  printb(a < b);
  printb(a <= b);
```

```
}

function int main(){
  float c;
  float d;

  c = 25.0;
  d = 5.12345;

  testfloat(c, d);
  testfloat(d, d);

}
```

test-float3.out

```
30.1234<br>
19.8766<br>
128.086<br>
4.87952<br>
0<br>
1<br>
1<br>
0<br>
1<br>
1<br>
0<br>
0<br>
10.2469<br>
0<br>
26.2497<br>
1<br>
1<br>
1<br>
0<br>
0<br>
0<br>
1<br>
0<br>
1<br>
```

test-for1.qwb

```
function int main(){
  int i;
  FOR each (i = 0 ; i < 5 ; i = i + 1){
    print(i);
  }
```

```
  print(12);
}
```

test-for1.out

```
0<br>
1<br>
2<br>
3<br>
4<br>
12<br>
```

test-for2.qwb

```
function int main(){
  int i;
  i = 0;
  FOR each ( ; i < 5; ){
    print(i);
    i = i + 1;
  }
  print(20);
}
```

test-for2.out

```
0<br>
1<br>
2<br>
3<br>
4<br>
20<br>
```

test-func1.qwb

```
function int add(int a, int b){
  output a + b;
}

function int main(){
  int a;
  a = add(7, 2);
  print(a);
}
```

test-func1.out

```
9<br>
```

test-func3.qwb

```
function void printem(int a, int b, int c, int d){
  print(a);
  print(b);
  print(c);
  print(d);
}

function int main(){
  printem(15,17,192,8);
}
```

test-func3.out

```
15<br>
17<br>
192<br>
8<br>
```

test-func4.qwb

```
function int add(int a, int b){
  int c;
  c = a + b;
  output c;
}

function int main(){
  int d;
  d = add(52, 10);
  print(d);
}
```

test-func4.out

```
62<br>
```

test-func5.qwb

```
function int foo(int a){
  output a;
}
```

```

function int main(){
}
```

test-func5.out

test-func6.qwb

```
function void foo(){
  }

function int bar(int a, bool b, int c){
 output a + c;
}

function int main(){
  print(bar(4, false, 100));
}
```

test-func6.out

```
104<br>
```

test-func7.qwb

```
int a;

function void foo(int c){
  a = c + 35;
}

function int main(){
  foo(25);
  print(a);
}
```

test-func7.out

```
60<br>
```

test-func8.qwb

```
function void foo(int a){
  print(a + 3);
```

```
}

function int main(){
  foo(40);
}
```

test-func8.out

```
43<br>
```

test-func9.qwb

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

function int main(){
  foo(40);
}
```

test-func9.out

```
43<br>
```

test-gcd.qwb

```
function int gcd(int a, int b){
  REPEAT until (a != b){
    if (a > b) a = a - b;
    otherwise b = b - a;
  }
  output a;
}

function int main(){
  print(gcd(2,14));
  print(gcd(3,15));
  print(gcd(99,121));
  output 0;
}
```

test-gcd.out

```
7<br>
```

```
4<br>
11<br>
```

test-gcd1.qwb

```
function int gcd(int a, int b){
  REPEAT until (a != b){
    if (a > b){
      a = a - b;
    }
    otherwise{
      b = b - a;
    }
  }
  output a;
}

function int main(){
  print(gcd(14,21));
  print(gcd(8,36));
  print(gcd(99,121));
}
```

test-gcd1.out

```
7<br>
4<br>
11<br>
```

test-global1.qwb

```
int a;
int b;

function void printa(){
  print(a);
}

function void printbb(){
  print(b);
}

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

```

```
function int main(){
  a = 30;
  b = 17;
  printa();
  printbb();
  incab();
  printa();
  printbb();
}
```

test-global1.out

```
30<br>
17<br>
31<br>
18<br>
```

test-global2.qwb

```
bool i;

function int main(){
  int i; # Should hide the global i
  i = 28;
  print(i + i);
}
```

test-global2.out

```
56<br>
```

test-global3.qwb

```
int i;
bool b;
int j;

function int main(){
  i = 14;
  j = 10;
  print(i + j);
}
```

test-global3.out

```
24<br>
```

test-hello.qwb

```
function int main(){
  createHeader("Hello World!");
  createSubheader("Hello World!");
  createParagraph("Hello World!");
  createList("Hello World!");
  createImage("https://source.unsplash.com/user/c_v_r");
  output 0;
}
```

test-hello.out

```
<h1>Hello World!</h1>
<h2>Hello World!</h2>
<p>Hello World!</p>
<li>Hello World!</li>
<img src='https://source.unsplash.com/user/c_v_r'>
```

test-hello2.qwb

```
function str helloworld(str a){
  output a;
}

function int main(){
  str b;
  b = helloworld("hello");
  createHeader(b);
  output 0;
}
```

test-hello2.out

```
<h1>hello</h1>
```

test-if1.qwb

```
function int main(){
  if (true){
    print(19);
  }
  print(17);
```

```
}
```

test-if1.out

```
19<br>
17<br>
```

test-if2.qwb

```
function int main(){
  if (true){
    print(19);
  }
  otherwise{
    print(8);
  }
  print(17);
}
```

test-if2.out

```
19<br>
17<br>
```

test-if3.qwb

```
function int main(){
  if (false){
    print(2);
  }
  print(17);
}
```

test-if3.out

```
17<br>
```

test-if4.qwb

```
function int main(){
  if (false){
    print(19);
  }
  otherwise{
    print(8);
```

```
  }
  print(17);
}
```

test-if4.out

```
8<br>
17<br>
```

test-if5.qwb

```
function int cond(bool b){
  int x;
  if (b)
    x = 19;
  otherwise
    x = 17;
  output x;
}

function int main(){
 print(cond(true));
 print(cond(false));
}
```

test-if5.out

```
19<br>
17<br>
```

test-if6.qwb

```
function int cond(bool b){
  int x;
  x = 10;
  if (b)
    if (x == 10)
      x = 19;
  otherwise
    x = 17;
  output x;
}

function int main(){
 print(cond(true));
 print(cond(false));
```

```
}
```

test-if6.out

```
19<br>
10<br>
```

test-local1.qwb

```
function void foo(bool i){
  int i; #Should hide the formal i

  i = 9;
  print(i + i);
}

function int main(){
  foo(true);
}
```

test-local1.out

```
18<br>
```

test-local2.qwb

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

  c = a;

  output c + 10;
}

function int main(){
 print(foo(37, false));
}
```

test-local2.out

```
47<br>
```

test-ops1.qwb

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

test-ops1.out

```
3<br>
-1<br>
2<br>
50<br>
99<br>
0<br>
1<br>
99<br>
1<br>
0<br>
99<br>
1<br>
0<br>
99<br>
1<br>
1<br>
0<br>
99<br>
```

```
0<br>
1<br>
99<br>
0<br>
1<br>
1<br>
```

test-ops2.qwb

```
function int main(){
  printb(true);
  printb(false);
  printb(true && true);
  printb(true && false);
  printb(false && true);
  printb(false && false);
  printb(true || true);
  printb(true || false);
  printb(false || true);
  printb(false || false);
  printb(!false);
  printb(!true);
  print(-10);
  print(--19);
}
```

test-ops2.out

```
1<br>
0<br>
1<br>
0<br>
0<br>
0<br>
1<br>
1<br>
1<br>
0<br>
1<br>
0<br>
-10<br>
19<br>
```

test-var1.qwb

```
function int main(){
  int a;
```

```
  a = 19;
  print(a);
}
```

test-var1.out

```
19<br>
```

test-var2.qwb

```
int a;

function void foo(int c){
  a = 15;
  a = a + 19;
}

function int main(){
  foo(73);
  print(a);
}
```

test-var2.out

```
34<br>
```

11.3.2 fail-*.qwb

fail-assign1.err

```
Fatal error: exception Failure("illegal assignment int = bool in i =
    false")
```

fail-assign1.qwb

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

  i = 42;
  i = 10;
  b = true;
  b = false;
  i = false; #Fail: assigning a bool to an integer
}
```

fail-assign2.err

```
Fatal error: exception Failure("illegal assignment int = void in i =
    voidFunc()")
```

fail-assign2.qwb

```
function void voidFunc(){
  }

function int main(){
  int i;
  i = voidFunc(); #Fail: assigning a void to an integer
}
```

fail-assign3.err

```
Fatal error: exception Failure("illegal assignment bool = str in b =
    "test string"")
```

fail-assign3.qwb

```
function int main(){
  str i;
  bool b;
  b = "test string"; #Fail: assigning a string to a bool
}
```

fail-dead1.err

```
Fatal error: exception Failure("nothing may follow a output")
```

fail-dead1.qwb

```
function int main(){
  int i;
  i = 15;
  output i;
  i = 32; #Error: code after a output
}
```

fail-expr1.err

```
Fatal error: exception Failure("illegal binary operator bool + int in d
    + a")
```

fail-expr1.qwb

```
int a;
bool b;

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

function int main(){
}
```

fail-expr2.err

```
Fatal error: exception Failure("illegal binary operator bool + int in b
    + a")
```

fail-expr2.qwb

```
int a;
bool b;

function void foo(int c, bool d){
  int d;
  bool e;
  b + a; #Error: bool + int
}

function int main(){
  output 0;
}
```

fail-float1.err

```
Fatal error: exception Failure("illegal binary operator float && int in
    -3.5 && 1")
```

fail-float1.qwb

```
function int main(){
  -3.5 && 1;
  output 0;
}
```

fail-for1.err

```
Fatal error: exception Failure("undeclared identifier j")
```

fail-for1.qwb

```
function int main(){
  int i;
  FOR each (i = 0; j < 10 ; i = i + 1){ #j undefined
    }
  output 0;
}
```

fail-for2.err

```
Fatal error: exception Failure("expected Boolean expression in i")
```

fail-for2.qwb

```
function int main(){
  int i;
  FOR each (i = 0; i ; i = i + 1){ # i is an integer, not Boolean
  }
  output 0;
}
```

fail-for3.err

```
Fatal error: exception Failure("unrecognized function foo")
```

fail-for3.qwb

```
function int main(){
  int i;
  FOR each (i = 0; i < 10; i = i + 1){
    foo(); #Error: no function foo
  }
```

```
  output 0;
}
```

fail-func1.err

```
Fatal error: exception Failure("duplicate function bar")
```

fail-func1.qwb

```
function int foo(){
}

function int bar(){
}

function int baz(){
}

function void bar(){ #Error: duplicate function bar
}

function int main(){
  output 0;
}
```

fail-func2.err

```
Fatal error: exception Failure("duplicate formal a")
```

fail-func2.qwb

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

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

function int main(){
  output 0;
}
```

fail-func3.err

```
Fatal error: exception Failure("illegal void formal b")
```

fail-func3.qwb

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

function void bar(int a, void b, int c){ #Error: illegal void formal b
}

function int main(){
  output 0;
}
```

fail-func4.err

```
Fatal error: exception Failure("function print may not be defined")
```

fail-func4.qwb

```
function int foo(){
}

function void bar(){
}

function int print(){ #Should not be able to define print
}

function void baz(){
}

function int main(){
  output 0;
}
```

fail-func5.err

```
Fatal error: exception Failure("illegal void local b")
```

fail-func5.qwb

```
function int foo(){
}

function int bar(){
  int a;
  void b; #Error: illegal void local b
```

```
  bool c;

  output 0;
}

function int main(){
  output 0;
}
```

fail-func6.err

```
Fatal error: exception Failure("expecting 2 arguments in foo(42)")
```

fail-func6.qwb

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

function int main(){
  foo(42, true);
  foo(42); #Wrong number of arguments
}
```

fail-func7.err

```
Fatal error: exception Failure("illegal argument found int expected bool
    in 42")
```

fail-func7.qwb

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

function int main(){
  foo(42, true);
  foo(42, 42); #Fail: int, not bool
}
```

fail-global1.err

```
Fatal error: exception Failure("illegal void global a")
```

fail-global1.qwb

```
int c;
```

```
bool b;
void a; #global variables should not be void

function int main(){
  output 0;
}
```

fail-global2.err

```
Fatal error: exception Failure("duplicate global b")
```

fail-global2.qwb

```
int b;
bool c;
int a;
int b; #Duplicate global variable

function int main(){
  output 0;
}
```

fail-if1.err

```
Fatal error: exception Failure("expected Boolean expression in 42")
```

fail-if1.qwb

```
function int main(){
  if (true){
    }
  if (false){
    }
  otherwise{
    }
  if (42){ #Error: non-bool predicate
    }
}
```

fail-if2.err

```
Fatal error: exception Failure("undeclared identifier foo")
```

fail-if2.qwb

```
function int main(){
  if (true){
    foo; #Error: undeclared variable
  }
}
```

fail-if3.err

```
Fatal error: exception Failure("undeclared identifier bar")
```

fail-if3.qwb

```
function int main(){
  if (true){
    42;
  }
  otherwise{
    bar; #Error: undeclared variable
  }
}
```

fail-nomain.err

```
Fatal error: exception Failure("unrecognized function main")
```

fail-nomain.qwb

fail-return1.err

```
Fatal error: exception Failure("output gives bool expected int in true")
```

fail-return1.qwb

```
function int main(){
  output true; #Should output int
}
```

fail-return2.err

```
Fatal error: exception Failure("output gives int expected void in 42")
```

fail-return2.qwb

```
function void foo(){
  if (true){ #Should output void
    output 42;
  }
  otherwise{
    output;
  }
}

function int main(){
  output 42;
}
```

fail-while1.err

```
Fatal error: exception Failure("expected Boolean expression in 42")
```

fail-while1.qwb

```
function int main(){
  int i;
  REPEAT until (true){
    i = i + 1;
  }

  REPEAT until (42){ #Should be boolean
    i = i + 1;
  }
}
```

fail-while2.err

```
Fatal error: exception Failure("unrecognized function foo")
```

fail-while2.qwb

```
function int main(){
  int i;
  REPEAT until (true){
    i = i + 1;
  }

  REPEAT until (true){
    foo(); #foo undefined
  }
}
```