COMS W4115: RJEC Final Report
Really Just Elementary Concurrency

Riya Chakraborty (rc3242), Justin Chen (jbc2186),
Yuanyuting (Elaine) Wang (yw3241), Caroline Hoang (cjh2222)

26 April 2021

Contents

1 Introduction ........................................... 10

2 Tutorial ........................................... 10
   2.1 Installation .................................. 10
   2.2 Language .................................. 11

3 Language Reference Manual .......................... 13
   3.1 Lexical Conventions ......................... 13
      3.1.1 Comments ................................ 13
      3.1.2 Reserved Keywords ...................... 13
      3.1.3 Literals ................................ 13
      3.1.4 Separators ................................ 14
   3.2 Variables .................................. 15
      3.2.1 Variable Naming ......................... 15
      3.2.2 Declaring a Variable .................... 15
      3.2.3 Multiple Variable Declaration ......... 16
   3.3 Typing Methodology .......................... 16
   3.4 Basic Data Types ........................... 17
      3.4.1 int .................................. 17
      3.4.2 bool .................................. 17
      3.4.3 char .................................. 17
   3.5 Arrays .................................... 17
      3.5.1 Declaring Arrays ....................... 18
3.11 Control Flow ........................................... 32
  3.11.1 for Loops ........................................ 32
  3.11.2 if and else Statements ......................... 33
  3.11.3 defer .............................................. 34
  3.11.4 select ............................................ 34

3.12 Built-in functions ................................. 35
  3.12.1 printi ........................................... 35
  3.12.2 printb ........................................... 35
  3.12.3 printc ........................................... 35
  3.12.4 prints ........................................... 35
  3.12.5 time ............................................ 35

4 Project Plan ........................................ 36
  4.1 Planning, Specification, Development, and Testing ....... 36
  4.2 Style Guide ......................................... 36
  4.3 Project Timeline ..................................... 37
  4.4 Team Roles and Responsibilities ....................... 39
  4.5 Software Development Environment ..................... 39
  4.6 Project Log ......................................... 40

5 Architectural Design .............................. 57
  5.1 Lexer ............................................... 57
  5.2 Parser .............................................. 58
  5.3 Semantic Checking and Transformation ................. 58
  5.4 Code Generation .................................... 58
  5.5 Imported C Code ................................... 59
  5.6 Division of Labor ................................... 59

6 Test Plan .......................................... 60
  6.1 Example Test Programs ............................. 60
    6.1.1 Single Producer-Consumer ....................... 60
    6.1.2 Implementation of C-Styled Mutex ............... 66
    6.1.3 Arrays of Structs ................................ 71
  6.2 Test Suite Design ................................ 76
    6.2.1 Test Automation and Scripts .................... 76
  6.3 Division of Labor .................................. 76

7 Lessons Learned ................................. 76
  7.1 Riya ............................................... 76

Riya Chakraborty, Justin Chen, Yuanyuting Wang, Caroline Hoang 3
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.2</td>
<td>Justin</td>
<td>77</td>
</tr>
<tr>
<td>7.3</td>
<td>Elaine</td>
<td>78</td>
</tr>
<tr>
<td>7.4</td>
<td>Caroline</td>
<td>79</td>
</tr>
<tr>
<td>8</td>
<td>Acknowledgements</td>
<td>79</td>
</tr>
<tr>
<td>9</td>
<td>Appendix</td>
<td>80</td>
</tr>
<tr>
<td>9.1</td>
<td>rjec.ml</td>
<td>80</td>
</tr>
<tr>
<td>9.2</td>
<td>scanner.ml</td>
<td>81</td>
</tr>
<tr>
<td>9.3</td>
<td>ast.ml</td>
<td>82</td>
</tr>
<tr>
<td>9.4</td>
<td>rjecparse.mly</td>
<td>86</td>
</tr>
<tr>
<td>9.5</td>
<td>sast.ml</td>
<td>92</td>
</tr>
<tr>
<td>9.6</td>
<td>semant.ml</td>
<td>95</td>
</tr>
<tr>
<td>9.7</td>
<td>codegen.ml</td>
<td>108</td>
</tr>
<tr>
<td>9.8</td>
<td>concurrency.c</td>
<td>126</td>
</tr>
<tr>
<td>9.9</td>
<td>println.c</td>
<td>130</td>
</tr>
<tr>
<td>9.10</td>
<td>Makefile</td>
<td>131</td>
</tr>
<tr>
<td>9.11</td>
<td>buildilibmill.sh</td>
<td>132</td>
</tr>
<tr>
<td>9.12</td>
<td>rjec.sh</td>
<td>132</td>
</tr>
<tr>
<td>9.13</td>
<td>testall.sh</td>
<td>134</td>
</tr>
<tr>
<td>9.14</td>
<td>tests/</td>
<td>138</td>
</tr>
<tr>
<td>9.14.2</td>
<td>fail-array1.rjec</td>
<td>138</td>
</tr>
<tr>
<td>9.14.3</td>
<td>fail-array2.err</td>
<td>139</td>
</tr>
<tr>
<td>9.14.4</td>
<td>fail-array2.rjec</td>
<td>139</td>
</tr>
<tr>
<td>9.14.5</td>
<td>fail-array3.err</td>
<td>139</td>
</tr>
<tr>
<td>9.14.6</td>
<td>fail-array3.rjec</td>
<td>139</td>
</tr>
<tr>
<td>9.14.7</td>
<td>fail-array4.err</td>
<td>140</td>
</tr>
<tr>
<td>9.14.8</td>
<td>fail-array4.rjec</td>
<td>140</td>
</tr>
<tr>
<td>9.14.9</td>
<td>fail-chan1.err</td>
<td>140</td>
</tr>
<tr>
<td>9.14.10</td>
<td>fail-chan1.rjec</td>
<td>140</td>
</tr>
<tr>
<td>9.14.11</td>
<td>fail-decl1.err</td>
<td>140</td>
</tr>
<tr>
<td>9.14.12</td>
<td>fail-decl1.rjec</td>
<td>140</td>
</tr>
<tr>
<td>9.14.13</td>
<td>fail-decl2.err</td>
<td>141</td>
</tr>
<tr>
<td>9.14.14</td>
<td>fail-decl2.rjec</td>
<td>141</td>
</tr>
<tr>
<td>9.14.15</td>
<td>fail-decl3.err</td>
<td>141</td>
</tr>
<tr>
<td>9.14.16</td>
<td>fail-decl3.rjec</td>
<td>141</td>
</tr>
<tr>
<td>Date</td>
<td>File Name</td>
<td>Page</td>
</tr>
<tr>
<td>----------</td>
<td>----------------------------</td>
<td>------</td>
</tr>
<tr>
<td>9.14.18</td>
<td>fail-decl4.rjec</td>
<td>142</td>
</tr>
<tr>
<td>9.14.19</td>
<td>fail-decl5.err</td>
<td>142</td>
</tr>
<tr>
<td>9.14.20</td>
<td>fail-decl5.rjec</td>
<td>142</td>
</tr>
<tr>
<td>9.14.22</td>
<td>fail-decl6.rjec</td>
<td>143</td>
</tr>
<tr>
<td>9.14.23</td>
<td>fail-defer1.err</td>
<td>143</td>
</tr>
<tr>
<td>9.14.24</td>
<td>fail-defer1.rjec</td>
<td>143</td>
</tr>
<tr>
<td>9.14.25</td>
<td>fail-defer2.err</td>
<td>143</td>
</tr>
<tr>
<td>9.14.26</td>
<td>fail-defer2.rjec</td>
<td>143</td>
</tr>
<tr>
<td>9.14.27</td>
<td>fail-defer3.err</td>
<td>144</td>
</tr>
<tr>
<td>9.14.28</td>
<td>fail-defer3.rjec</td>
<td>144</td>
</tr>
<tr>
<td>9.14.29</td>
<td>fail-func1.err</td>
<td>144</td>
</tr>
<tr>
<td>9.14.30</td>
<td>fail-func1.rjec</td>
<td>144</td>
</tr>
<tr>
<td>9.14.31</td>
<td>fail-func2.err</td>
<td>144</td>
</tr>
<tr>
<td>9.14.32</td>
<td>fail-func2.rjec</td>
<td>145</td>
</tr>
<tr>
<td>9.14.33</td>
<td>fail-func3.err</td>
<td>145</td>
</tr>
<tr>
<td>9.14.34</td>
<td>fail-func3.rjec</td>
<td>145</td>
</tr>
<tr>
<td>9.14.35</td>
<td>fail-struct1.err</td>
<td>145</td>
</tr>
<tr>
<td>9.14.36</td>
<td>fail-struct1.rjec</td>
<td>145</td>
</tr>
<tr>
<td>9.14.37</td>
<td>fail-struct2.err</td>
<td>146</td>
</tr>
<tr>
<td>9.14.38</td>
<td>fail-struct2.rjec</td>
<td>146</td>
</tr>
<tr>
<td>9.14.39</td>
<td>fail-struct3.err</td>
<td>146</td>
</tr>
<tr>
<td>9.14.40</td>
<td>fail-struct3.rjec</td>
<td>146</td>
</tr>
<tr>
<td>9.14.41</td>
<td>fail-yeet1.err</td>
<td>147</td>
</tr>
<tr>
<td>9.14.42</td>
<td>fail-yeet1.rjec</td>
<td>147</td>
</tr>
<tr>
<td>9.14.43</td>
<td>fail-yeet2.err</td>
<td>147</td>
</tr>
<tr>
<td>9.14.44</td>
<td>fail-yeet2.rjec</td>
<td>147</td>
</tr>
<tr>
<td>9.14.45</td>
<td>test-array1.out</td>
<td>147</td>
</tr>
<tr>
<td>9.14.46</td>
<td>test-array1.rjec</td>
<td>147</td>
</tr>
<tr>
<td>9.14.49</td>
<td>test-array11.out</td>
<td>149</td>
</tr>
<tr>
<td>9.14.50</td>
<td>test-array11.rjec</td>
<td>149</td>
</tr>
<tr>
<td>9.14.51</td>
<td>test-array2.out</td>
<td>149</td>
</tr>
<tr>
<td>9.14.52</td>
<td>test-array2.rjec</td>
<td>150</td>
</tr>
<tr>
<td>9.14.53</td>
<td>test-array3.out</td>
<td>150</td>
</tr>
<tr>
<td>9.14.54</td>
<td>test-array3.rjec</td>
<td>151</td>
</tr>
<tr>
<td>Time</td>
<td>File Name</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>-------------------</td>
<td>------</td>
</tr>
<tr>
<td>9.14.56</td>
<td>test-array4.rjec</td>
<td>151</td>
</tr>
<tr>
<td>9.14.57</td>
<td>test-array5.out</td>
<td>152</td>
</tr>
<tr>
<td>9.14.58</td>
<td>test-array5.rjec</td>
<td>152</td>
</tr>
<tr>
<td>9.14.60</td>
<td>test-array6.rjec</td>
<td>153</td>
</tr>
<tr>
<td>9.14.61</td>
<td>test-array7.out</td>
<td>153</td>
</tr>
<tr>
<td>9.14.63</td>
<td>test-array8.out</td>
<td>154</td>
</tr>
<tr>
<td>9.14.64</td>
<td>test-array8.rjec</td>
<td>154</td>
</tr>
<tr>
<td>9.14.67</td>
<td>test-assign1.out</td>
<td>155</td>
</tr>
<tr>
<td>9.14.68</td>
<td>test-assign1.rjec</td>
<td>155</td>
</tr>
<tr>
<td>9.14.69</td>
<td>test-assign2.out</td>
<td>156</td>
</tr>
<tr>
<td>9.14.70</td>
<td>test-assign2.rjec</td>
<td>156</td>
</tr>
<tr>
<td>9.14.71</td>
<td>test-assign3.out</td>
<td>156</td>
</tr>
<tr>
<td>9.14.72</td>
<td>test-assign3.rjec</td>
<td>156</td>
</tr>
<tr>
<td>9.14.73</td>
<td>test-assign4.out</td>
<td>157</td>
</tr>
<tr>
<td>9.14.74</td>
<td>test-assign4.rjec</td>
<td>157</td>
</tr>
<tr>
<td>9.14.75</td>
<td>test-assign5.out</td>
<td>157</td>
</tr>
<tr>
<td>9.14.76</td>
<td>test-assign5.rjec</td>
<td>157</td>
</tr>
<tr>
<td>9.14.77</td>
<td>test-chan1.out</td>
<td>158</td>
</tr>
<tr>
<td>9.14.78</td>
<td>test-chan1.rjec</td>
<td>158</td>
</tr>
<tr>
<td>9.14.79</td>
<td>test-chan2.out</td>
<td>158</td>
</tr>
<tr>
<td>9.14.80</td>
<td>test-chan2.rjec</td>
<td>158</td>
</tr>
<tr>
<td>9.14.81</td>
<td>test-chan3.out</td>
<td>159</td>
</tr>
<tr>
<td>9.14.82</td>
<td>test-chan3.rjec</td>
<td>159</td>
</tr>
<tr>
<td>9.14.83</td>
<td>test-chan4.out</td>
<td>159</td>
</tr>
<tr>
<td>9.14.84</td>
<td>test-chan4.rjec</td>
<td>160</td>
</tr>
<tr>
<td>9.14.85</td>
<td>test-chan5.out</td>
<td>161</td>
</tr>
<tr>
<td>9.14.86</td>
<td>test-chan5.rjec</td>
<td>161</td>
</tr>
<tr>
<td>9.14.87</td>
<td>test-decl-assign1.out</td>
<td>161</td>
</tr>
<tr>
<td>9.14.88</td>
<td>test-decl-assign1.rjec</td>
<td>162</td>
</tr>
<tr>
<td>9.14.89</td>
<td>test-decl-assign2.out</td>
<td>162</td>
</tr>
<tr>
<td>9.14.90</td>
<td>test-decl-assign2.rjec</td>
<td>162</td>
</tr>
<tr>
<td>9.14.91</td>
<td>test-decl-assign3.out</td>
<td>163</td>
</tr>
<tr>
<td>Date</td>
<td>File Name</td>
<td>Page</td>
</tr>
<tr>
<td>----------</td>
<td>--------------------</td>
<td>------</td>
</tr>
<tr>
<td>9.14.95</td>
<td>test-decl1.out</td>
<td>164</td>
</tr>
<tr>
<td>9.14.96</td>
<td>test-decl1.rjec</td>
<td>164</td>
</tr>
<tr>
<td>9.14.97</td>
<td>test-decl2.out</td>
<td>164</td>
</tr>
<tr>
<td>9.14.98</td>
<td>test-decl2.rjec</td>
<td>165</td>
</tr>
<tr>
<td>9.14.99</td>
<td>test-decl3.out</td>
<td>165</td>
</tr>
<tr>
<td>9.14.100</td>
<td>test-decl3.rjec</td>
<td>165</td>
</tr>
<tr>
<td>9.14.101</td>
<td>test-defer1.out</td>
<td>166</td>
</tr>
<tr>
<td>9.14.102</td>
<td>test-defer1.rjec</td>
<td>166</td>
</tr>
<tr>
<td>9.14.103</td>
<td>test-defer2.out</td>
<td>166</td>
</tr>
<tr>
<td>9.14.104</td>
<td>test-defer2.rjec</td>
<td>166</td>
</tr>
<tr>
<td>9.14.109</td>
<td>test-for1.out</td>
<td>168</td>
</tr>
<tr>
<td>9.14.110</td>
<td>test-for1.rjec</td>
<td>168</td>
</tr>
<tr>
<td>9.14.111</td>
<td>test-for2.out</td>
<td>169</td>
</tr>
<tr>
<td>9.14.112</td>
<td>test-for2.rjec</td>
<td>169</td>
</tr>
<tr>
<td>9.14.113</td>
<td>test-for3.out</td>
<td>169</td>
</tr>
<tr>
<td>9.14.114</td>
<td>test-for3.rjec</td>
<td>170</td>
</tr>
<tr>
<td>9.14.115</td>
<td>test-for4.out</td>
<td>170</td>
</tr>
<tr>
<td>9.14.117</td>
<td>test-for5.out</td>
<td>171</td>
</tr>
<tr>
<td>9.14.118</td>
<td>test-for5.rjec</td>
<td>171</td>
</tr>
<tr>
<td>9.14.119</td>
<td>test-func1.out</td>
<td>171</td>
</tr>
<tr>
<td>9.14.120</td>
<td>test-func1.rjec</td>
<td>172</td>
</tr>
<tr>
<td>9.14.121</td>
<td>test-func2.out</td>
<td>172</td>
</tr>
<tr>
<td>9.14.122</td>
<td>test-func2.rjec</td>
<td>172</td>
</tr>
<tr>
<td>9.14.127</td>
<td>test-func5.out</td>
<td>174</td>
</tr>
<tr>
<td>9.14.128</td>
<td>test-func5.rjec</td>
<td>174</td>
</tr>
<tr>
<td>9.14.129</td>
<td>test-hello.out</td>
<td>175</td>
</tr>
<tr>
<td>9.14.130</td>
<td>test-hello.rjec</td>
<td>175</td>
</tr>
<tr>
<td>9.14.131</td>
<td>test-if1.out</td>
<td>176</td>
</tr>
</tbody>
</table>

Riya Chakraborty, Justin Chen, Yuanyuting Wang, Caroline Hoang
9.14.170 test-string1.rjec ............................................. 192
9.14.171 test-string2.out ............................................. 192
9.14.172 test-string2.rjec ............................................. 193
9.14.175 test-string4.out ............................................. 193
9.14.176 test-string4.rjec ............................................. 194
9.14.177 test-struct1.out ............................................. 194
9.14.178 test-struct1.rjec ............................................. 194
9.14.179 test-struct2.out ............................................. 195
9.14.183 test-struct4.out ............................................. 196
9.14.184 test-struct4.rjec ............................................. 196
9.14.185 test-struct5.out ............................................. 196
9.14.186 test-struct5.rjec ............................................. 197
9.14.188 test-struct6.rjec ............................................. 198
9.14.189 test-time1.out ............................................. 199
9.14.190 test-time1.rjec ............................................. 199
9.14.191 test-yeet1.out ............................................. 199
9.14.192 test-yeet1.rjec ............................................. 200
9.14.193 test-yeet2.out ............................................. 200
9.14.194 test-yeet2.rjec ............................................. 200
9.14.195 test-yeet3.out ............................................. 201
9.14.196 test-yeet3.rjec ............................................. 201
9.15 demo/ .............................................................. 201
9.15.1 mapreduce.rjec ................................................. 201
9.15.2 mutex.rjec ..................................................... 207
9.15.3 producer_consumer.rjec ........................................ 208
9.15.4 rand_ints.rjec .................................................. 209
9.16 libmill/ ............................................................ 209
1 Introduction

RJEC (Really Just Elementary Concurrency) is an imperative language with a primary focus on concurrent programming. It is based on Go, and its syntax and features are largely a strict subset of Go’s. Like Go, RJEC is imperative, and concurrency abstractions are incorporated as language primitives. These features enable somewhat higher-level, CSP-style concurrency, which are useful in distributed systems applications.

RJEC has clear, concise, consistent, and clean syntax, allowing for convenient imperative programming. It adds syntactic sugar to help programmers write succinct and readable code. For example, limited compile-time type deduction is provided to make variable declaration clean and easy, and yeetroutines abstract away the passing of arguments and communication channels so that the programmer can focus on writing nice concurrent code.

RJEC provides basic language constructs such as arrays and structs, alongside the high-level concurrency constructs of yeetroutines and channels. These features, combined, enable the programmer to write powerful and expressive code, in both sequential and concurrent paradigms.

Note that the goal of RJEC is concurrency, not parallelism. Strong concurrent programming semantics enable good parallel computing, but a concurrent program does not need to be executing on multiple cores simultaneously for it to have good and correct structure. Our language is designed to enable correct high-level concurrent programming, which will remain correct even if the environment is optimized to make it parallel.

2 Tutorial

2.1 Installation

We currently support installation by Docker. Download the source code and run the following commands:

```bash
docker run --rm -it -v `pwd`:/home/microc -w=/home/microc
columbiasedwards/plt
make all
```
To run the test suite, run make or ./testall.sh when the compiler is already compiled.

To compile your own programs, run ./rjec.sh <filename>.rjec when the compiler is already compiled.

2.2 Language

In the words of Kernighan & Ritchie, the first program is the same for all languages:

```go
func main() {
    prints("hello, world");
}
```

You may declare and initialize variables as follows.

```go
var i int = 5;
```

You could also use the short-form same-line declaration and initialization syntax as follows. The type of the variable to declare is deduced from the right-hand-side expression in compile-time:

```go
i := 5;
```

RJEC supports three basic data types: integers, booleans, and characters.

Composite data types such as arrays and structs are also supported. Arrays are declared with this syntax:

```go
var arr [i]char;
arr2 := [3]bool{true, true, false};
```

You may also create and store strings as null-terminated char arrays:

```go
str := "hello";
```

Structs are defined globally:

```go
struct my_struct {
    i int;
    b bool;
    c char;
}
```
And can be declared with the following syntax. When creating struct literals and conducting value assignments, member fields are optional; if a member field value is not specified, it will be initialized with the zero value of its type.

```go
s := my_struct {
    i: 1,
    c: 'a'
}
```

You can define a function as follows, where the formal arguments are in the parentheses and the return type is at the end:

```go
func foo (i int, j char) int {
    printc(j);
    return i + 1;
}
```

To start a concurrent thread executing a function call, we use yeetroutines:

```go
yeet foo(5);
```

To communicate between yeetroutines, we use channels, another composite data type that we support:

```go
func bar(ch chan int) {
    ch <- 5;
}

func main() {
    ch := make(chan int);
    yeet bar(ch);
    printi(<- ch);
}
```

This concludes our brief tour of this language. For more details, refer to our manual below.
3 Language Reference Manual

3.1 Lexical Conventions

3.1.1 Comments

RJEC has support for multi-line comments (though they may take up a single line). Any tokens following /* are considered part of a comment and are essentially discarded (they are not included in further lexical analysis or parsing). A comment is over when the */ token is encountered.

3.1.2 Reserved Keywords

The reserved keywords are as follows, grouped together by their particular uses:

The keywords representing data types are: int, bool (for which we have true, and false), char, chan, struct, array. Note that the first three indicate the basic data types, and the following three indicate the composite data types.

The keywords useful for dictating control flow are: if, else, for, defer, select, case, return, yeet.

The keywords necessary for long-form declarations are: var.

The keywords related to channels are: make, close (discussed in greater detail below).

3.1.3 Literals

Literals can be used to represent either the basic data types (int, char, bool), strings, or composite data types such as arrays and structs.

String literals are a sequence of characters wrapped in double quotation marks, i.e. "rjec is the best". String literals do not need to be bound to a variable.

The lexer is able to identify string literals by going into a different mode of lexing when encountering a double quote (") and reads contents to a buffer (which constitute the contents of a string literal) until another double quote (") is encountered.
A char literal is a single character wrapped in single quotation marks, i.e. `var mychar char = 'r'`. Character literals do not necessarily have to be bound to a variable either. The user is allowed to use `'0'` to represent the zero value of the `char` type (also used as null terminators for strings).

An integer literal can be any sequence of integers, each of which are between 0 and 9. The corresponding regex that is matched to by the lexer is `['0'-'9']*`.

A boolean literal can either be the `true` or `false` keyword, i.e. `var mybool bool = false`.

Array and struct literals are described in their respective sections below.

### 3.1.4 Separators

RJEC uses parentheses `()` to override any default expression precedence. Similarly, square brackets `[]`, curly braces `{}` are processed as separators. White spaces are indeed separators, but are not recognized as tokens. The other potential separators include commas `,`, semicolons `;`, dots `.`, and colons `:`.

The following provide examples of usage of separators:

```plaintext
a, b := 1, 2;
/* ',' as a separator in multiple variable declaration */

var newArr [2]char = ['a', 'b'];
/* [] and {} for array declaration and initialization */

newArr[1] = 'c';
/* [] in array access and ';' indicate the end of a statement */

x_coord = point.x;
/* '.' to access field of a struct */

case data <- x:
/* ':' in case statement */
```
3.2 Variables

3.2.1 Variable Naming

Variable, function, and type identifiers must begin with a letter, and must contain only letters, numbers, and underscores. As an example `var_name_1` is a valid identifier, as is `VarName2`. `2strong` is not a valid variable, and neither is `_var_name_3`.

3.2.2 Declaring a Variable

Variables may be declared by using their names followed by their type, except in the case of arrays, for which there is a special syntax for noting their type and size, even though the length of an array is not part of its type:

\[
\text{vdecl:}
\]

\[
\text{var id-list vdecl-typ}
\]

\[
id-list:
\]

\[
\text{identifier, id-list}
\]

\[
vdecl-typ:
\]

\[
\text{int, bool, char, chan basic-typ, [expr] typ, struct identifier}
\]

Variables may be declared both globally and in functions. Variables declared without being explicitly initialized are initialized to their zero value. Variables may be declared and initialized in the same line within functions. The operator `identifier := expr` may act as shorthand for variable initialization by assigning the type of the expression to the new variable (see section 3.10.19).

As examples, the following are both valid variable initialization:

```
var i int = 5;
j := 5;
```
3.2.3 Multiple Variable Declaration

As can be seen in the grammar above, multiple variables of the same type may be declared and initialized in one line by being separated by commas. Furthermore, using the := operator (as seen in the grammar in 3.10.19), multiple variables of different types may be declared in one line by being assigned. As examples:

```plaintext
var i, j int = 2, 3;
k, s := 5, "hi";
```

3.3 Typing Methodology

RJEC is statically typed and strongly typed. The language supports three basic data types: `int`, `bool`, and `char`. In addition, it also supports three composite types: `array`, `struct`, and `chan`. Note that to this end, string literals will be represented as char arrays rather than as its own type. The types are represented in RJEC’s grammar as:

```plaintext
typ:
   int
   bool
   char
   chan basic-typ
   [] non-arr-typ
   struct identifier
```

RJEC enforces a strong typing system, which means the language does not conduct implicit type conversions or casting. Operands on both sides of binary operators have to be of the same type, and failure to abide by the typing system will lead to compiler errors. RJEC does not support type casting or implicit type conversions, although RJEC offers limited compile-time type deduction when using the := operator (where type of the right-hand-side expression is used to declare the variable on the left-hand-side).

Note that unlike some other language conventions, RJEC treats `int` and `bool` as distinct types, which means statements such as `1 == true` will lead to compiler errors as well.
3.4 Basic Data Types

The basic data types in RJEC consist of the following:

\[
\text{basic-typ:}
\]

- \text{int}
- \text{bool}
- \text{char}

3.4.1 int

The \text{int} type represents signed, 4-byte integer numbers. The integer values are implicitly non-negative, and negative integers are represented by prefixing the unary minus operator \(-\) to the numerical value. The zero value for \text{int} is 0.

3.4.2 bool

The \text{bool} type represents 1-byte boolean values. A \text{bool} type variable can have the value of either \text{true} or \text{false}. The zero value for \text{bool} is \text{false}.

3.4.3 char

The \text{char} type represents 1-byte ASCII characters. A char literal is represented by enclosing ASCII symbols in single quotes, i.e.

\[
'\langle\text{ASCII\_symbol}\rangle'
\]

Note that in RJEC, string literals are represented as a char array, and are represented by enclosing 0 or more ASCII symbols in double quotes, i.e.

\[
'\langle\text{ASCII\_symbol}\rangle'*
\]

The zero value for \text{char} is '\0\', which also represents null terminator in a string literal.

3.5 Arrays

In RJEC, \text{array} is a composite data type that allows for the storage of an ordered set of elements in a consecutive memory. Note that the length of an array is fixed upon declaration, though they may be declared using variables, and that all the elements in an array must be of the same data type as specified in the declaration. The data
type contained in an array can be any other data type supported by RJEC, including composite data types such as `struct` and `chan`, except for `array` (nested arrays are not supported).

### 3.5.1 Declaring Arrays

One declares an array by specifying the data type of the elements, the length, and the identifier of the array. The declaration takes the following form:

```go
var id-list [expr] non-arr-typ
```

Where `non-arr-typ` is a type which is not an array type:

```go
non-arr-typ:
  int
  bool
  char
  chan basic-typ
  struct identifier
```

For example, to declare an integer array of length 10 and identifier “myArr”:

```go
var myArr [10]int;
```

Once declared, an array object has the type of `[]<data_type>`. Note that the `expr` that specifies the length of the array must evaluate to an integer value.

### 3.5.2 Defining Arrays

In the grammar, an array literal is defined as:

```go
[expr] non-arr-typ { args-list }
```

Where `expr` should evaluate to an integer value that represents the length of the array. With this, one can initialize the contents of an array upon declaration through enumeration. That is:

```go
var identifier []<data-type> =
[]<data-type>{elem1, elem2, ...};
```
If the variable to be declared is to be initialized immediately in the same statement, one can also choose to use a short-form assignment statement instead:

\[
\text{identifier} := []<\text{data-type} \{ \text{elem1, elem2, ...} \};
\]

For example, here are the two ways to declare and initialize an integer array of length 3 and populate it with values:

\[
\text{var myArr [3]int = []int\{1, 2, 3\};}
/* or equivalently */
\text{myArr := []int\{1, 2, 3\};}
\]

Note that if an array is not immediately initialized upon declaration, its fields will be filled by the zero values of the data type specified for its elements.

One can also define an array post-declaration, either by assigning a new value to the entire array, or by accessing and modifying individual array elements with the following format:

\[
\text{identifier[index] = <new_value>};
\]

### 3.5.3 Accessing Array Elements

One can access specific elements in an array by their index, with the following expression:

\[
\text{identifier[expr]}
\]

Note that in RJEC, the array index starts with 0, so for any array, except for the 0-length arrays, any index value between 0 and \( \text{length} - 1 \) (inclusive) is within range.

### 3.6 Structs

A **struct** in RJEC is a user-defined data type that consists of fields taken up by elements of the basic data types (**int**, **char**, **bool**).

#### 3.6.1 Defining Structs

In RJEC program files, all structs are defined globally, with all fields public. A struct can be defined with the following grammar:
For example, to define a coordinate type that represents a point in a cartesian coordinate system:

```c
struct coordinate {
    x int;
    y int;
}
```

Note that the RJEC compiler enforces that the identifiers for the struct need to be globally unique, and identifiers for the struct field need to be unique within the struct. Also, note again that currently RJEC does not support composite data types for struct fields.

### 3.6.2 Declaring and Initializing Variables of Type Struct

Once a struct type has been globally defined, one can declare variables of the struct type globally or locally, similar to how one declares objects of the primitive types. A struct literal is defined in the grammar as:

```c
...:

struct identifier { element-list_opt }
...
```

A struct object can be declared and initialized with the following syntax:
var identifier struct struct-id = struct struct-id {
    field-id1 : <value1>,
    field-id2 : <value2>
};
/* or equivalently */
identifier := struct struct-id {
    field-id1 : <value1>,
    field-id2 : <value2>
};

For example, to declare and initialize a coordinate struct object:

    coord := struct coordinate {
        x : 1,
        y : 2
    };

Note that similar to arrays, if a struct is declared without being immediately ini-
tialized, then all of its fields will be taken up by the zero values specified for their
respective data types. Later, one can assign values to the individual struct fields
with the struct access operator:

    struct-id.field-id = <value>;

Furthermore, a struct literal may only mention some but not all of the member
fields, in which case the remaining member fields are assigned to their zero values,
as described above.

3.6.3 Accessing Fields of a Struct

All fields in a struct are public, and can be accessed through the struct access operator
. with the following expression:

    expr.field-id

Where the expr evaluates to an identifier for a struct or a struct array member.

3.7 Channels

A chan type, or channel, provides a way for concurrently executing yeetroutines to
communicate by sending and receiving values of a specified data type.
3.7.1 Declaring and Initializing Channels

A `chan` can be declared with the following grammar:

```
var id-list chan basic-typ
```

The declared channel will then have a type specification of `chan <basic-typ>`.

In addition, a `chan` variable can be initialized by the function-like keyword `make`, which allocates resources for either an unbuffered channel, or a buffered channel with a user-defined size. The expression for initializing an unbuffered channel:

```
make(chan basic-typ)
```

For initializing a buffered channel:

```
make(chan basic-typ, expr)
```

Here, the expression used to specify the buffer size should evaluate to a non-negative integer value.

When a channel is unbuffered, or when it has a buffer size of 0, the communication succeeds only when both the sender and the receiver are ready for the transaction. When a channel has a positive buffer size, then the elements can "queue up" in the channel, and communication will succeed as long as the buffer is not full (for sending) and not empty (for receiving).

3.7.2 Sending and Receiving Items

Once a channel has been created and is shared between two subroutines, the two functions can act as sender and receiver of elements through the channel, by the use of the arrow operator `<-`.

Sending an element through a channel uses the expression:

```
  id-or-subscript <- expr
```

Receiving an element uses the expression:

```
  <- id-or-subscript
```

Where `id-or-subscript` is either an ID or an array member:
The receive operation returns the element received. The program can also assign the received element directly to a variable by:

\[
\text{identifier} := \langle- \text{chan}_\text{id}\rangle;
\]

Note that the send and receive operations might block under the circumstances where either a buffered channel is full (for sending) or empty (for receiving), or the sender and receiver for an unbuffered channel are not both ready for the transaction.

### 3.7.3 Passing Channels as Function Parameters

A \texttt{chan} object can be shared between multiple concurrently executing subroutines / functions for them to communicate with each other. To do this, the channel could be locally created in one of the routines, and then passed into the other routines as a parameter. An example of this can be seen in the example code in section 6.1.1.

### 3.7.4 Closing Channels

The function-like keyword \texttt{close} can be used to close an existing channel. The associated expression is

\[
\text{close}(\text{id-or-subscript})
\]

Note that a channel only needs to be closed once for a thorough cleanup. An attempt to close a channel more than once would result in an error.

After a channel is closed, an attempt to receive from the channel will result in the zero value for the specified data type being returned. However, attempt to send an item through a closed channel would result in an error. Therefore, although not enforced in the grammar, it is recommended practice to always close the channel from the sender’s side.

### 3.8 Functions

A RJEC program consists of a sequence of global variable, function, and struct declarations, which can be defined in any order in relation to one another.
### 3.8.1 Defining Functions

Function definitions have the form

\[
\text{function-declaration:} \\
\quad \text{func identifier ( parameter-list opt ) return-types opt } \{ \text{statement-list opt } \}
\]

\[
\text{parameter-list:} \\
\quad \text{identifier type} \\
\quad \text{identifier type, parameter-list}
\]

\[
\text{return-types:} \\
\quad \text{type}
\]

\[
\text{type-list:} \\
\quad \text{type} \\
\quad \text{type-list, type}
\]

\[
\text{statement-list:} \\
\quad \text{statement-list statement}
\]

A function definition notes its identifying name, its parameters, its return types, and a list of statements (which are executed in order unless otherwise specified by control flow). The grammar has the framework for supporting multiple return types, but this has not yet been implemented.

The following is a valid complete function definition:

\[
\text{func add(x int, y int) int } \{ \\
\quad \text{return x + y;}
\}
\]
3.8.2 Calling Functions

Functions may be called via the following expression:

\[ \text{identifier} \left( \text{args-list}_{\text{opt}} \right) \]

We list the function parameters as expressions separated by commas:

\[ \text{args-list:} \]

\[ \text{expr} \]
\[ \text{expr, args-list} \]

Functions in RJEC are pass-by-value. Calling a function passes the parameters to the function by value, and then executes the statements within that function. However, note that arrays and channels store their information by reference.

3.8.3 The main Function

The main function is the entry point for code execution. It runs immediately after all global definitions are taken. It has no parameters and no return types.

\[
\text{func main()} \{ \\
    /* \text{execute code} */ \\
\}
\]

3.8.4 The return Statement

The return statement is defined as follows:

\[ \text{return} \ \text{args-list}_{\text{opt}}; \]

The return statement ends execution of the current function and returns to the calling function (or in the case of the main function, ends execution of the program). It must be followed by a list of expressions representing the return arguments. These expressions must be of the respective return types specified in the function declaration.
3.8.5 yeeting Functions

The yeet statement is defined as follows:

```
  yeet expr;
```

Where the expression must be a function call. The function may take any number of formals, though currently the function return type of a yeeted function is limited to int and none.

yeeting a function runs that function concurrently in a separate thread. That thread terminates when the function returns.

Here is an example of a valid yeet:

```
  yeet foo(2, 4);
```

3.9 Statements & Expressions

As noted in the previous section, a function body consists of a list of statements. A statement is defined as follows:

```
statement:

  expr;
  vdecl;
  assign-stmt;
  return args-listopt;
  { statement-list }
  if expr { statement-list } else-statementopt
  for assign-stmtopt; expr; assign-stmtopt { statement-list }
  for expr { statement-list }
  for { statement-list }
  select { case-list }
  defer expr;
  yeet expr;
```

Most statements are expression statements; i.e. consisting of an expression (expr;). Other statements are described in their relevant sections in this document.

Expressions are defined as follows:
expr:

int-literal
string-literal
char-literal
bool-literal
[expr] non-arr-typ { args-list }
struct identifier { element-list } identifier
expr binop expr
- expr
! expr
identifier ( args-listopt )
( expr )
expr . identifier
identifier [ expr ]
id-or-subscript <- expr
<- id-or-subscript
make(chan basic-typ )
make(chan basic-typ , expr )
close( identifier )

Where binop is defined as the arithmetic, boolean, and logical binary operators +, -, *, /, %, ==, <, <=, &&, and ||.

These expressions are all explained in their various relevant sections.

3.10 Operators

3.10.1 Associativity and Order of Precedence

The associativity and order of precedence for operators in RJEC can be seen in table 3.10.1, with the various operators listed in order of precedence from top to bottom.

3.10.2 Arithmetic Operators

RJEC provides operators that perform basic arithmetic operations, as seen in the following subsections. Note that the binary operators only support integer arithmetic operations, and return an int value.
<table>
<thead>
<tr>
<th>Operators</th>
<th>Symbols</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Struct access operator</td>
<td>.</td>
<td>left-to-right</td>
</tr>
<tr>
<td>Array access and instantiation operator</td>
<td>[]</td>
<td>left-to-right</td>
</tr>
<tr>
<td>Logical NOT operator and unary negation operator</td>
<td>!, -</td>
<td>right-to-left</td>
</tr>
<tr>
<td>Arrow operator</td>
<td>&lt;-</td>
<td>right-to-left</td>
</tr>
<tr>
<td>Multiplication, Division and Modulo operators</td>
<td>*, /, %</td>
<td>left-to-right</td>
</tr>
<tr>
<td>Addition and Subtraction operators</td>
<td>+, -</td>
<td>left-to-right</td>
</tr>
<tr>
<td>Less than or equal to operators</td>
<td>&lt;=, &lt;</td>
<td>left-to-right</td>
</tr>
<tr>
<td>Equality operator</td>
<td>==</td>
<td>left-to-right</td>
</tr>
<tr>
<td>Logical AND operator</td>
<td>&amp;&amp;</td>
<td>left-to-right</td>
</tr>
<tr>
<td>Logical OR operator</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Assignment operators</td>
<td>=, :=</td>
<td>right-to-left</td>
</tr>
</tbody>
</table>

### 3.10.3 Addition Operator

The addition binary operator + returns the sum of the operands. Its associated expression is:

\[ expr + expr \]

### 3.10.4 Subtraction Operator

The subtraction operator - subtracts the right operand from the left operand and returns the result. Its associated expression is:

\[ expr - expr \]

### 3.10.5 Multiplication Operator

The multiplication operator * returns the product of the operands. Its associated expression is:

\[ expr * expr \]

### 3.10.6 Division Operator

The division operator / divides the left operand with the right operand and returns the result. Its associated expression is:

\[ expr / expr \]
3.10.7 Modulo Operator
The modulo operator is used to obtain the remainder produced by dividing the left operand with the right operand. Its associated expression is:

\[ expr \% expr \]

3.10.8 Unary Negation Operator
The unary negation operator flips the sign of the original integer value of its operand. Its associated expression is:

\[-expr\]

3.10.9 Boolean Operators
Boolean operators are described below. They all return a bool value as the result.

3.10.10 Equality Operator
The equality operator == conducts a comparison of its two operands, and return true if and only if the two operands are identical by value, or false otherwise. This operator only supports types of int, bool, and char. Its associated expression is:

\[ expr == expr \]

3.10.11 Less Than Operator
The less than operator < returns true if the left operand evaluates to a smaller value than its right operand, or false otherwise. This operator only supports types of int and char. Its associated expression is:

\[ expr < expr \]

3.10.12 Less Than or Equal To Operator
The less than or equal to operator <= returns true if the left operand evaluates to a smaller value than, or is equal to its right operand, or false otherwise. This operator only supports types of int and char. Its associated expression is:

\[ expr <= expr \]
3.10.13 Logical Operators

The logical operators are used to test the truth value of various combinations of one or two expressions. Note that these logical operators only support \texttt{bool} operands, and return a \texttt{bool} value.

3.10.14 AND Operator

The logical AND operator, or logical conjunction operator, \&\& returns \texttt{true} if and only if both operands evaluate to \texttt{true}, or \texttt{false} otherwise. If the left operand already evaluates to \texttt{false}, then the expression returns \texttt{false} without evaluating the right operand. Its associated expression is:

\[
expr \&\& expr
\]

3.10.15 OR Operator

The logical OR operator, or logical disjunction operator, \|\| returns \texttt{true} if and only if at least one of the two operands evaluates to \texttt{true}, or \texttt{false} otherwise. If the left operand already evaluates to \texttt{true}, then the expression returns \texttt{true} without evaluating the right operand. Its associated expression is:

\[
expr \|\| expr
\]

3.10.16 NOT Operator

The logical NOT operator, or logical negation operator, \! flips the truth value of its operand. Its associated expression is:

\[
!expr
\]

3.10.17 Arrow Operator

The arrow operator <- is used for sending and receiving elements through objects of type \texttt{chan}. The expressions associated with this operator are:

\[
\text{\textit{id-or-subscript} <- expr} \\
<- \text{\textit{id-or-subscript}}
\]

For more details on its usage, see section 3.7.2.
3.10.18 Access Operators

The access operators are used to access specific items contained in the composite data types `struct` and `array`. You can use the struct access operator . to access the fields of a struct with the following expression:

\[ \text{expr}.\text{field-id} \]

Where the `expr` can be either a struct identifier or an expression that evaluates to a struct type (such as access to a struct array element).

You can use the array access operator [] to access elements at specific indices of an array with the following expression:

\[ \text{array-identifier}[\text{expr}] \]

See sections 3.5.3 and 3.6.3 for more usage of access operators in composite data types.

3.10.19 Assignment Operators

Assign operators = and := are used to store values in variables. The various assignment statements are defined as such in the RJEC grammar:

\[
\text{assign-stmt:}
\]

\[
\text{id-list} = \text{args-list}
\]
\[
\text{vdecl} = \text{args-list}
\]
\[
\text{id-list} := \text{args-list}
\]

If the left-hand-side variable(s) have already been declared beforehand, then = is used to store the right-hand-side expression value(s) in the variables, designated by the identifier(s) on the left hand side. Both the variables and expressions are separated by commas. For example, to assign 2 integer values to two previously-declared `int` type variables:

\[ a, b = 1, 2; \]

Note that the behavior of assigning multiple variables to other variables referenced in the same statement currently evaluates these assignments in order. This is a known issue; this syntax should thus currently not be used to swap variables, or similar.
The assignment operators can also be used in the case where variable(s) are declared and initialized in the same statement, in which case one could use either the long-form or the short-form assignment statements.

A long-form assignment statement first declares the variable(s) to be initialized, specifying their data type using the `var` keyword on the left hand side, and then uses the `=` operator for the assignment. For example, to declare and initialize one or two `int` variables:

```go
var a int = 0;
var b, c int = 1, 2;
```

Note that in the long-form statement, if there are multiple variables to be declared, they are required to be of the same type.

A short-form assignment statement simply refers to variable(s) to be declared by their identifier(s) on the left hand side, and directly initializes each variable with its corresponding expression on the right hand side, using the `:=` operator. For example,

```go
a := 0;
b, c := 1, 2;
```

Note that in this case, the variables to be declared together can have values of different data types assigned to each of them. For example,

```go
d, e := 3, "4";
```

### 3.11 Control Flow

#### 3.11.1 for Loops

`for` loops are used in order to repeat the execution of a sequence of statements and expressions for a specified number of repetitions, for until the termination condition is reached, or infinitely otherwise. The conventional `for` loop definition involves specifying the `initialize` statement (run prior to the loop), `test` boolean expression (termination condition checked after each iteration of the loop), and `step` statement (run after each iteration of the loop).

```
for assign-stmt_opt; expr; assign-stmt_opt { statement-list }
```
A for loop may also have the behavior of a conventional while loop if only the termination condition, or the test expression, is specified, in which case the statements will keep executing until the condition evaluates to false.

for expr { statement-list }

If no conditions are specified, the for loop will continue executing infinitely, unless a return statement is executed at some point.

for { statement-list }

### 3.11.2 if and else Statements

if statements are used to place a condition on the execution of sequence of statements. The statements wrapped in an if statement will only execute if the condition expression evaluates to true.

The inclusion of else statements as well as else if statements are optional but can be provided in order to further specify a group of statements for execution under additional circumstances.

In an if statement block, there exists one if condition, at most one else, and 0 or more else if conditions in-between. The conditions are evaluated in order, and statements contained in the first true condition will be executed. If an else exists, its statements are executed only after all preceding conditions have evaluated to false.

statement:

...  
if expr { statement-list } else-statement  
...  
else-statement:

no-else  
else if expr { statement-list } else-statement  
else { statement-list }
Where *no-else* is defined as an empty token and has lower precedence than the *else* token.

### 3.11.3 defer

The defer statement is used to defer the evaluation of certain expressions until right before the current function returns.

A `defer` statement is defined as follows:

```language-none
defer expr;
```

Note that `defer` is currently somewhat unstable and does not work with some conditional blocks. This is a known issue. However, it still works well for the common use case of easily freeing a resource (e.g., a user-defined lock) in basic functions.

### 3.11.4 select

The `select` statement is used in order to block and wait on multiple channel-related expressions, each represented as a *case*. It then executes the statements contained in the first case that successfully executes, i.e., the first channel through which it is able to send or receive an element.

The `select` statement can be defined by the following grammar:

```language-none
statement:

... select{ case-list } ...

case-list:

case case-stmt : statement-list
case case-stmt : statement-list case-list

case-stmt:

  id-or-subscript <- expr
  <- id-or-subscript
  assign-stmt
```
Note that for each case, the expr can take the form of either sending or receiving an item through a channel object. It can also be an assignment statement where an item is received from a channel and then directly assigned / used to initialize a variable. An example of using select statements:

```java
select {
    case a := <- my_chan:
        ...
    case my_chan_2 <- b:
        ...
}
```

### 3.12 Built-in functions

We support some built-in functions for printing, debugging, measurement, and utility purposes.

#### 3.12.1 printi

The function printi takes a single int and prints it in its own line as an integer. It returns nothing.

#### 3.12.2 printb

The function printb takes a single bool and prints it in its own line as a boolean (lowercase “true” or “false”). It returns nothing.

#### 3.12.3 printc

The function printc takes a single char and prints it in its own line as a character. It returns nothing.

#### 3.12.4 prints

The function prints takes a null-terminated char array (or equivalently, a string literal) and prints it in its own line as a string. It returns nothing.

#### 3.12.5 time

The function time takes no formal arguments. It returns the wall clock time, in microseconds, as an int.
4 Project Plan

4.1 Planning, Specification, Development, and Testing

RJEC was developed over the course of the semester in a series of full-group pair-coding meetings. During these pair-coding meetings, one person would share their screen and the team as a whole would make progress towards the code writing. These longer meetings would proceed as follows: at the start, we would review the features (usually one or two) we aimed to complete by the end of the meeting. Then, after debugging, we would immediately follow the feature creation with test writing in parallel. We maintained this procedure throughout, making sure to test immediately after feature completion and frequently, to ensure accurate and thorough feature implementation.

In addition, we scheduled and participated in weekly Friday meetings during which we discussed design decisions, reflected on progress, aligned project visions, identified areas of confusion that should be communicated with the TA or professor, performed sprint planning for the next deliverable, and scheduled the next sprint’s group-coding sessions. We roughly identify sprints as 2 week-long periods of execution, with anywhere from 3-4 or more full-group pair-coding sessions, depending on member availability and sprint plans.

When specifying project scope, our team worked with the professor in order to narrow the scope in order to ensure that it was feasible.

Alongside assignment deadlines, which we used for major checkpoints, we also scheduled our own soft deadlines before each course milestone. In this way, we were able to ensure that we would have time after completing each iteration in order to review the work before submitting and make any final changes accordingly.

4.2 Style Guide

RJEC was written with a combination of OCaml as well as C. Language definition was largely written in OCaml, and C was utilized in order to interface with system calls and the Libmill library to support concurrency-related features.

In an effort to write clean code and streamline development, we adhered to the following style guidelines for code:
1. When naming variable and function names, utilize the professor’s naming conventions if we are referencing them.

2. Match statements should be written in vertical blocks where all the bars are aligned.

3. Wild cards for invalid patterns in match statements should raise an error. If they are intended to cover future features, they should take the form of “〈feature〉 not implemented,” in which case this wild card should be removed when the feature is implemented.

4. "TODO" comments should be made to note features to be implemented in the future, along with a short description or ideation of potential feature implementation. This would be replaced with appropriate code later in development when the feature is completed.

5. Use snake_case for variable naming.

6. Follow the DRY principle, and make code as modular and reusable as possible. If a helper function would be used repeatedly in the code, define it in the scope accessible to all its use cases to avoid code duplication.

7. When working on each complex feature, write robust and exhaustive test cases to see if the feature is functioning as expected. Include both cases that should work and cases that shouldn’t.

8. Write clean and concise code where possible.

9. Work on complex features in their own branches and merge into master after the feature is complete and well tested.

10. Follow K&R style for braces and naming in C code.

11. Follow Go braces style in RJEC code, though non-cuddled else statements are allowed.

4.3 Project Timeline

The planning of our project was centered around the idea of iterative development.

We started out with brainstorming sessions where we fleshed out the syntax and semantics of the language, with which we finished the first draft of the Language
<table>
<thead>
<tr>
<th>Date</th>
<th>Checkpoint</th>
<th>Deadline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jan 15-21</td>
<td>Project idea brainstorming</td>
<td></td>
</tr>
<tr>
<td>Feb 1</td>
<td>Proposal finalized</td>
<td></td>
</tr>
<tr>
<td>Feb 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Feb 19</td>
<td>Scanner and Parser finalized</td>
<td></td>
</tr>
<tr>
<td>Feb 21-23</td>
<td>LRM writing</td>
<td></td>
</tr>
<tr>
<td>Feb 24</td>
<td></td>
<td></td>
</tr>
<tr>
<td>March 21</td>
<td>Hello world program working</td>
<td></td>
</tr>
<tr>
<td>March 24</td>
<td></td>
<td>Hello world</td>
</tr>
<tr>
<td>April 2</td>
<td>Variable declaration and assignment working</td>
<td></td>
</tr>
<tr>
<td>April 7</td>
<td>Struct-related features working</td>
<td></td>
</tr>
<tr>
<td>April 8</td>
<td>Function-related features working</td>
<td></td>
</tr>
<tr>
<td>April 10</td>
<td>Control flow features working</td>
<td></td>
</tr>
<tr>
<td>April 18</td>
<td>chan and yeet working</td>
<td></td>
</tr>
<tr>
<td>April 19</td>
<td>select working</td>
<td></td>
</tr>
<tr>
<td>April 21</td>
<td>Array and string-related features working</td>
<td></td>
</tr>
<tr>
<td>April 22-24</td>
<td>Wrap-up, report writing, final presentation prep</td>
<td>Final presentation</td>
</tr>
<tr>
<td>April 25</td>
<td></td>
<td>Final presentation</td>
</tr>
<tr>
<td>April 26</td>
<td></td>
<td>Final report</td>
</tr>
</tbody>
</table>

Table 1: Project Timeline

Reference Manual. Then, based on the established high-level ideas, we implemented the scanner and the parser to formalize the language into a context-free grammar, and resolved all reduce/reduce and shift/reduce conflicts to make sure the grammar is unambiguous.

We then worked on all components of the compiler, including the semantic checker and code generator in iterations, progressing one feature at a time. We started by trying to get the hello world program working, and then basic arithmetic operations and printing. We then built incrementally upon each feature to finish the next, from variable declaration and assignment, structs, function definitions and function calls, control flow (if, for, defer), concurrency features (yeet, chan, select), and finally arrays. We also implemented related test cases for each feature along the way.

With this principle, our eventual timeline is shown in table 1.
4.4 Team Roles and Responsibilities

We assigned the following preliminary roles:

Managers: Caroline Hoang, Elaine Wang
Language Gurus: Justin Chen, Riya Chakraborty
Systems Architects: Riya Chakraborty, Justin Chen
Testing: Elaine Wang, Caroline Hoang

In practice, everyone worked on everything in the code. As mentioned above, we worked in full group programming sessions, so for the most part, everyone was aware of everything which was done in the project.

4.5 Software Development Environment

We developed in Docker. The image can be pulled from ‘columbiasedwards/plt’. The environment uses Ubuntu 18.04.1 LTS.

We also did some early development in macOS 10.14 Mojave, but later switched fully to Docker for a consistent development environment.

Languages, libraries, and build tools:
OCaml 4.05.0
OCaml LLVM 6.0.0
LLVM 6.0.0
OCamlbuild 0.12.0
C11 with GNU extensions
GCC 7.3.0
glibc 2.27
GNU Make 4.1
CMake 3.10.2
GNU bash 4.4.19(1)
Libmill 1.18

We developed using VS Code with the official OCaml and clangd language servers. We used GitHub and git for version control. For the project reports, we used \LaTeX, on Overleaf.
4.6 Project Log

commit ae8836c6cd4ce79d0fb0847fd4f8c910f46045d9
Author: Justin Chen <justinbrianchen@yahoo.com>
Date:  Sun Apr 25 15:23:49 2021 -0700

    re-sign codegen

Author: Justin Chen <justinbrianchen@yahoo.com>
Date:  Sun Apr 25 11:26:59 2021 -0700

    fix more signing

commit cd84f61fd4a3c79792f54f357d30209aldafa16d
Author: Justin Chen <justinbrianchen@yahoo.com>
Date:  Fri Apr 23 15:22:37 2021 -0400

    fix signing

commit 33eda83476a1755f6193e0b9b0b48beef0240caf
Author: Justin Chen <justinbrianchen@yahoo.com>
Date:  Fri Apr 23 05:36:00 2021 -0400

    update latex compiler scripts

commit 12a252b2182b1017696123a782de7b491fc33755
Author: Justin Chen <justinbrianchen@yahoo.com>
Date:  Fri Apr 23 04:57:39 2021 -0400

    add latex conversion scripts

commit 284d76e4fb8301ae3935e19722a656f0d533100b
Merge: a840262 4f24368
Author: Cremebrutale <caroline.hoang@columbiaspectator.com>
Date:  Fri Apr 23 04:25:31 2021 -0400

    Merge branch 'master' of https://github.com/PLT-Project/RJEC

commit a840262c6f43c14ea1853b626a2bd340cd4e309
Author: Cremebrutale <caroline.hoang@columbiaspectator.com>
Date:  Fri Apr 23 04:22:53 2021 -0400

    rjecparse.mly additional changes

commit d420c53bbbeb753e4c67d7b3691f6066b90fed53b
Author: Cremebrutale <caroline.hoang@columbiaspectator.com>
Date: Fri Apr 23 04:21:18 2021 -0400

   cleaned rjecparse.mly

commits 8c9cb8bdc2f4bdc4b6e07b464074267ed6c90daf2c6a
Author: Cremebrutale <caroline.hoang@columbiaspectator.com>
Date: Fri Apr 23 04:04:15 2021 -0400

   cleaned up ast.ml

commits 4f243685c1d512fb6fc0cef6e162af73c237d91
Author: Riya Chakraborty <47572810+ayirr7@users.noreply.github.com>
Date: Fri Apr 23 02:50:35 2021 -0500

   Update README.md

commits da61608656418553ed5dee4ea33fd5856308572
Author: Riya Chakraborty <47572810+ayirr7@users.noreply.github.com>
Date: Fri Apr 23 02:50:19 2021 -0500

   Update README.md

commits 73cb936027651c145e332497e160f1aae9b071b6
Author: Cremebrutale <caroline.hoang@columbiaspectator.com>
Date: Fri Apr 23 03:45:26 2021 -0400

   cleaned up codegen.ml

commits 0456bc1d62dbfe0be9690df81bb801201c6f5186
Author: Justin Chen <justinbrianchen@yahoo.com>
Date: Fri Apr 23 02:26:56 2021 -0400

   update README

commits 0f68a230bed79ee974eb8ebe85c405d3e8248aal
Author: Justin Chen <justinbrianchen@yahoo.com>
Date: Fri Apr 23 02:14:40 2021 -0400

   update README

commits 45e8909790d676f83775eb8dd530ec41b94ae6e6
Author: Justin Chen <justinbrianchen@yahoo.com>
Date: Fri Apr 23 02:14:00 2021 -0400

   add README

Riya Chakraborty, Justin Chen, Yuanyuting Wang, Caroline Hoang
commit 245460c2fd0973e0d5e9ea9a995e24d29b574fe6
Author: Justin Chen <justinbrianchen@yahoo.com>
Date: Fri Apr 23 01:18:50 2021 -0400

  limit equality to basic types

commit c438fa43084f9d8986b8f15e170dad780705fece
Author: Justin Chen <justinbrianchen@yahoo.com>
Date: Fri Apr 23 01:03:41 2021 -0400

  clean up comments

commit 3e8d3c1bf80c41164d9deb4fd9d9a0bd2ddb71eb
Author: Justin Chen <justinbrianchen@yahoo.com>
Date: Fri Apr 23 00:58:06 2021 -0400

  code signing

commit 0c0383ab732e75d92fe1732b4c57fbb076ea71
Author: Justin Chen <justinbrianchen@yahoo.com>
Date: Fri Apr 23 00:55:46 2021 -0400

  code signing and cleanup

commit 18a8c8a6addf61de1463bb0534e0add835662241
Merge: c027b81 d0c3f38
Author: Justin Chen <justinbrianchen@yahoo.com>
Date: Thu Apr 22 18:01:44 2021 -0400

  Merge branch 'master' of https://github.com/PLT-Project/RJEC

commit c027b8124b16556f07037c1538e4c4f09c3c1793
Author: Justin Chen <justinbrianchen@yahoo.com>
Date: Thu Apr 22 18:01:39 2021 -0400

  mutex demo

commit d0c3f387e567ef1387143f5101856914d2e871c5
Merge: d511c1b 94d9397
Author: Riya Chakraborty <riyamch@yahoo.com>
Date: Thu Apr 22 16:29:58 2021 -0500

  Merge branch 'master' of https://github.com/PLT-Project/RJEC into fail_cases

commit d511c1b20242d728609022fdbdd728c365d1a9c0
modified parser and fail cases

commit 94d9397bbe88390b5adcd823eb1c299085c8050b
Merge: eb07e60 a7fcd97
Author: Justin Chen <justinbrianchen@yahoo.com>
Date: Thu Apr 22 17:21:35 2021 -0400

Merge branch 'master' of https://github.com/PLT-Project/RJEC

commit eb07e60c81af90369781d990e61a0225f5f45531
Author: Justin Chen <justinbrianchen@yahoo.com>
Date: Thu Apr 22 17:21:31 2021 -0400

cleanup mapreduce

commit 9da2aabd2c3a184ced9539403d586f1b79a22c39
Merge: 828481a a7fcd97
Author: Riya Chakraborty <riyamch@yahoo.com>
Date: Thu Apr 22 15:39:17 2021 -0500

Merge branch 'master' of https://github.com/PLT-Project/RJEC into fail_cases
Merge master

commit 828481a0f15fb9173383215a38f33f5525f6df2a
Author: Riya Chakraborty <riyamch@yahoo.com>
Date: Thu Apr 22 15:39:06 2021 -0500

add more fail cases

commit a7fcd9790f044e8dad160e751c00d8f569a95aab
Merge: 67cca4a f0d39ba
Author: Yuanyuting (Elaine) Wang <33389045+wangyyt1013@users.noreply.github.com>
Date: Thu Apr 22 16:29:48 2021 -0400

Merge pull request #9 from PLT-Project/comp-warnings

fixed basic compilation warnings

commit f0d39ba474471c25dba573d2466296ee4b337b7f
Author: Yuanyuting Wang <yw3241@columbia.edu>
Date: Thu Apr 22 16:24:45 2021 -0400
fixed basic compilation warnings

commit 554ab6dbbbda2e8051e3b52bf40e66c55adfd4f6
Author: Justin Chen <justinbrianchen@yahoo.com>
Date: Thu Apr 22 15:45:36 2021 -0400

fix bug in mapreduce

commit 67cca4a2a9019a41bb7167db508c17023774910c
Author: Justin Chen <justinbrianchen@yahoo.com>
Date: Thu Apr 22 14:42:02 2021 -0400

add timing and comparison with iterative solution for mapreduce

commit 7207f176573db693e6c65440d159d77b9683f199
Author: Justin Chen <justinbrianchen@yahoo.com>
Date: Thu Apr 22 05:46:57 2021 -0400

add mapreduce demo and fix local variable size array declaration

commit 16ba82517fbbf5be0161af736f2420cdc82ca986a
Author: Justin Chen <justinbrianchen@yahoo.com>
Date: Thu Apr 22 03:16:22 2021 -0400

compiler script + begin demo programs

commit 035ba55d301bc6068e8a7f61e5e15f551d2d8068
Author: Justin Chen <justinbrianchen@yahoo.com>
Date: Thu Apr 22 03:00:24 2021 -0400

add timer built-in function

commit 8c64fc554637cc4c16f76eb467211633a6df2b7b
Author: Justin Chen <justinbrianchen@yahoo.com>
Date: Thu Apr 22 02:20:39 2021 -0400

channels working with send/receive/select and array subscript

commit 0a66e0c493de51b3e262ec8994ce3c29734176cf
Author: Justin Chen <justinbrianchen@yahoo.com>
Date: Thu Apr 22 01:37:30 2021 -0400

parser and ast for arrays of channels

commit 1292c558c18974cfc5a700aa6f9bf0900b922d02

Riya Chakraborty, Justin Chen, Yuanyuting Wang, Caroline Hoang
update grammar to support arrays of chans for send/recv/select

commit 3dcb48cab9c1890ade971c8b746a84393eed6a45
Merge: a5bd492 e06261c
Author: Yuanyuting (Elaine) Wang <33389045+wangyyt1013@users.noreply.github.com>
Date: Wed Apr 21 02:10:07 2021 -0400

Merge pull request #8 from PLT-Project/array

Array

commit e06261c4f84961c936d3d4ebb9923893f6c98279
Author: Yuanyuting Wang <yw3241@columbia.edu>
Date: Wed Apr 21 02:02:59 2021 -0400

added '\0'

commit aa9a553619f5a7fee6aebb510428974a18225fe8
Author: Yuanyuting Wang <yw3241@columbia.edu>
Date: Wed Apr 21 01:46:31 2021 -0400

string literal <-> char arrays kinda working

commit 2bb290962af4771b2a9bd1a41c86cff91912386d
Author: Yuanyuting Wang <yw3241@columbia.edu>
Date: Wed Apr 21 01:17:04 2021 -0400

select/array of chans test case

commit 037d4c85c5ed9ed0e6ad808c67c0fd8442817c96
Author: Yuanyuting Wang <yw3241@columbia.edu>
Date: Wed Apr 21 00:58:23 2021 -0400

array literals

commit f7e15810e0a3a2ee04cdadcc2c90469c06856981
Merge: d96a793 a5bd492
Author: Yuanyuting Wang <yw3241@columbia.edu>
Date: Tue Apr 20 23:45:00 2021 -0400

Merge branch 'master' of https://github.com/PLT-Project/RJEC
array access & assign; syntax such as a[i].x valid

add producer consumer test case

array declaration and subscript working?? need to fix subscript and access parsing

Merge pull request #7 from PLT-Project/select

Implementation of select

switch-branching codegen and test cases for select

semant and partial codegen (pass clauses into C function) for select

C code, parser, and AST for select
commit 98b0f6bde9cdacbd918930d98163e76258ealf6f
Author: Justin Chen <justinbrianchen@yahoo.com>
Date: Mon Apr 19 01:42:37 2021 -0400

delete accidentally committed file

commit f1ee3c1ef2eaba2fbd5bd3e8b6186d659bbf484fe8
Merge: b44dd22 fd88571
Author: Yuanyuting (Elaine) Wang <33389045+wangyyt1013@users.noreply.github.com>
Date: Mon Apr 19 00:39:55 2021 -0400

Merge pull request #6 from PLT-Project/chan

Chan

commit fd88571005035655db3ade17d197fa5806d16ce
Author: Yuanyuting Wang <yw3241@columbia.edu>
Date: Mon Apr 19 00:35:48 2021 -0400

modified test case for close

commit 7545f255fa00de9ef54308a90b6676d03f8a45c8
Author: Yuanyuting Wang <yw3241@columbia.edu>
Date: Sun Apr 18 23:49:23 2021 -0400

channel with close kinda working (block on receive when channel is closed)

commit 9035eb70ac26de709e6d9859f2b21957a24d4e2d
Author: Yuanyuting Wang <yw3241@columbia.edu>
Date: Sun Apr 18 23:10:35 2021 -0400

send and recv working (?)

commit cb7553e751b21245bab9459b212feb114beae0ec
Author: Yuanyuting Wang <yw3241@columbia.edu>
Date: Sun Apr 18 21:58:36 2021 -0400

declaring and assigning channels working?

commit b44dd2204808a50b0e89a2b97478d1766eea37c
Merge: f456924 7e79ede
Author: Yuanyuting (Elaine) Wang <33389045+wangyyt1013@users.noreply.github.com>
Date: Sun Apr 18 20:57:45 2021 -0400

Merge pull request #5 from PLT-Project/yeet

 Yeet

commit 7e79ede692e273dbca7e232f7f47dedcc1c57835
Author: Yuanyuting Wang <yw3241@columbia.edu>
Date: Sun Apr 18 19:44:02 2021 -0400

 yeet working

commit 833481474c9c7ba13e0c5249aa570c8d3555295d
Author: Yuanyuting Wang <yw3241@columbia.edu>
Date: Sun Apr 18 19:26:55 2021 -0400

 yeet kinda working?? but struct literal had issues

commit 810f28617316aa4e259c2bd4f08d9bbec5fd3b71
Author: Yuanyuting Wang <yw3241@columbia.edu>
Date: Sun Apr 18 19:46:46 2021 -0400

 mismatch in function type

commit f456924c4e617e0a5aefd4036ee53123a79725b3
Merge: 6186c11 eaad431
Author: Yuanyuting (Elaine) Wang <33389045+wangyyt1013@users.noreply.github.com>
Date: Sun Apr 18 17:55:31 2021 -0400

 Merge pull request #4 from PLT-Project/function_void_ptr_args

 in codegen, all functions get void* as input arg

commit eaad4316456b53c032d875ddf4e5b42a13a6d5a6
Author: Yuanyuting Wang <yw3241@columbia.edu>
Date: Sun Apr 18 17:49:20 2021 -0400

 in codegen, all functions get void* as input arg

commit 6186c11c88f97883ab55bd8a31d1404246fbbf30
Merge: 6f8b5b1 1971f49
Author: Justin Chen <41352045+justinbchen@users.noreply.github.com>
Date: Sun Apr 18 17:01:21 2021 -0400

 Merge pull request #3 from PLT-Project/concurrency

Riya Chakraborty, Justin Chen, Yuanyuting Wang, Caroline Hoang
Concurrency library setup

commit 1971f49d14d42f97f83a1ba74a33e49ba4a4e165
Author: Justin Chen <justinbrianchen@yahoo.com>
Date: Sun Apr 18 17:00:20 2021 -0400

switch to void * function and cleanup

commit 0647ce9d9831b776dd01163b49fae1a33e9ff53f
Author: Justin Chen <justinbrianchen@yahoo.com>
Date: Mon Apr 12 03:30:58 2021 -0400

completed library setup for libmill

commit 64fb2a9d961cadc3c388b854533e016f8f897615
Author: Justin Chen <justinbrianchen@yahoo.com>
Date: Mon Apr 12 02:37:59 2021 -0400

library build setup

commit eff65372451af196074898400ace7937f7d87a63
Author: Justin Chen <justinbrianchen@yahoo.com>
Date: Mon Apr 12 02:31:05 2021 -0400

copy newer version of libmill

commit 70c61bc066ebdbd72b70ef37b861ba801166e1a1
Author: Justin Chen <justinbrianchen@yahoo.com>
Date: Mon Apr 12 01:45:20 2021 -0400

add libmill library

commit 6f8b5b182cadde8531aaab66cb6b51833a2b2720
Author: Yuanyuting Wang <yw3241@columbia.edu>
Date: Mon Apr 12 00:10:16 2021 -0400

implemented defer

commit 1900ae51ff3c5130162018d3d05aa38a60382824
Author: Riya Chakraborty <riyamch@yahoo.com>
Date: Sat Apr 10 00:49:15 2021 -0500

added support for for loops and tests

commit 796a0bac82e3591399cd9b38f2f58fec92acfd53
added modification of struct members and tests

commit 790b8bd973c88f0a84eeecfd48ae553615c82f25c
Author: Riya Chakraborty <riyamch@yahoo.com>
Date:   Fri Apr 9 20:34:33 2021 -0500

added support for Init operator

commit b642f8214e41d0a4805499401ad2752f0af9a0d2
Author: Riya Chakraborty <riyamch@yahoo.com>
Date:   Fri Apr 9 19:29:28 2021 -0500

test cases for decl + assign

commit 987aac2c5b4578a7b35a95bca51af9beeba80901
Author: Riya Chakraborty <riyamch@yahoo.com>
Date:   Fri Apr 9 19:18:06 2021 -0500

Decl + Assign same line

commit fd140ff34a37061f10a535841e33f3c7853e6237
Author: Yuanyuting Wang <yw3241@columbia.edu>
Date:   Thu Apr 8 01:35:08 2021 -0400

functions with single return typ working

commit cb0d4b2ece769b89a7f30d07ec66a231dba69228
Author: Yuanyuting Wang <yw3241@columbia.edu>
Date:   Wed Apr 7 23:57:39 2021 -0400

structs working

commit d80d7dfa430322e55e039fa44e56c28efbc17c18
Author: Yuanyuting Wang <yw3241@columbia.edu>
Date:   Wed Apr 7 23:20:57 2021 -0400

struct lit & access; test still broken

commit d82c45100b89460927e293e050ce5b1f2c6f42d2
Author: Yuanyuting Wang <yw3241@columbia.edu>
Date:   Wed Apr 7 21:36:20 2021 -0400

struct declaration working
commit 494b020ee51ade7d1e6a0f9e36b30d887ed6862b
Author: Yuanyuting Wang <yw3241@columbia.edu>
Date: Tue Apr 6 02:16:51 2021 -0400

    added support for structlit in semant; added struct_decls in codegen

commit ec7b3ae764197bb884a3ae9439bf93a2c65dd6b9
Author: Yuanyuting Wang <yw3241@columbia.edu>
Date: Mon Apr 5 23:50:39 2021 -0400

    changed semant for dealing with multiple return types

commit 0319197e8f9f9f54c976468b98f7c2a13ff3fb28
Merge: 4838813 96dc946
Author: Riya Chakraborty <47572810+ayirr7@users.noreply.github.com>
Date: Fri Apr 2 14:27:24 2021 -0500

    Merge pull request #2 from PLT-Project/local_vars

    Supporting Local Variable Declarations

commit 96dc946e0e8f9f54c976468b98f7c2a13ff3fb28
Author: Riya Chakraborty <riyamch@yahoo.com>
Date: Fri Apr 2 14:26:23 2021 -0500

    tests for local variable decls

commit 523e0d4c9c0ca97e46ab836098baabf88073c8f9
Author: Riya Chakraborty <riyamch@yahoo.com>
Date: Fri Apr 2 14:25:09 2021 -0500

    completed codegen for local variable decls

commit 687caa6983c9c6f9dc21ebe31b12017b447b300
Author: Yuanyuting Wang <yw3241@columbia.edu>
Date: Thu Apr 1 02:05:49 2021 -0400

    halfway through working with codegen for local vars

commit 483881316b7011ddfd1921b4b1ec2ccbebecc2f
Author: Yuanyuting Wang <yw3241@columbia.edu>
Date: Thu Apr 1 01:40:45 2021 -0400

    Fixed indentation for codegen
commit 18a90965929f0e3042c5cf9ab6a898f3ba016866
Author: Yuanyuting Wang <yw3241@columbia.edu>
Date: Thu Apr 1 01:10:05 2021 -0400

updated semant for local vars and scopes

commit af97606a8a84a3a5da6833f90d9902aca13945
Author: Riya Chakraborty <riyamch@yahoo.com>
Date: Fri Mar 26 21:38:05 2021 -0500

More assign tests

commit 684ae42d231d6643b5c58670ed5ac80473221c29
Author: Riya Chakraborty <riyamch@yahoo.com>
Date: Fri Mar 26 21:22:23 2021 -0500

Add assign and tests

commit c7e08cad92079c4da24259677f7729e1b5729a2
Author: Riya Chakraborty <riyamch@yahoo.com>
Date: Fri Mar 26 23:59:01 2021 +0000

added if/ops tests

commit 4343d49c1924d04bb090bb9539077eel23cbe468
Author: Justin Chen <justinbrianchen@yahoo.com>
Date: Tue Mar 23 00:03:59 2021 -0400

unsuccessful attempt at printf

commit 7024295b0d5f8dc511866626ec6f19293ed3d31b
Author: Justin Chen <justinbrianchen@yahoo.com>
Date: Mon Mar 22 19:38:41 2021 -0400

working print for string and char

commit 8715ec256fcb1cd10b649525f632514522c413b2
Author: Justin Chen <justinbrianchen@yahoo.com>
Date: Mon Mar 22 04:37:52 2021 -0400

fix assignstmt fix

commit d812a7b63342d91717c7fa2fcec7554093e68bf06
Author: Justin Chen <justinbrianchen@yahoo.com>
Date: Mon Mar 22 04:17:48 2021 -0400
remove unused trd function

commit 6b81ae8a3d5a29e9654287af5a6756ac66bccd4f
Merge: 0105727 7e52cf0
Author: Justin Chen <41352045+justinbchen@users.noreply.github.com>
Date: Sun Mar 21 17:03:02 2021 -0400

Merge pull request #1 from PLT-Project/hello

Hello World

commit 7e52cf0f80e9801860217b95d599f3473c996677
Author: Justin Chen <justinbrianchen@yahoo.com>
Date: Sun Mar 21 17:01:52 2021 -0400

working regression test for hello

commit 57de2b6fa987f77b12b40d6c6ebc2706a0987d4c
Author: Justin Chen <justinbrianchen@yahoo.com>
Date: Sun Mar 21 16:37:44 2021 -0400

working hello world (but with non-zero return value)

commit 94641a3666d7aa17130794cf7ad9810a433f1386
Author: Justin Chen <justinbrianchen@yahoo.com>
Date: Fri Mar 19 22:01:34 2021 -0400

first attempt on code gen

commit 0105727bab9370a328e7c976f732c1e482b69a8a
Author: Justin Chen <justinbrianchen@yahoo.com>
Date: Fri Mar 19 17:00:04 2021 -0400

fix func type definition in semantic checking

commit cc0ddf23d02eeb9d906915b520ea00426cd4701
Author: Justin Chen <justinbrianchen@yahoo.com>
Date: Tue Mar 16 22:05:19 2021 -0400

preliminary semantic checking

commit 06261f3132fa5bf8d967alca5149fbb6dae5b258
Author: Justin Chen <justinbrianchen@yahoo.com>
Date: Thu Mar 11 16:09:02 2021 -0500
replace While with For and MakeBuffered with Make with optional buffer

commit 5725627ad56b53f31ce185ad768a38e21155533e2
Author: Justin Chen <justinbrianchen@yahoo.com>
Date: Sun Mar 7 22:54:01 2021 -0500

first compiling AST

commit 9d1a9bb5cdc14ee005adfefac0c9e0ca3bb91520
Author: Justin Chen <justinbrianchen@yahoo.com>
Date: Sun Mar 7 21:30:01 2021 -0500

add AST from microC

commit 02031ff27db6672aa51af00be4bf83f14369c9bd
Author: Justin Chen <justinbrianchen@yahoo.com>
Date: Tue Mar 2 21:27:02 2021 -0500

remove TYPE from scanner

commit 1084a522ba8d0f0a8a128717322a5391c3724d5a
Author: Justin Chen <justinbrianchen@yahoo.com>
Date: Wed Feb 24 00:10:04 2021 -0500

fix typo in case grammar

commit 0aea34939c7bc349b57ff18df132bbd4985412c9
Author: Justin Chen <justinbrianchen@yahoo.com>
Date: Tue Feb 23 23:59:59 2021 -0500

fix issues with case grammar

commit b367292fc0dd5ea023cd3e63c463df413baf3cec
Author: Justin Chen <justinbrianchen@yahoo.com>
Date: Tue Feb 23 23:45:42 2021 -0500

define array and struct literals; change struct definition format

commit f4288a6740ec095daddc31daf6956eaf254fd7d4
Author: Justin Chen <justinbrianchen@yahoo.com>
Date: Tue Feb 23 22:42:51 2021 -0500

for statements use assign-stmt

commit 26eb977960f55525b85409829e62ac87d2d8fedfe
struct takes only basic types as members

commit 30a8424aac854acee483130e4f02f1111928371e
Author: Justin Chen <justinbrianchen@yahoo.com>
Date: Tue Feb 23 03:00:55 2021 -0500

allow variable declaration without assignment inside functions

commit 8def2ab9507349e0208ce7729ce817cdeb4eb901
Author: Justin Chen <justinbrianchen@yahoo.com>
Date: Tue Feb 23 02:57:12 2021 -0500

rename typ_opt to return_types and typ_formal_opt to typ_opt

commit 2efbf4b7426f30ebfdb9996d10b413efc3672e61
Author: Justin Chen <justinbrianchen@yahoo.com>
Date: Tue Feb 23 02:55:15 2021 -0500

rename chan_typ to basic_typ

commit fb0616e0754282805ca60d97520d3cad92e1b50f
Author: Justin Chen <justinbrianchen@yahoo.com>
Date: Tue Feb 23 02:53:22 2021 -0500

move assign_stmt to be with the other statements

commit 273b527744f640840e9e7e137bed06a59d776276
Author: Justin Chen <justinbrianchen@yahoo.com>
Date: Mon Feb 22 20:23:50 2021 -0500

fix channel type definition

commit d44c9d33e08e17a20f297115d7340bf5e8448a5f
Author: Justin Chen <justinbrianchen@yahoo.com>
Date: Mon Feb 22 18:40:11 2021 -0500

add back unary minus

commit 832337b9be6843186e0f1c1b4819dcbe232b4508
Author: Justin Chen <justinbrianchen@yahoo.com>
Date: Sun Feb 21 18:29:10 2021 -0500

fixed grammar for function return types
commit 358c90d546cd0ae43b5f710d8cda83759c8fda61
Author: Justin Chen <justinbrianchen@yahoo.com>
Date: Sun Feb 21 18:14:53 2021 -0500

    fix else if + preliminary function type as parameter

commit 21d507bc4a09595132e4d45ec792031ed2d4b988
Author: Justin Chen <justinbrianchen@yahoo.com>
Date: Sun Feb 21 17:38:03 2021 -0500

    support multiple return, assignment, declaration

commit 7ab3a99737e36bd049296e504bcb6ad5099b9378
Author: Justin Chen <justinbrianchen@yahoo.com>
Date: Fri Feb 19 23:56:55 2021 -0500

    array with size

commit 59465eaa98772e8ebde95bba9f0883c1c8879b4e
Author: Justin Chen <justinbrianchen@yahoo.com>
Date: Fri Feb 19 23:40:32 2021 -0500

    fixed Blocks and redundant vdecl list

commit 80667a1690a7ae8bb7bae341ede7f943dddb224e6
Author: Justin Chen <justinbrianchen@yahoo.com>
Date: Fri Feb 19 23:04:20 2021 -0500

    first attempt at scanner/parser

commit 83d00ec586c2b39161673485d1d943c7136b2843
Author: Justin Chen <justinbrianchen@yahoo.com>
Date: Fri Feb 19 17:42:07 2021 -0500

    copy over scanner/parser/dockerfile

commit eaf5c96ed80e85a3b9ba17225f02b9a2c5696b0c
Author: Justin Chen <justinbrianchen@yahoo.com>
Date: Fri Feb 19 17:39:45 2021 -0500

    move microc stuff into their own folder
5 Architectural Design

The RJEC compiler consists of several components that, through an established flow, converts a source code for a RJEC program into an executable file. The overall architecture is shown below:

5.1 Lexer

The lexer, implemented in scanner.mll, takes in a stream of ASCII symbols from the source code, and outputs a series of tokens to be parsed in the parser. Tokens supported by the parser, including their associativity and precedence, are defined
in the `rjecparse.mly` file. Since RJEC supports multi-line comments, the scanner ignores all the symbols after each opening comment token `/*`, and only returns to the normal parsing mode after encountering an ending comment token `*/`. If any invalid combination of symbols is found, the lexer will throw an error.

### 5.2 Parser

The parser, implemented in `rjecparse.mly`, parses the stream of tokens inputted from the lexer, and outputs an abstract syntax tree (AST) according to the context-less grammar rules specified in the parser. The valid nodes for the AST are defined in `ast.ml`. If there’s any token pattern that violates the grammar rules specified, a parsing error will be thrown.

### 5.3 Semantic Checking and Transformation

The semantic checker, implemented in `semant.ml`, takes in an AST outputted from the parser and outputs a semantically-checked AST (SAST). The semantic checker recursively walks through the AST, checking for type errors in the evaluation of each node. It also checks for scoping errors by constructing C-styled symbol tables that register all variables that the currently executing statement/expression has access to. The semantic checker then convert a valid AST into a SAST with semantically-checked nodes, which are defined in the `sast.ml` file.

Some semantic transformation is also performed here. For instance, in the short-form variable declaration, the evaluated type of the expression is used as the type for the variable declaration stored in the SAST.

### 5.4 Code Generation

The SAST outputted by the semantic checker is then passed into the LLVM code generator, implemented in `codegen.ml`, which generates an intermediate representation of the program in an LLVM module.

For basic operations such as arithmetics, code generator directly makes calls to the LLVM library in OCaml.

For variable declarations and assignments, the code generator constructs symbol tables that store the currently declared variables, their type specifications, as well
as their memory locations, and keep track of the tables as it goes through variable declaration / assignment statements.

For structs, the code generator converts struct definitions in the source code into LLVM types defined by the member fields of each struct definition. When a struct variable is declared, the code generator allocates a memory space for this user-defined type, and stores its member fields, which can only have basic types, by value. Upon declaration, the member fields are initialized with their default values. When assigning a struct variable or passing it into a function, a copy of the struct value is passed around.

For arrays, the code generator allocates space on the heap for a fixed number of data of the specific type for each array. The fixed array size comes from evaluating the array size expression during runtime, which means it can come from a variable. All the elements in an array are also initialized by their default values upon declaration.

For support of concurrency features, including channels, yeetroutines, and select, the code generator interfaces with our imported C code, which uses the Libmill library that provides go-like concurrency features implemented in C. For select especially, we also conducted control flow manipulation in the code generator so that the case where the send/receive through a channel is successful gets executed.

### 5.5 Imported C Code

Part of our built-in functions, such as `time`, the printing suite and the functions related to concurrency, are written in C, linked into the compilation, and called in the LLVM code. Aside from the common C libraries (`stdbool.h`, `string.h`, `sys/time.h`, `stdint.h`, `stdio.h`), we also imported the Libmill library by Martin Sustrik (`libmill/libmill.h` and `libmill/chan.h`).

### 5.6 Division of Labor

Everyone participated in the collective brainstorming and iterative design of the RJEC grammar, syntax, and semantics. Everyone also equally contributed during group meetings and group programming sessions where we fleshed out further implementation details of the overall architecture.
6 Test Plan

6.1 Example Test Programs

6.1.1 Single Producer-Consumer

RJEC Code:

```go
/* simple producer-consumer problem */

func foo(ch1 chan char, ch2 chan int, quit chan bool) {
    for {
        select {
            case ch1 <- 'a':
                printb("sending a in foo... ");
            case val2 := <- ch2:
                printb("receiving in foo: ");
                printi(val2);
            case q := <- quit:
                printb(q);
                if q {
                    printb("quitting...");
                }
                return;
        }
    }
}

cfunc main() {
    ch1 := make(chan char);
    ch2 := make(chan int);
    quit := make(chan bool, 10);
    yeet foo(ch1, ch2, quit);
    for i := 0; i < 5; i = i + 1 {
        c := <- ch1;
        printb("received in main: ");
        printc(c);
        printb("sending i in main...");
        ch2 <- i;
    }
    quit <- true;
}
```

LLVM Code:
; ModuleID = 'RJEC'
source_filename = "RJEC"

@fmt = private unnamed_addr constant [4 x i8] c"\%d\n\00"
@fmt.1 = private unnamed_addr constant [4 x i8] c"\%c\n\00"
@fmt.2 = private unnamed_addr constant [4 x i8] c"\%s\n\00"
@strlit = private unnamed_addr constant [18 x i8] c"received in main :
\00"
@strlit.3 = private unnamed_addr constant [21 x i8] c"sending i in main...
\00"
@fmt.4 = private unnamed_addr constant [4 x i8] c"\%d\n\00"
@fmt.5 = private unnamed_addr constant [4 x i8] c"\%c\n\00"
@fmt.6 = private unnamed_addr constant [4 x i8] c"\%s\n\00"
@strlit.7 = private unnamed_addr constant [20 x i8] c"sending a in foo...
\00"
@strlit.8 = private unnamed_addr constant [18 x i8] c"receiving in foo:
\00"
@strlit.9 = private unnamed_addr constant [12 x i8] c"quitting ...
\00"

declare i32 @printf(i8*, ...)
declare i32 @get_time()
declare i32 @printbool(i1)
declare void @yeet(i32 (i8*)*, i8*)
declare i8* @makechan(i8, i32)
declare void @send(i8*, i8, i32)
declare i32 @recv_int(i8*)
declare i1 @recv_bool(i8*)
declare i8 @recv_char(i8*)
declare void @closechan(i8*, i8)
declare i32 @selectchan({ i8, i8* , i8* , i64 }*, i32)

define i32 @main(i8*) {
  entry:

%main_args_ptr = bitcast i8* %0 to {}*
%main_args = load {}, {}* %main_args_ptr
%ch1 = alloca i8*
store i8 null, i8** %ch1
%makechan = call i8* @makechan(i8 99, i32 0)
store i8* %makechan, i8** %ch1
%ch2 = alloca i8*
store i8 null, i8** %ch2
%makechan1 = call i8* @makechan(i8 105, i32 0)
store i8* %makechan1, i8** %ch2
%quit = alloca i8*
store i8 null, i8** %quit
%makechan2 = call i8* @makechan(i8 98, i32 10)
store i8* %makechan2, i8** %quit
%quit3 = load i8, i8** %quit
%ch15 = load i8*, i8** %ch2
%malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64
ptrtoint (i1 **
getelementptr (i1** getelementptr (i1*, i1** null, i32 1) to i64),
i64 3) to i32))
%foo_args = bitcast i8* %malloccall to { i8*, i8*, i8* }*
%arg_0 = insertvalue { i8*, i8*, i8* } zeroinitializer, i8* %ch15, 0
%arg_1 = insertvalue { i8*, i8*, i8* } %arg_0, i8* %ch24, 1
%arg_2 = insertvalue { i8*, i8*, i8* } %arg_1, i8* %quit3, 2
store { i8*, i8*, i8* } %arg_2, { i8*, i8*, i8* }* %foo_args
%foo_arg_pointer = bitcast { i8*, i8*, i8* }* %foo_args to i8*
call void @yeet(i32 (i8* (i8*)* @foo, i8* %foo_arg_pointer)
%i = alloca i32
store i32 0, i32* %i
store i32 0, i32* %i
br label %while
while: ; preds = %
    while_body, %entry
    %i13 = load i32, i32* %i
    %tmp14 = icmp slt i32 %i13, 5
    br i1 %tmp14, label %while_body, label %merge
while_body: ; preds = %while
    %c = alloca i8
    store i8 0, i8* %c
    %ch16 = load i8*, i8** %ch1
    %recv_int = call i8 @recv_char(i8* %ch16)
    store i8 %recv_int, i8* %c

Riya Chakraborty, Justin Chen, Yuanyuting Wang, Caroline Hoang
COMS W4115  RJEC Final Report

%printf = call i32 (i8 * ..., ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.2, i32 0, i32 0), i8* getelementptr inbounds ([18 x i8], [18 x i8]* @strlit, i32 0, i32 0))
%ch = load i8, i8* %c
%printf = call i32 (i8 * ..., ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.1, i32 0, i32 0), i8 %c7)
%ch = load i8, i8* %c
%printf = call i32 (i8 * ..., ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.2, i32 0, i32 0), i8* getelementptr inbounds ([21 x i8], [21 x i8]* @strlit.3, i32 0, i32 0))
%ch2 = load i8*, i8** %ch2
%ch1 = extractvalue { i8*, i8*, i8* } %foo_args, 0
%ch2 = extractvalue { i8*, i8*, i8* } %foo_args, 1
%quit = extractvalue { i8*, i8*, i8* } %foo_args, 2
%ch1 = alloca i8*
%ch2 = alloca i8*
%quit = alloca i8*
store i8* %ch1, i8** %ch11
%ch2 = alloca i8*
%ch2 = alloca i8*
%quit = alloca i8*
store i8* %ch2, i8** %ch22
%quit = alloca i8*
store i8* %quit, i8** %quit3
br label %while

while: ; preds = %merge, %entry
br il true, label %while_body, label %merge25

while_body: ; preds = %while
%malloccall = tail call i8* @malloc(i32 mul (i32 ptrtoint ({ i8,
i8*, i8*, i64 }* getelementptr ({ i8, i8*, i8*, i64 }, { i8,
i8*, i8*, i64 }* null, i32 1) to i32), i32 3))
%1 = bitcast i8* %malloccall to { i8, i8*, i8*, i64 }*
%ch14 = load i8*, i8** %ch11
%2 = alloca i8
store i8 97, i8* %2
%clause = insertvalue { i8, i8*, i8*, i64 } { i8 115, i8* null, i8*
null, i64 0 }, i8* %ch14, 1
%clause5 = insertvalue { i8, i8*, i8*, i64 } %clause, i8* %2, 2
%clause6 = insertvalue { i8, i8*, i8*, i64 } %clause5, i64
ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i64), 3
%3 = getelementptr { i8, i8*, i8*, i64 }, { i8, i8*, i8*, i64 }*
%1, i32 0
store { i8, i8*, i8*, i64 } %clause6, { i8, i8*, i8*, i64 }* %3
%ch27 = load i8*, i8** %ch22
%4 = alloca i32
store i32 0, i32* %4
%5 = bitcast i32* %4 to i8*
%clause8 = insertvalue { i8, i8*, i8*, i64 } { i8 114, i8* null, i8*
null, i64 0 }, i8* %ch27, 1
%clause9 = insertvalue { i8, i8*, i8*, i64 } %clause8, i8* %5, 2
%clause10 = insertvalue { i8, i8*, i8*, i64 } %clause9, i64
ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i64), 3
%6 = getelementptr { i8, i8*, i8*, i64 }, { i8, i8*, i8*, i64 }*
%1, i32 1
store { i8, i8*, i8*, i64 } %clause10, { i8, i8*, i8*, i64 }* %6
%quit11 = load i8*, i8** %quit3
%7 = alloca i1
store i1 false, i1* %7
%8 = bitcast i1* %7 to i8*
%clause12 = insertvalue { i8, i8*, i8*, i64 } { i8 114, i8* null, i8*
null, i64 0 }, i8* %quit11, 1
%clause13 = insertvalue { i8, i8*, i8*, i64 } %clause12, i8* %8, 2
%clause14 = insertvalue { i8, i8*, i8*, i64 } %clause13, i64
ptrtoint (i1* getelementptr (i1, i1* null, i32 1) to i64), 3
%9 = getelementptr { i8, i8*, i8*, i64 }, { i8, i8*, i8*, i64 }*
%1, i32 2
store { i8, i8*, i8*, i64 } %clause14, { i8, i8*, i8*, i64 }* %9
%select = call i32 @selectchan({ i8, i8*, i8*, i64 }* %1, i32 3)
%recv_val = load i32, i32* %4
%val12 = alloca i32
store i32 0, i32* %val12
store i32 %recv_val, i32* %val12
%recv_val20 = load i1, i1* %7
%q = alloca i1
store i1 false, i1* %q
store i1 %recv_val20, i1* %q
switch i32 %select, label %merge [
  i32 0, label %case
  i32 1, label %case15
  i32 2, label %case19
]

merge:
  ; preds = %
  while_body, %case15, %case
  br label %while

case:
  ; preds = %
  while_body
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.6, i32 0, i32 0), i8* getelementptr inbounds ([20 x i8], [20 x i8]* @strlit.7, i32 0, i32 0))
  br label %merge

case15:
  ; preds = %
  while_body
  %printf16 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.6, i32 0, i32 0), i8* getelementptr inbounds ([18 x i8], [18 x i8]* @strlit.8, i32 0, i32 0))
  %val217 = load i32, i32* %val2
  %printf18 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.4, i32 0, i32 0), i32 %val217)
  br label %merge

case19:
  ; preds = %
  while_body
  %q21 = load i1, i1* %q
  %printbool = call i32 @printbool(i1 %q21)
  %q22 = load i1, i1* %q
  br i1 %q22, label %then, label %else

merge23:
  ; preds = %else, %
  then
  ret i32 0

then:
  ; preds = %case19
  %printf24 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.6, i32 0, i32 0), i8* getelementptr inbounds ([12 x i8], [12 x i8]* @strlit.9, i32 0, i32 0))
### 6.1.2 Implementation of C-Styled Mutex

**RJEC Code:**

```go
/* implementation of a mutex using channels and demonstration with defer */

func make_mutex() chan bool {
    mu := make(chan bool, 1);
    mu <- true;
    return mu;
}

func lock(mu chan bool) {
    <- mu;
}

func unlock(mu chan bool) {
    mu <- true;
}

func foo(n int, mu chan bool) {
    if 30 < n {
        return;
    } else {
        lock(mu);
        defer unlock(mu);
    }

    for i := n; i < n + 10; i = i + 1 {
        printi(i);
    }
```
```go
func main() {
    mu := make_mutex();
    yeet foo(0, mu);
    yeet foo(25, mu);
    yeet foo(-100, mu);
    foo(-5, mu);
}
```

LLVM Code:

```llvm
; ModuleID = 'RJEC'
source_filename = "RJEC"

@fmt = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.1 = private unnamed_addr constant [4 x i8] c"%c\0A\00"
@fmt.2 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@fmt.3 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.4 = private unnamed_addr constant [4 x i8] c"%c\0A\00"
@fmt.5 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@fmt.6 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.7 = private unnamed_addr constant [4 x i8] c"%c\0A\00"
@fmt.8 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@fmt.9 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.10 = private unnamed_addr constant [4 x i8] c"%c\0A\00"
@fmt.11 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@fmt.12 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.13 = private unnamed_addr constant [4 x i8] c"%c\0A\00"
@fmt.14 = private unnamed_addr constant [4 x i8] c"%s\0A\00"

declare i32 @printf(i8*, ...)
declare i32 @get_time()
declare i32 @printbool(i1)
declare void @yeet(i32 (i8*)*, i8*)
declare i8* @makechan(i8, i32)
declare void @send(i8*, i8, i32)
declare i32 @recv_int(i8*)
```
declare i1 @recv_bool(i8*)
declare i8 @recv_char(i8*)
declare void @closechan(i8*, i8)
declare i32 @selectchan({ i8, i8*, i8*, i64 }*, i32)
define i32 @main(i8*) {
  entry:
  %main_args_ptr = bitcast i8* %0 to {}*
  %main_args = load {}, {}* %main_args_ptr
  %mu = alloca i8*
  store i8* null, i8** %mu
  %malloccall = tail call i8* @malloc(i32 0)
  %make_mutex_args = bitcast i8* %malloccall to {}*
  store {} zeroinitializer, {}* %make_mutex_args
  %make_mutex_arg_pointer = bitcast {}* %make_mutex_args to i8*
  %make_mutex_result = call i8* @make_mutex(i8* %
     make_mutex_arg_pointer)
  store i8* %make_mutex_result, i8** %mu
  %mul = load i8*, i8** %mu
  %malloccall12 = tail call i8* @malloc(i32 ptrtoint {{ i32, i8* }*
     getelementptr {{ i32, i8* }*, { i32, i8* }* null, i32 1} to i32} )
  %foo_args = bitcast i8* %malloccall12 to { i32, i8* }*
  %arg_1 = insertvalue {{ i32, i8* } zeroinitialize, i8* %mul, 1
  store { i32, i8* } %arg_1, { i32, i8* }* %foo_args
  %foo_arg_pointer = bitcast { i32, i8* }* %foo_args to i8*
  call void @yeet(i32 (i8*)* @foo, i8* %foo_arg_pointer)
  %mu3 = load i8*, i8** %mu
  %malloccall14 = tail call i8* @malloc(i32 ptrtoint {{ i32, i8* }*
     getelementptr {{ i32, i8* }*, { i32, i8* }* null, i32 1} to i32} )
  %foo_args5 = bitcast i8* %malloccall14 to { i32, i8* }*
  %arg_16 = insertvalue { i32, i8* } { i32 25, i8* null }, i8* %mu3,
  1
  store { i32, i8* } %arg_16, { i32, i8* }* %foo_args5
  %foo_arg_pointer7 = bitcast { i32, i8* }* %foo_args5 to i8*
  call void @yeet(i32 (i8*)* @foo, i8* %foo_arg_pointer7)
  %mu8 = load i8*, i8** %mu
  %malloccall19 = tail call i8* @malloc(i32 ptrtoint {{ i32, i8* }*
     getelementptr {{ i32, i8* }*, { i32, i8* }* null, i32 1} to i32} )
%foo_args10 = bitcast i8* %malloccall19 to { i32, i8* }*
%arg111 = insertvalue { i32, i8* } { i32 -100, i8* null }, i8* %mu8, 1
store { i32, i8* } %arg111, { i32, i8* }* %foo_args10
%foo_arg_pointer12 = bitcast { i32, i8* }* %foo_args10 to i8*
call void @yeet(i32 (i8* )* @foo, i8* %foo_arg_pointer12)
%mu13 = load i8*, i8** %mu
%malloccall14 = tail call i8* @malloc(i32 ptrtoint (( i32, i8* )* getelementptr {{ i32, i8* }}, { i32, i8* }* null, i32 1) to i32)

%foo_args15 = bitcast i8* %malloccall14 to { i32, i8* }*
%arg116 = insertvalue { i32, i8* } { i32 -5, i8* null }, i8* %mu13, 1
store { i32, i8* } %arg116, { i32, i8* }* %foo_args15
%foo_arg_pointer17 = bitcast { i32, i8* }* %foo_args15 to i8*
%1 = call i32 @foo(i8* %foo_arg_pointer17)
ret i32 0
}

define i32 @foo(i8*) {
entry:
%foo_args_ptr = bitcast i8* %0 to { i32, i8* }*
%foo_args = load { i32, i8* }, { i32, i8* }* %foo_args_ptr
%n = extractvalue { i32, i8* } %foo_args, 0
%mu = extractvalue { i32, i8* } %foo_args, 1
%n1 = alloca i32
store i32 %n, i32 %n1
%mu2 = alloca i8*
store i8* %mu, i8** %mu2
%n3 = load i32, i32* %n1
%tmp = icmp slt i32 30, %n3
br i1 %tmp, label %then, label %else
merge:
 ; preds = %else
%i = alloca i32
store i32 0, i32* %i
%n8 = load i32, i32* %n1
store i32 %n8, i32* %i
br label %while
then:
 ; preds = %entry
ret i32 0
else:
 ; preds = %entry
%mu4 = load i8*, i8** %mu2
%malloccall = tail call i8* @malloc(i32 ptrtoint (il** getelementptr (il*, il** null, i32 1) to i32))

%lock_args = bitcast i8* %malloccall to { i8* }*
%arg_0 = insertvalue { i8* } zeroinitializer, i8* %mu4, 0
store { i8* } %arg_0, { i8* }* %lock_args
%lock_arg_pointer = bitcast { i8* }* %lock_args to i8*
%1 = call i32 @lock(i8* %lock_arg_pointer)
%mu5 = load i8*, i8** %mu2
%malloccall6 = tail call i8* @malloc(i32 ptrtoint (il** getelementptr (il*, il** null, i32 1) to i32))

%unlock_args = bitcast i8* %malloccall6 to { i8* }*
%arg_07 = insertvalue { i8* } zeroinitializer, i8* %mu5, 0
store { i8* } %arg_07, { i8* }* %unlock_args
%unlock_arg_pointer = bitcast { i8* }* %unlock_args to i8*

br label %merge

while:
    ; preds = %
    while_body, %merge
    %i12 = load i32, i32* %i
    %n13 = load i32, i32* %n1
    %tmp14 = add i32 %n13, 10
    %tmp15 = icmp slt i32 %i12, %tmp14
    br i1 %tmp15, label %while_body, label %merge16

while_body:
    ; preds = %while
    %i9 = load i32, i32* %i
    %printf = call i32 (i8*, ... @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.3, i32 0, i32 0), i32 %i9)
    %i10 = load i32, i32* %i
    %tmp11 = add i32 %i10, 1
    store i32 %tmp11, i32* %i
    br label %while

merge16:
    ; preds = %while
    %2 = call i32 @unlock(i8* %unlock_arg_pointer)
    ret i32 0

}
define i32 @unlock(i8*) {
entry:
    %unlock_args_ptr = bitcast i8* %0 to { i8* }*
    %unlock_args = load { i8* }, { i8* }* %unlock_args_ptr
    %mu = extractvalue { i8* } %unlock_args, 0
    %mul = alloc %mul
    store i8* %mu, i8** %mul
6.1.3 Arrays of Structs

RJEC Code:

```c
struct foo {
    x int;
    y int;
    z bool;
}
```
func change_struct(x []struct foo) {
    x[0].x = x[1].x + 1;
    x[1].y = x[2].y - 2;
    x[2].z = true;
}

func main() {
    var x [10]struct foo;
    x[0] = struct foo { x: 1, y: 2 };
    x[1] = struct foo { x: 3, y: 5, z: true };
    x[2] = struct foo { x: 0, y: 7};
    change_struct(x);
    printi(x[0].x);
    printi(x[1].y);
    printb(x[2].z);
}

LLVM Code:

; ModuleID = 'RJEC'
source_filename = "RJEC"

@fmt = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.1 = private unnamed_addr constant [4 x i8] c"%c\0A\00"
@fmt.2 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@fmt.3 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.4 = private unnamed_addr constant [4 x i8] c"%c\0A\00"
@fmt.5 = private unnamed_addr constant [4 x i8] c"%s\0A\00"

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

declare void @closechan(i8*, i8)

declare i32 @selectchan({ i8, i8*, i8*, i64 }*, i32)

define i32 @main(i8*) {
  entry:
  %main_args_ptr = bitcast i8* %0 to {}*
  %main_args = load {}, {} * %main_args_ptr
  %malloccall = tail call i8* @malloc(i32 mul (i32 ptrtoint (( i32, i32, i1 )* getelementptr (i i32, i32, i1 ), { i32, i32, i1 }* null, i32 1) to i32), i32 10))
  %1 = bitcast i8* %malloccall to { i32, i32, i1 }*
  %x = alloca ( i32, i32, i1 )*
  store { i32, i32, i1 }* %1, { i32, i32, i1 }** %x
  %malloccall1 = tail call i8* @malloc(i32 ptrtoint (( i32, i32, i1 )* getelementptr (i i32, i32, i1 ), { i32, i32, i1 }* null, i32 1) to i32))
  %foo = bitcast i8* %malloccall1 to { i32, i32, i1 }*
  store { i32, i32, i1 } { i32 1, i32 2, i1 false }, { i32, i32, i1 }* %foo
  %tmp = load { i32, i32, i1 } { i32 1, i32 2, i1 false }, { i32, i32, i1 }* %foo
  %tmp2 = load { i32, i32, i1 }* %x
  %tmp3 = getelementptr { i32, i32, i1 } { i32 1, i32 1, i1 }* %tmp2, i32 0
  store { i32, i32, i1 } %tmp, { i32, i32, i1 }* %2
  %malloccall3 = tail call i8* @malloc(i32 ptrtoint (( i32, i32, i1 )* getelementptr (i i32, i32, i1 ), { i32, i32, i1 }* null, i32 1) to i32))
  %foo4 = bitcast i8* %malloccall3 to { i32, i32, i1 }*
  store { i32, i32, i1 } { i32 3, i32 5, i1 true }, { i32, i32, i1 }* %foo4
  %tmp5 = load { i32, i32, i1 } { i32 3, i32 5, i1 true }, { i32, i32, i1 }* %foo4
  %tmp6 = load { i32, i32, i1 }* %x
  %tmp7 = getelementptr { i32, i32, i1 } { i32 1, i32 1, i1 }* %tmp6, i32 1
  store { i32, i32, i1 } %tmp5, { i32, i32, i1 }* %3
  %malloccall7 = tail call i8* @malloc(i32 ptrtoint (( i32, i32, i1 )* getelementptr (i i32, i32, i1 ), { i32, i32, i1 }* null, i32 1) to i32))
  %foo8 = bitcast i8* %malloccall7 to { i32, i32, i1 }*
  store { i32, i32, i1 } { i32 0, i32 7, i1 false }, { i32, i32, i1 }* %foo8
%tmp9 = load { i32, i32, i1 }, { i32, i32, i1 }* %foo8
%tmp10 = load { i32, i32, i1 }, { i32, i32, i1 }* %x
%4 = getelementptr { i32, i32, i1 }, { i32, i32, i1 }* %tmp10, i32 2
store { i32, i32, i1 } %tmp9, { i32, i32, i1 }* %4
%x11 = load { i32, i32, i1 }, { i32, i32, i1 }* %x
%malloccall12 = tail call i8* @malloc(i32 ptrtoint (i1**
getelementptr (i1*, i1** null, i32 1) to i32))
%change_struct_args = bitcast i8* %malloccall12 to { { i32, i32, i1 }*
%arg_0 = insertvalue { { i32, i32, i1 }* } zeroinitializer, { i32, i32, i1 }* %x11, 0
store { { i32, i32, i1 }* } %arg_0, { { i32, i32, i1 }* } %change_struct_args
%change_struct_arg_pointer = bitcast { { i32, i32, i1 }* } %change_struct_args to i8*
%5 = call i32 @change_struct(i8* %change_struct_arg_pointer)
%x13 = load { i32, i32, i1 }, { i32, i32, i1 }* %x
%x = getelementptr { i32, i32, i1 }, { i32, i32, i1 }* %x13, i32 0
%7 = load { i32, i32, i1 }, { i32, i32, i1 }* %6
%x14 = extractvalue { i32, i32, i1 } %7, 0
%printf = call i32 @printf(i8*, ...) @printf(i8* getelementptr inbounds
([4 x i8], [4 x i8]* @fmt, i32 0, i32 0), i32 %x14)
%x15 = load { i32, i32, i1 }, { i32, i32, i1 }* %x
%x = getelementptr { i32, i32, i1 }, { i32, i32, i1 }* %x15, i32 1
%9 = load { i32, i32, i1 }, { i32, i32, i1 }* %8
%y = extractvalue { i32, i32, i1 } %9, 1
%printf16 = call i32 @printf(i8*, ...) @printf(i8* getelementptr inbounds
([4 x i8], [4 x i8]* @fmt, i32 0, i32 0), i32 %y)
%x17 = load { i32, i32, i1 }, { i32, i32, i1 }* %x
%10 = getelementptr { i32, i32, i1 }, { i32, i32, i1 }* %x17, i32 2
%11 = load { i32, i32, i1 }, { i32, i32, i1 }* %10
%z = extractvalue { i32, i32, i1 } %11, 2
%printfbool = call i32 @printfbool(i1 %z)
ret i32 0
}
define i32 @change_struct(i8*) {
entry:
%change_struct_args_ptr = bitcast i8* %0 to { { i32, i32, i1 }* }*
%change_struct_args = load { { i32, i32, i1 }* }, { { i32, i32, i1 }* }* %change_struct_args_ptr
%x = extractvalue { { i32, i32, i1 }* } %change_struct_args, 0
%x1 = alloca { i32, i32, i1 }*
store { i32, i32, i1 },* %x, { i32, i32, i1 } /** %x1
%x2 = load { i32, i32, i1 },* { i32, i32, i1 } /** %x1
%1 = getelementptr { i32, i32, i1 }, { i32, i32, i1 }* %x2, i32 0
%2 = load { i32, i32, i1 }, { i32, i32, i1 }* %1
%x3 = load { i32, i32, i1 },* { i32, i32, i1 } /** %x1
%3 = getelementptr { i32, i32, i1 }, { i32, i32, i1 }* %x3, i32 1
%4 = load { i32, i32, i1 }, { i32, i32, i1 }* %3
%x4 = extractvalue { i32, i32, i1 } %4, 0
%tmp = add i32 %x4, 1
%x5 = insertvalue { i32, i32, i1 }, %2, i32 %tmp, 0
%tmp6 = load { i32, i32, i1 },* { i32, i32, i1 } /** %x1
%5 = getelementptr { i32, i32, i1 }, { i32, i32, i1 }* %tmp6, i32 0
store { i32, i32, i1 },* %x5, { i32, i32, i1 } /** %5
%x7 = load { i32, i32, i1 },* { i32, i32, i1 } /** %x1
%6 = getelementptr { i32, i32, i1 }, { i32, i32, i1 }* %x7, i32 1
%7 = load { i32, i32, i1 }, { i32, i32, i1 }* %6
%x8 = load { i32, i32, i1 },* { i32, i32, i1 } /** %x1
%8 = getelementptr { i32, i32, i1 }, { i32, i32, i1 }* %x8, i32 2
%9 = load { i32, i32, i1 }, { i32, i32, i1 }* %8
%y = extractvalue { i32, i32, i1 } %9, 1
%tmp9 = sub i32 %y, 2
%y10 = insertvalue { i32, i32, i1 }, %7, i32 %tmp9, 1
%tmp11 = load { i32, i32, i1 },* { i32, i32, i1 } /** %x1
%10 = getelementptr { i32, i32, i1 }, { i32, i32, i1 }* %tmp11, i32 1
store { i32, i32, i1 },* %y10, { i32, i32, i1 } /** %10
%x12 = load { i32, i32, i1 },* { i32, i32, i1 } /** %x1
%11 = getelementptr { i32, i32, i1 }, { i32, i32, i1 }* %x12, i32 2
%12 = load { i32, i32, i1 }, { i32, i32, i1 }* %11
%z = insertvalue { i32, i32, i1 }, %12, i1 true, 2
%tmp13 = load { i32, i32, i1 },* { i32, i32, i1 } /** %x1
%13 = getelementptr { i32, i32, i1 }, { i32, i32, i1 }* %tmp13, i32 2
store { i32, i32, i1 },* %z, { i32, i32, i1 } /** %13
ret i32 0
}

declare noalias i8* @malloc(i32)
6.2 Test Suite Design

Individual features were built on branches or on the master branch directly, depending on the anticipated duration, difficulty, and/or unpredictability of said feature. Those that fell in the latter categories were developed on branches to provide for an isolated setting for development and testing. Once we had a feature working, we immediately wrote tests, to understand and verify the behavior of the feature we had just built. As a result, a feature and its associated integration tests, going hand-in-hand, were committed together.

Within our integration tests, we have both success and fail test cases, which cover code both in semant (for typing) and in codegen (for other compilation errors). There are a total of 98 test cases.

6.2.1 Test Automation and Scripts

All tests are in the /test/ folder. Success test cases have the format test-*.rjec, with expected output stored in the corresponding test-*.out. Failure test cases are denoted as fail-*.rjec, with expected Failure in fail-*.err.

Testing automation is based off the MicroC testing suite. The testall.sh script compiles and runs *.rjec files under /tests/*. It compares with expected output (in *.out, *.err), and yields *.diff and actual *.out, *.err files when the test fails/doesn’t match with anticipated output. testall.sh is run automatically when RJEC is built and compiled with make.

6.3 Division of Labor

We wrote tests alongside the implemented features, largely during our group programming sessions. As such, everyone wrote tests and helped write tests. While wrapping up our language, we also worked on augmenting our test suite in a group session.

7 Lessons Learned

7.1 Riya

Working on this project together, and seeing it evolve steadily throughout the semester from where we began (a MicroC-based template) to where we are now (a functioning
language with CSP-style concurrency features) has been truly a rewarding experience. I found it particularly interesting to see concepts and theoretical pieces taught in class translate to our application of them while coding up our compiler in OCaml.

Having had some experience with functional programming (Haskell) from the previous semester, it was really cool to see it in the specific context of building a complete compiler. With this project, I continued to see the benefits of writing such concise and clean, yet clearly impactful code; moreover, I feel that the functional programming "mindset" has left its mark in the way I think about coding in general.

I was definitely more new to some of the (Go-like) concurrency features we ended up implementing in our language, and I think our pair programming sessions and associated discussions were very helpful for me to understand both how we would incorporate these aspects (via linking of open-source C libraries) and what kinds of interesting programs we could support with these features (Producer-Consumer, Mutex implementation, MapReduce).

Finally, I found our method of coordinating work and completing pieces of the compiler to be engaging and supportive - we did most of our coding in said sessions where every member was present, and I believe this method gave us a sense of togetherness and helped us learn from each other continuously, something I really valued during this remote semester.

7.2 Justin

This project was a fascinating hands-on experience in building a compiler. It was extremely rewarding to start from barely anything and slowly combine it together into a fully-fledged language. Go has been one of my favorite languages for a long time now, and to me, this language is a labor of love to better understand the design of this language and how it can be implemented.

This class has also been my first experience with functional programming. It took time to get used to, but programming the compiler with OCaml was a very interesting experience and I have learned a lot doing it this way. The paradigm has affected my thinking across my programming in general, and I understand now why Professor Edwards is such a proponent of it, especially for compiler work.

Our team was well-organized, with regularly scheduled weekly meetings and de-
tailed written notes/plans. Our model of programming everything together worked very well for us, although it may not be the most time efficient, it was very useful for knowing the project and learning everything. I would absolutely recommend this technique of group programming to future teams.

Overall, this has been a very fulfilling learning experience. It’s not often that a class makes me want to stay up all night working on it, but this project has certainly done this to me.

### 7.3 Elaine

It was really amazing to see a project evolve from a very high-level and general idea in the beginning, to a fully functioning and relatively low-level code implementation in the end. Learning about syntactic and semantic analysis during the lecture felt completely different from actually trying to figure out how to apply these concepts in code, including how to devise an unambiguous grammar, as well as how to define succinct types for the AST and SAST that cover the entire set of syntactic and semantic rules.

Building out the features in `semant.ml` and `codegen.ml` was also a fascinating learning experience. After struggling with it for a while, we finally gained enough experience in OCaml, a really eye-opening functional programming language to have learned, to be able to design more intricate infrastructure and mechanisms in the code in order to optimize our codebase. It was really fun to derive from the expected behavior of certain features what their underlying implementation should be, and how they fit into the existing code structure we already had. If nothing else, it definitely was a great opportunity for me personally to reflect on language design conventions in a completely new light.

I am especially grateful for the fact that we also decided to take a stab at concurrency in our features. This motivated us to survey the internet for potentially helpful libraries, and in this process, we learned how to incorporate and interface external libraries and code with our main pipeline. Finally, the process of trying to come up with test cases for all kinds of edge cases was also nerve-racking but exciting, especially when we manage to discover obscure new bugs and resolve them.
7.4 Caroline

This project was an interesting opportunity in order to use functional coding (in OCaml) to apply the theoretical concepts behind language design and implementation (as discussed in class) in a practical setting.

The stark contrast between class material and tribulations of working in an unfamiliar coding scheme certainly made it a challenging but rewarding experience. In the code there were real consequences for poor language design and implementation (ambiguous grammars, etc.), and these errors could be tough to resolve due to the succinct and compact nature of OCaml. This was also a first for me with regards to coding functionally rather than imperatively, let alone at such a low level.

Like a past student has said, I agree that I have never had to think so long before composing code. Premature decisions often resulted in needing to make corrections later in order to accommodate other features and make them work. The question of whether or not we would support one edge case or another based on what has already been written has arisen many a time. This made me have to think harder about what aspects of Go we truly wanted to adhere to and what we would inevitably adapt.

Furthermore, this experience was also an opportunity for me to learn more about concurrency from my peers, since it was a fresh topic for me coming in. Having had limited experience with Go as a language, there was a lot to familiarize myself with. To this end, I really valued our pair-coding sessions since a lot could be thought through and discussed during development. Overall, I would say that this was certainly a valuable learning experience.

8 Acknowledgements

We would like to thank Professor Edwards for his tireless effort to educate us in functional programming and compilers, and for creating MicroC, which formed the initial base for our compiler. We would additionally like to thank Shoo as an exemplary past project which helped us with implementing many of our features, as well as Harmonica as an example of a past project with concurrency.

We would also like to thank Martin Sustrik for his Libmill library, which we used to implement our concurrency constructs. Finally, Go is a wonderful language which served as primary inspiration for our language, for which we offer our gratitude.

Riya Chakraborty, Justin Chen, Yuanyuting Wang, Caroline Hoang
Talks by its creators, such as Rob Pike’s “Concurrency is Not Parallelism,” were instrumental in realizing our language’s philosophy and focus on enabling correct, clean, concurrent code.

9 Appendix

9.1 rjec.ml

```ml
(* Top-level of the RJEC compiler: scan & parse the input, check the resulting AST and generate an SAST from it, generate LLVM IR, and dump the module
* Initially based on MicroC
* Written by Elaine Wang, Justin Chen, Riya Chakraborty, and Caroline Hoang
*)

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: ./rjec.native [-a|-s|-l|-c] [file.rjec]"
  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 = Rjecparse.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
```

Riya Chakraborty, Justin Chen, Yuanyuting Wang, Caroline Hoang
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)

9.2 scanner.mll

(* Ocamllex scanner for RJEC
 * Initially based on MicroC, with inspiration from Shoo
 * Written largely by Justin Chen in collaboration with Riya, Elaine
   , Caroline
 *)

{ open Rjecparse }

let digit = ['0' - '9']
let digits = digit+

rule token = parse
    [' ' '	' '' '
'] { token lexbuf } (* Whitespace *)
| '/' /* { comment lexbuf } */ (* Comments *)
| '(' { LPAREN } |
| ')' { RPAREN } |
| '{' { LBRACE } |
| '}' { RBRACE } |
| '[' { LSQUARE } |
| ']' { RSQUARE } |
| ':' { COLON } |
| '.' { DOT } |
| ';' { SEMI } |
| ',' { COMMA } |
| '+' { PLUS } |
| '-' { MINUS } |
| '*' { TIMES } |
| '/' { DIVIDE } |
| '%' { MOD } |
| '=' { ASSIGN } |
| ':=' { INIT } |
| '==' { EQ } |
| '<' { LT } |
9.3 ast.ml

Riya Chakraborty, Justin Chen, Yuanyuting Wang, Caroline Hoang
Abstract Syntax Tree and functions for printing it
* Initially based on MicroC, with inspiration from Shoo
* Written by Justin Chen, Elaine Wang, Riya Chakraborty, and
  Caroline Hoang
*)

```
type op = Add | Sub | Mult | Div | Mod | Equal | Less | Leq | And | Or

type uop = Neg | Not

type typ = Int | Bool | Char
  | Chan of typ
  | Array of typ
  | Struct of string

type bind = typ * string

type expr =
  | IntLit of int
  | StrLit of string
  | CharLit of int
  | BoolLit of bool
  | ArrLit of typ * expr list
  | StructLit of string * (string * expr) list
  | Id of string
  | Binop of expr * op * expr
  | Unop of uop * expr
  | Call of string * expr list
  | Access of expr * string
  | Subscript of string * expr
  | Send of expr * expr
  | Recv of expr
  | Make of typ * (expr option)
  | Close of expr
  | Noexpr

type vdecl_typ = Int | Bool | Char
  | Chan of typ
  | ArrayInit of expr * typ
  | Struct of string

type vdecl = vdecl_typ * string list
```
type assign_stmt =
    DeclAssign of vdecl * expr list
| Assign of expr list * expr list
| Init of expr list * expr list

type stmt =
    Block of stmt list
| Expr of expr
| VdeclStmt of vdecl
| AssignStmt of assign_stmt
| Return of expr list
| If of expr * stmt * stmt
| For of (assign_stmt option) * expr * (assign_stmt option) * stmt
| While of expr * stmt
| Select of (stmt * stmt list) list
| Defer of expr
| Yeet of expr

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

type sdecl = string * bind list

type program = vdecl list * (func_decl list * sdecl list)

(* Pretty-printing functions *)

let string_of_op = function
    Add   -> "+
| Sub   -> "-"
| Mult  -> "*
| Div   -> "/"
| Mod   -> "%"
| Equal -> "=="
| Less  -> "<"
| Leq   -> "<="
| And   -> "&&"
| Or    -> "||"

let string_of_uop = function
    Neg   -> "-"
let rec string_of_typ = function
  | Array(t) -> "[]" ^ string_of_typ t
  | Int -> "int"
  | Bool -> "bool"
  | Char -> "char"
  | Struct(t) -> t
  | Chan(t) -> "chan " ^ string_of_typ t

let rec string_of_expr = function
  | IntLit(l) -> string_of_int l
  | StrLit(l) -> l
  | CharLit(l) -> String.make 1 (Char.chr l)
  | BoolLit(true) -> "true"
  | BoolLit(false) -> "false"
  | ArrLit(t, el) -> "[]" ^ string_of_typ t
                   ^ "{" ^ String.concat "", " (List.map string_of_expr el) ^ "})"
  | StructLit(n, m) -> "struct " ^ n ^ "{" ^ String.Concat "", "
                   (List.map (fun (n, v) -> (n ^ ": " ^ string_of_expr v)) m) ^ 
                   "})"
  | Id(s) -> s
  | Binop(e1, o, e2) -> string_of_expr e1 ^ " " ^ string_of_op o ^ " " ^
                       string_of_expr e2
  | Unop(o, e) -> string_of_uop o ^ string_of_expr e
  | Call(f, el) -> f ^ "(" ^ String.Concat "", " (List.map string_of_expr el) ^ "")
    "
  | Access(s, m) -> (string_of_expr s) ^ "." ^ m
  | Subscript(a, i) -> a ^ "[" ^ string_of_expr i ^ "]"
  | Send(c, e) -> string_of_expr c ^ "" ^ string_of_expr e
  | Recv(c) -> "<- " ^ string_of_expr c
  | Make(t, e) -> "make " ^ "(" ^ "chan " ^ string_of_typ t ^
                 (match e with None -> "" | Some(ex) -> "," ^ string_of_expr ex
                   ) ^ ")"
  | Close(c) -> "close " ^ "(" ^ string_of_expr c ^ ")"
  | Noexpr -> ""

let rec string_of_stmt = function
  | Block(stmts) -> "{" ^ String.Concat "", " (List.map string_of_stmt stmts) ^ 
                   "})\n    "
  | Expr(expr) -> string_of_expr expr ^ ";\n    
  | AssignStmt(s) -> let string_of_assign = function

9.4 rjecparse.mly

/* Ocamlyacc parser for RJEC
   * Initially based on MicroC, with inspiration from Shoo

Riya Chakraborty, Justin Chen, Yuanyuting Wang, Caroline Hoang 86
/* Written by Justin Chen, Elaine Wang, Riya Chakraborty, and Caroline Hoang */
%
open Ast
%
%token SEMI LPAREN RPAREN LBRACE RBRACE LSQUARE RSQUARE DOT COMMA COLON
%token PLUS MINUS TIMES DIVIDE MOD ASSIGN INIT
%token NOT EQ LT LEQ AND OR
%token RETURN IF ELSE FOR WHILE INT BOOL CHAR CHAN STRUCT
%token ARROW DEFER SELECT CASE
%token <int> ILIT CLIT
%token <bool> BLIT
%token <string> SLIT ID
%token EOF

%start program
%type <Ast.program> program

%nonassoc NOELSE
%nonassoc ELSE
%right ASSIGN INIT
%left OR
%left AND
%left EQ
%left LT LEQ
%left PLUS MINUS
%left TIMES DIVIDE MOD
%right ARROW
%right NOT
%left DOT

%
program:
  decls EOF { $1 }

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

fdecl:
  FUNC ID LPAREN formals_opt RPAREN return_types LBRACE stmt_list RBRACE
  { { fname = $2;
  formals = List.rev $4;
  types = List.rev $6;
  body = List.rev $8 } }

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

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

return_types:
  /* nothing */ { [] } |
  typ { [($1] }

typ:
  INT { Int } |
  BOOL { Bool } |
  CHAR { Char } |
  CHAN basic_typ { Chan($2) } |
  LSQUARE RSQUARE non_arr_typ { Array($3) } |
  STRUCT ID { Struct($2) }

basic_typ:
  INT { Int } |
  BOOL { Bool } |
  CHAR { Char } |

non_arr_typ:
  INT { Int } |
  BOOL { Bool } |
  CHAR { Char } |
  CHAN basic_typ { Chan($2) } |
  STRUCT ID { Struct($2) }

vdecl_typ:
  INT { Int }
vdecl:
   VAR id_list vdecl_typ { ($3, List.rev $2) }

id_list:
   ID { [$1] }
   | id_list COMMA ID { $3 :: $1 }

sdecl:
   STRUCT ID LBRACE member_list RBRACE { ($2, $4) }

member_list:
   ID basic_typ SEMI { [($2,$1)] }
   | member_list ID basic_typ SEMI { ($3,$2) :: $1 }

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

stmt:
   expr SEMI { Expr $1 }
   | vdecl SEMI { VdeclStmt $1 }
   | assign_stmt SEMI { AssignStmt $1 }
   | RETURN args_opt SEMI { Return $2 }
   | LBRACE stmt_list RBRACE { Block(List.rev $2) }
   | IF expr LBRACE stmt_list RBRACE else_opt { If($2, Block(List.rev $4), $6) }
   | FOR assign_stmt_opt SEMI expr SEMI assign_stmt_opt LBRACE stmt_list RBRACE { For($2, $4, $6, Block(List.rev $8) ) }
   | FOR expr LBRACE stmt_list RBRACE { While($2, Block(List.rev $4) ) }
FOR LBRACE stmt_list RBRACE
  { While(BoolLit(true), Block(List.rev $3)) }

SELECT LBRACE case_list RBRACE
  { Select(List.rev $3) }

DEFER expr SEMI
  { Defer($2) }

YEET expr SEMI
  { Yeet($2) }

else_opt:
  %prec NOELSE
  { Block([]) }

ELSE IF expr LBRACE stmt_list RBRACE else_opt
  { If($3, Block(List.rev $5), $7) }

ELSE LBRACE stmt_list RBRACE
  { Block(List.rev $3) }

case_list:
  CASE case_stmt COLON stmt_list
    { ($2, List.rev $4) :: $1 }

case_stmt:
  id_subscript ARROW expr
    { Expr(Send($1, $3)) }
  ARROW id_subscript
    { Expr(Recv$2) }
  assign_stmt
    { AssignStmt $1 }

assign_stmt:
  vdecl ASSIGN args_list
    { DeclAssign($1, List.rev $3) }
  args_list ASSIGN args_list
    { Assign(List.rev $1, List.rev $3) }
  args_list INIT args_list
    { Init(List.rev $1, List.rev $3) }

assign_stmt_opt:
  /* nothing */
  { None }
  assign_stmt
    { Some($1) }

expr:
  ILIT
    { IntLit($1) }
9.5  sast.ml

(* Semantically-checked Abstract Syntax Tree and functions for printing it
 * Initially based on MicroC, with inspiration from Shoo
 * Written by Elaine Wang, Justin Chen, Riya Chakraborty, and
   Caroline Hoang
 *)
open Ast
type sexpr = typ * sx
  and sx =
    SIntLit of int
  | SStrLit of string
  | SCharLit of int
  | SBoolLit of bool
  | SStructLit of string * (string * sexpr) list
  | SArrLit of typ * sexpr list
  | SID of string
  | SBinop of sexpr * op * sexpr
  | SUnop of uop * sexpr
  | SCall of string * sexpr list
  | SAccess of sx * string * string
  | SMake of typ * sexpr
  | SSend of sexpr * sexpr
  | SRecv of sexpr * typ
  | SClose of sexpr * typ
  | SSubscript of string * sexpr
  | SNoexpr

type svdecl_typ = SInt | SBool | SChar
  | SArrayInit of svdecl_typ * sexpr
  | SChan of svdecl_typ
  | SStruct of string

type sassign_stmt =
  | SAssign of (sexpr * sexpr) list
  | SDeclAssign of ((string * svdecl_typ) list) * ((string * sexpr) list)
  | SInit of sassign_stmt list

type sstmt =
  | SBlock of sstmt list
  | SExpr of sexpr
  | SAssignStmt of sassign_stmt
  | SReturn of sexpr list
  | SIf of sexpr * sstmt * sstmt
  | SFor of sstmt * sexpr * sstmt * sstmt
  | SWhile of sexpr * sstmt
  | SVdeclStmt of (string * svdecl_typ) list
  | SDefer of sexpr
  | SYeet of sx
  | SSelect of (sx * sstmt) list
type sfunc_decl = {
  stypes : typ list;
  sfname : string;
  sformals : 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 
    | SIntLit(l) -> string_of_int l 
    | SStrLit(l) -> l 
    | SCharLit(l) -> String.make 1 (Char.chr l) 
    | SBoolLit(true) -> "true" 
    | SBoolLit(false) -> "false" 
    | 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 
    | SCall(f, el) -> 
      f ^ "(" ^ String.concat ", " (List.map string_of_sexpr el) ^ 
      ")" 
    | SNoexpr -> "" 
    | _ -> "general sexpr string" 
  ) ^ ")" 

let rec string_of_sstmt = function 
  SBlock(stmts) -> 
    "{
      " ^ String.concat "" (List.map string_of_sstmt stmts) ^ 
    "}"
  | SExpr(expr) -> string_of_sexpr expr ^ ";
  | SReturn(exprs) -> "return " ^ (String.concat ", " (List.map string_of_sexpr exprs)) ^ ";
  | 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_sstmt e1 ^ ") ; " ^ string_of.sexpr e2 ^ 
    " ; " ^ 
    string_of_sstmt e3 ^ ") " ^ string_of_sstmt s
| SWhile(e, s) -> "while (" ^ string_of_sexp e ^ ") " ^ string_of_sstmt s 
| _ -> "general stmt string"

let string_of sfdecl fdecl = 
    fdecl.sfname ^ "(" ^ String.concat ", " (List.map snd fdecl.
        sformals) ^ 
    ")" ^ String.concat ", " (List.map string_of_typ fdecl.stypes) ^ 
    ")\n{\n" ^ String.concat "" (List.map string_of_sstmt fdecl.sbody) ^ 
    "}\n"

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

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

---

9.6 semant.ml

(* Semantic checking for the RJEC compiler
 * Initially based on MicroC, with inspiration from Shoo
 * Written by Elaine Wang, Justin Chen, Riya Chakraborty, and
   Caroline Hoang
 *)

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, structs)) =

    let vdecl_typ_to_typ : vdecl_typ -> typ = function
        Int -> Int
    | Bool -> Bool
    | Char -> Char
    | Chan(t) -> Chan(t)
| ArrayInit(_, t) -> Array(t)  
| Struct(s) -> Struct(s) 

let typ_to_vdecl_typ : typ -> vdecl_typ = function  
    Int -> Int  
    Bool -> Bool  
    Char -> Char  
    Chan(t) -> Chan(t)  
    Array(_) -> raise (Failure("array not implemented"))  
    Struct(s) -> Struct(s) 

let default_vals_in_sexpr : typ -> typ * sx = function  
    Int -> (Int, SIntLit 0)  
    Bool -> (Bool, SBoolLit false)  
    Char -> (Char, SCharLit 0)  
    _ -> raise(Failure("composite types have to be checked separately for their default values")) 

let flatten_global global = List.map  
    (fun name -> (vdecl_typ_to_typ (fst global), name)) (snd global) 

let add_global map vd =  
    let dup_err = "duplicate global " ^ snd vd  
    and make_err er = raise (Failure er)  
    and n = snd vd  
    and typ = fst vd  
    in match n with  
    | _ when StringMap.mem n map -> make_err dup_err  
    | _ -> StringMap.add n typ map 

let check_global map global =  
    List.fold_left add_global map (flatten_global global) 

let global_decls = List.fold_left check_global StringMap.empty globals 

let check_binds (kind : string) (binds : bind list) =  
    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
(* Add structs to symbol table *)
let add_struct smap sd = let dup_err = "duplicate struct " ^ (fst sd) and make_err er = raise (Failure er) and n = fst sd in match n with (* No duplicate globals *) |
| _ when StringMap.mem n smap -> make_err dup_err |
| _ -> check_binds ("member in definition of struct " ^ (fst sd) ^ ":") (snd sd); let members = List.fold_left (fun m (t, n) -> StringMap.add n t m) StringMap.empty (snd sd) in StringMap.add n members smap |
in
let struct_decls = List.fold_left add_struct StringMap.empty structs |
in
(*** Check functions ***)
(* Collect function declarations for built-in functions: no bodies *)
let built_in_decls = let add_bind map (name, ty) = StringMap.add name { types = []; fname = name; formals = [(ty, "x")]; body = [ ] } map in let print_decls = List.fold_left add_bind StringMap.empty [ ("printi", Int); ("printb", Bool); ("printc", Char); ("prints", Array(Char)) ] in StringMap.add "time" { types = [Int]; fname = "time"; formals = [ ]; body = [ ] } print_decls
in

(∗ Add function name to symbol table ∗)
let add_func map fd =
    let built_in_err = "built-in function " ^ fd.fname ^ " is already 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_struct s =
    try StringMap.find s struct_decls
    with Not_found -> raise (Failure ("unrecognized struct " ^ s))
    in

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

let check_function func =
    (∗ Make sure no formals or locals are duplicates ∗)
    check_binds "formal" func.formals;

    (∗ 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
let add_to_scope (v_type : typ) (v_name : string) (scope : typ StringMap.t list) =
  let map = List.hd scope in
  try
    match (StringMap.find v_name map) with
      _ -> raise (Failure ("local variable " ^ v_name ^ " has already been declared"))
  with Not_found ->
    let newMap = StringMap.add v_name v_type map in
    newMap::List.tl scope
  in

let create_scope list =
  let rec helper m = function
    [] -> m
  | (t, n)::tl ->
    let new_m = StringMap.add n t m in
    helper new_m tl
  in helper StringMap.empty list
in

let scope = [ create_scope func.formals; global_decls ]

let type_of_identifier (v_name : string) (scope : typ StringMap.t list) =
  try
    StringMap.find v_name (List.hd scope)
  with Not_found ->
    match List.tl scope with
      [] -> raise (Failure("undeclared reference " ^ v_name))
    | tail -> type_of_identifier v_name tail
  in

let check_chan : typ -> typ = function
  Chan(t) -> t
  | _ -> raise (Failure("tried to perform channel operation through a non-channel variable"))
in

let rec vdecl_to_svdecl_typ (t: vdecl_typ) (scope: _) :
  svdecl_typ = match t with
  Int -> SInt
  | Bool -> SBool
Char -> SChar
Chan(t) -> SChan(vdecl_to_svdecl_typ (typ_to_vdecl_typ t) scope)
Struct(s) -> SStruct(s)
ArrayInit(e, t) ->
  let (t', e') = expr scope e in
  if t' <> Int then raise(Failure("array size can only be integer expressions"));
  SArrayInit(vdecl_to_svdecl_typ (typ_to_vdecl_typ t) scope, (t', e'))

and
(* Return a semantically-checked expression, i.e., with a type *)
expr (scope : typ StringMap.t list) (e : expr) : sexpr = match e with
  IntLit l -> (Int, SIntLit l)
  StrLit l -> (Array(Char), SStrLit l)
  CharLit l -> (Char, SCharLit l)
  BoolLit l -> (Bool, SBoolLit l)
  ArrLit(t, el) ->
    let check_elem e =
      let (t', e') = expr scope e in
      if t' <> t then raise(Failure("array element doesn't match declared array type"));
      (t', e')
in
    let selems = List.map check_elem el in
    (Array(t), SArrLit(t, selems))
  StructLit(sn, ml) ->
    let smembers = find_struct sn in
    let check_member sname members vm (n, e) =
      let (t, e') = expr scope e in
      match n with
      _ when ((StringMap.mem n members) && (t = StringMap.find n members)) -> StringMap.add n (t, e') vm
      _ -> raise(Failure("unknown member " ^ n ^ " of type " ^ (string_of_typ t) ^ " in definition of struct " ^ sname )) in
    let member_vals = List.fold_left (check_member sn smembers) StringMap.empty ml in
    let full_member_vals = List.fold_left (fun m (n, t) -> match n with
      _ when StringMap.mem n m -> m
      _ -> StringMap.add n (default_vals_in_sexpr t) m
    ) StringMap.empty member_vals in
    Struct(s, full_member_vals)
member_vals (StringMap.bindings smembers) in
(Struct(sn), SStructLit(sn, StringMap.bindings
full_member_vals))
| Id s   -> (type_of_identifier s scope, SId s)
| Unop(op, e) as ex ->
  let (t, e') = expr scope e in
  let ty = match op with
    Neg when t = Int -> 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 scope e1
  and (t2, e2') = expr scope 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 : typ = match op with
    Add | Sub | Mult | Div | Mod when same && t1 = Int ->
     Int
  | Equal when same && (t1 = Int || t1 = Char || t1 =
      Bool) -> Bool
  | Less | Leq when same && (t1 = Int || t1 = Char) -> 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')))
| Make(t, buf_raw) ->
  let check_buf (buf_raw : expr option) : sexpr = match
    buf_raw with
    None -> (Int, SIntLit 0)
  | Some(e) -> let (t', e') = expr scope e in
    if t' <> Int then raise(Failure("non−
      integer used for channel buffer size"))
    else (t', e') in
let buf = check_buf buf_raw in
(Chan(t), SMake(t, buf))

| Send(n, e) ->
  let (ct, ce) = expr scope n in
  let chan_type = check_chan ct in
  let (t', e') = expr scope e in
  if t' <> chan_type then raise(Failure("Channel type mismatch
  with type of element to send"));
  (t', SSend((ct, ce), (t', e')))

| Recv n ->
  let (ct, ce) = expr scope n in
  let chan_type = check_chan ct in
  (chan_type, SRecv((ct, ce), chan_type))

| Close n ->
  let (ct, ce) = expr scope n in
  let chan_type = check_chan ct in
  (chan_type, SClose((ct, ce), chan_type))

| Access(e, mn) ->
  let e' = expr scope e in
  let extract_struct_name (t : typ) = match t with
    Struct(n) -> n
  | _ -> raise(Failure("Invalid syntax: access member field
    of non-struct object\n")) in
  let check_struct_or_arr_field e' = match (snd e') with
    SId(n) -> extract_struct_name (type_of_identifier n
    scope)
  | SSubscript(an, _) -> (function
    Array(Struct(n)) -> n
  | _ -> raise(Failure("subscript on non-struct array
    element!"))) in
  let sn = check_struct_or_arr_field e' in
  let smembers = find_struct sn in
  let t = match sn with
    _ when StringMap.mem mn smembers -> StringMap.find mn
    smembers
  | _ -> raise(Failure("unknown field " ^ mn ^ " in struct " ^
    sn)) in
  (t, SAccess((snd e'), sn, mn))

| Subscript(s, e) ->
  let arr = expr scope (Id s) in
  let array_t = (function
    Array(t) -> t
let index = expr scope e in
if (fst index) <> Int then raise (Failure("subscript with non-
integer expression!"));
(array_t, SSubscript(s, index))
| 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 scope 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
  (* fix multiple returns later *)
  let rt_type : typ = match fd.types with
    [] -> Int
    | t :: [] -> t
    | _ -> raise(Failure("Multiple return types not-
      implemented yet")) in
  (rt_type, SCall(fname, args'))
| _ -> raise (Failure("not yet implemented"))

let check_assign_var scope var e =
  let lt = type_of_identifier var scope and (rt, e') = expr
    scope e in
  let err = "illegal assignment " ^ string_of_typ lt ^ " = " ^
    string_of_typ rt
  in ignore(check_assign lt rt err); (var, (rt, e'))

let check_bool_expr scope e =
  let (t', e') = expr scope e
  and err = "expected Boolean expression in " ^ string_of_expr e
  in if t' != Bool then raise (Failure err) else (t', e')
let rec check_stmt scope = function
  Expr e -> (SExpr(expr scope e), scope)
| If(p, b1, b2) ->
    let (sstmt1, _) = check_stmt scope b1
    and (sstmt2, _) = check_stmt scope b2 in
    (SIf(check_bool_expr scope p, sstmt1, sstmt2), scope)
| For(e1, e2, e3, st) ->
    let rec forbid_defer = function
      Defer _ -> raise(Failure("defer statement inside of
                   while block"))
      | Block sl -> List.iter forbid_defer sl
      | _ -> () in
    forbid_defer st;

    let (sstmt, nscope) =
      (function
        None    -> (SExpr(Int, SNoexpr), scope)
        | Some(s) -> check_stmt scope (AssignStmt s)
      ) el in

    let sexpr = check_bool_expr nscope e2 in

    let (sstmt3, nscope) = (fun scope e ->
                              match e with
                                Some(Assign(_, _) as e') ->
                                check_stmt scope (AssignStmt e')
                              | None                -> (SExpr(Int, SNoexpr), scope)
                              | _                   -> raise(Failure("variable declaration
                                      misplaced in for loop"))
                        ) nscope e3 in

    let (sstmt4, _) = check_stmt nscope st in

    (SFor(sstmt, sexpr, sstmt3, sstmt4), scope)
| While(e, s) ->
    let rec forbid_defer = function
      Defer _ -> raise(Failure("defer statement inside of
                   while block"))
      | Block sl -> List.iter forbid_defer sl
      | _ -> () in
    forbid_defer s;

    let (sstmt, nscope) = check_stmt scope s in

    (SWhile(check_bool_expr nscope e, sstmt), nscope)
| Return el ->
if (List.length el <> List.length func.types) then
    raise(Failure("The function " ^ func.fname ^ ", has " ^
     string_of_int (List.length func.types) ^
     " return types, but only " ^ string_of_int (List.length el) ^
     " expressions were returned"));

let check_return_typ e rt =
    let (t, e') = expr scope e in
    if t = rt then (t, e')
    else raise (Failure("return gives " ^ string_of_typ t ^ 
    expected " ^
    string_of_typ rt ^ " in " ^ string_of_expr e))
in
(SReturn(List.map2 check_return_typ el func.types), scope)

(* A block is correct if each statement is correct and
nothing
follows any Return statement. Nested blocks are
flattened. *)

| Yeet e ->
    let scall = (function
        Call(f, _) as call ->
            let fdecl = find_func f in
            if (fdecl.types <> []) && (fdecl.types <> [Int])
            then raise(Failure("a yeet function call can only
            return int or nothing"));
            snd (expr scope call)
        | _ -> raise(Failure("can't yeet a non-function call"))
    ) e In (SYeet(scall), scope)
| AssignStmt s ->
    let check_assign_stmt = function
        Assign(vl, el) ->
            let helper v e =
                let (mt, e') = expr scope v in
                (function
                    SId s ->
                        let (_, e') = check_assign_var scope s e in
                        ((fst e', SId(s)), e')
                    | SAccess (n, sn, mn) ->
                        let (t, e') = expr scope e in
                        if mt <> t then raise(Failure("illegal
                        assignment " ^ string_of_typ mt ^ " = " ^
                        string_of_typ t)); ((mt, SAccess(n, sn, mn))
                        , (t, e'))
                    | SSubscript (an, index) ->
                        let mt = type_of_identifier an scope in
                        ...

Riya Chakraborty, Justin Chen, Yuanyuting Wang, Caroline Hoang
let (t, e') = expr scope e in
if mt <> Array(t) then raise(Failure("illegal assignment of element to array"));
((mt, SSubscribe(an, index)), (t, e'))
| _ → raise(Failure("invalid assignment"))
) e'

in
(SAssign (List.map2 helper vl el), scope)

| DeclAssign(vd, el) -> let (_, nscope) = check_stmt scope (VdeclStmt vd) in
  let vdl = List.map (fun n -> (n, vdecl_to_svdecl_typ (fst vd) scope) ) (snd vd) in
  let assl = List.map2 (check_assign_var nscope) (snd vd) el in
  (SDeclAssign(vdl, assl), nscope)
| Init(vl, el) ->
  let helper (ll, scope) v e =
    let s = (function
      | Id s → s
      | _ → raise(Failure("trying to initialize non-identifier"))
    ) v in

  let (t, e') = expr scope e in
  let nscope = add_to_scope t s scope in
  let lhs_svdecl = (function
    (_ , SArrLit(elem_t, el)) →
      let arr_len = List.length el in
      SArrayInit(  
        vdecl_to_svdecl_typ (typ_to_vdecl_typ elem_t) scope,  
        (Int, SIntLit(arr_len))  
      )
    | (,, SStrLit l) → SArrayInit(SChar, (Int, SIntLit((String.length l) + 1)))
    | (Array(elem_t, _), _) → SArrayInit(  
        vdecl_to_svdecl_typ (typ_to_vdecl_typ elem_t) scope,  
        (Int, SIntLit(1))
    )
    | _ → vdecl_to_svdecl_typ (typ_to_vdecl_typ t ) scope
  ) (t, e')
in
(SDeclAssign([[(s, lhs_svdecl)], [(s, (t, e'))]]))

in
let (dal, nscope) = List.fold_left2 helper ([], scope) vl el
in (SInit(List.rev dal), nscope)

in let (sassign, sscope) = check_assign_stmt s in
(SAssignStmt(sassign), sscope)
| VdeclStmt s ->
let (t, nl) = s in
let (nscope, vdecls) = List.fold_left (fun (scope, vdecls) n ->
    (add_to_scope (vdecl_typ_to_typ t) n scope, (n, vdecl_to_svdecl_typ t scope)::vdecls)
) (scope, []) nl
in (SVdeclStmt(vdecls), nscope)
| Defer e ->
(function
    Call(_, _) -> (SDefer(expr scope e), scope)
| _ -> raise(Failure("deferring a non-function call"))
) e
| Select cl ->
let rec check_case_list scope = function
    (case, sl) :: tl ->
let case_instr = (function
    Expr(Send(c, e)) -> SSend(expr scope c, expr scope e)
| Expr(Recv(c)) | AssignStmt(DeclAssign(_, [Recv(c)]))
| AssignStmt(Assign(_, [Recv(c)]))
| AssignStmt(Init(_, [Recv(c)]))
let (ct, ce) = expr scope c in
let chan_type = check_chan ct in
SRecv((ct, ce), chan_type)
| _ -> raise(Failure("wrong case format in semant"))
) case in
let blockstmt = Block(case :: sl) in
let (ret, _) = check_stmt scope blockstmt in
let ret2 = check_case_list scope tl in
((case_instr, ret) :: ret2)
| [] -> []
in
(SSelect(check_case_list scope cl), scope)
let bscope = (create_scope []) :: scope in
let rec check_stmt_list scope = function
  | Return _ as s ->
    let (ret, _) = check_stmt scope s in (ret, scope)
  | Return _ :: _ -> raise (Failure "nothing may follow a return")
  | s :: ss ->
    let (ret, nscope) = check_stmt scope s in
    let (ret2, nscope2) = check_stmt_list nscope ss in
    (ret :: ret2, nscope2)
  | [] -> ([], scope)
in let (bret, _) = check_stmt_list bscope sl
in (SBlock(bret), scope)


9.7 codegen.ml

(* Code generation: translate takes a semantically checked AST and
   Initially based on MicroC, with inspiration from Shoo
   Written by Elaine Wang, Justin Chen, Riya Chakraborty, and
   Caroline Hoang
*)

module L = Llvm
module A = Ast
open Sast

module StringMap = Map.Make(String)

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

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

(* Get types from the context *)
let i64_t = L.i64_type context
and i32_t = L.i32_type context
and i8_t = L.i8_type context
and i1_t = L.i1_type context
and void_t = L.void_type context
and void_ptr_t = L.pointer_type (L.i8_type context) in
let func_t = L.pointer_type (L.function_type i32_t [|
  void_ptr_t |]) in

let rec generate_seq n = if n >= 0 then (n :: (generate_seq (n-1))
  ) else [] in

let struct_decls = ((A.typ StringMap.t) * L.lltype) StringMap.t =
let add_struct m (n, ml) =
  let ltype_of_basic_typ ((t, _) : (A.typ * string)) : L.lltype
  = match t with
    A.Int -> i32_t
  | A.Bool -> i1_t
  | A.Char -> i8_t
  | _ -> raise(Failure("Struct member typ not basic --
    should've been checked in parser!\n"))
in
let compare_by (_, n1) (_, n2) = compare n1 n2 in
let member_typs = Array.of_list (List.map ltype_of_basic_typ (List.sort compare_by ml)) in
let struct_t = L.struct_type context member_typs in
let member_names = List.fold_left (fun m (t, n) -> StringMap.
  add n t m) StringMap.empty ml in
StringMap.add n (member_names, struct_t) m
in List.fold_left add_struct StringMap.empty structs in

(* Return the LLVM type for a RJEC type *)
let rec ltype_of_typ : A.typ -> L.lltype = function
  A.Int -> i32_t
| A.Bool -> i1_t
| A.Char -> i8_t
| A.Struct(n) -> snd (StringMap.find n struct_decls)
| A.Chan(_) -> void_ptr_t  
| A.Array(t) -> L.pointer_type (ltype_of_typ t) 

let typ_to_typ_char (t : A.typ) = match t with 
  | A.Int   -> L.const_int i8_t (Char.code 'i') 
  | A.Bool  -> L.const_int i8_t (Char.code 'b') 
  | A.Char  -> L.const_int i8_t (Char.code 'c') 
  | _       -> raise(Failure("non-basic type for chan; should've 
                             checked in parser")) 

(* 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 
    _ -> L.const_int (ltype_of_typ t) 0 
    (* TODO: add other cases *) 
    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 gettime_t : L.lltype = 
  L.function_type i32_t [ | ] in 
let gettime_func : L.llvalue = 
  L.declare_function "get_time" gettime_t the_module in 

let printbool_t : L.lltype = 
  L.function_type i32_t [ | i1_t |] in 
let printbool_func : L.llvalue = 
  L.declare_function "printbool" printbool_t the_module in 

let yeet_t : L.lltype = 
  L.function_type void_t [ | func_t ; void_ptr_t |] in 
let yeet_func : L.llvalue = 
  L.declare_function "yeet" yeet_t the_module in 

let makechan_t : L.lltype = 
  L.function_type void_ptr_t [ | i8_t ; i32_t |] in 
let makechan_func : L.llvalue = 
  L.declare_function "makechan" makechan_t the_module in
let send_t : L.lltype = 
    L.function_type void_t [| void_ptr_t ; i8_t ; i32_t |] in
let send_func : L.llvalue = 
    L.declare_function "send" send_t the_module in

let recv_int_t : L.lltype = 
    L.function_type i32_t [| void_ptr_t |] in
let recv_int_func : L.llvalue = 
    L.declare_function "recv_int" recv_int_t the_module in

let recv_bool_t : L.lltype = 
    L.function_type i1_t [| void_ptr_t |] in
let recv_bool_func : L.llvalue = 
    L.declare_function "recv_bool" recv_bool_t the_module in

let closechan_t : L.lltype = 
    L.function_type void_t [| void_ptr_t ; i8_t |] in
let closechan_func : L.llvalue = 
    L.declare_function "closechan" closechan_t the_module in

let clause_member_typs = [| i8_t ; void_ptr_t ; void_ptr_t ; i64_t |
    |] in
let clause_t = L.struct_type context clause_member_typs in
let select_t : L.lltype = 
    L.function_type i32_t [| L.pointer_type clause_t ; i32_t |] in
let select_func : L.llvalue = 
    L.declare_function "selectchan" select_t the_module in

let function_arg_structs = 
    let add_function_arg_struct m fdecl = 
        let member_typs = Array.of_list (List.map (fun (t, _) -> 
            ltype_of_typ t) fdecl.sformals) in
        let struct_t = L.struct_type context member_typs in
        StringMap.add fdecl.sfname struct_t m
    in
    List.fold_left add_function_arg_struct StringMap.empty functions
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 in
  (  
    (* TODO: actually handle the multiple return types. *)  
    let rt_type = match fdecl.stypes with  
       []  ⇒ i32_t  
    | t :: []  ⇒ ltype_of_typ t  
    | _  ⇒ raise(Failure("Multiple return types not implemented yet")) in  
    let ftype = L.function_type rt_type [| void_ptr_t |] in  
    StringMap.add name (L.define_function name ftype the_module, fdecl) m  
  )
  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\n" "fmt" builder
    and char_format_str = L.build_global_stringptr "%c\n" "fmt" builder
    and str_format_str = L.build_global_stringptr "%s\n" "fmt" builder in

    (* TODO: reevaluate string formatting *)

    (* Construct the function's "locals": formal arguments and locally
debclared variables. Allocate each on the stack, initialize
their value, if appropriate, and remember their values in the "locals" map *)

    let formal_vals =
      let args_type = StringMap.find fdecl.sfname function_arg_structs in
      let void_ptr = List.hd (Array.to_list (L.params the_function)) in
      let ptr = L.build_bitcast void_ptr (L.pointer_type args_type) (fdecl.sfname ^ "_args_ptr") builder in
      let args = L.build_load ptr (fdecl.sfname ^ "_args") builder in
      let idxs = List.rev (generate_seq ((List.length fdecl.sformals

Riya Chakraborty, Justin Chen, Yuanyuting Wang, Caroline Hoang 112
let args_list = List.fold_left (fun l idx ->
    let arg = L.build_extractvalue args idx (
        fdecl.sfname ^ "_arg_" ^ (string_of_int idx) 
    ) builder in arg::l
) [] idxs in
List.rev args_list

let local_vars =
let add_formal m (t, n) p =
    L.set_value_name n p;
let local =
    let create_local (t: A.typ) = match t with
        A.Struct(_) as t -> L.build_malloc (ltype_of_typ t) n builder
    | _ -> builder in
    create_local t 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
    formal_vals in
List.fold_left add_local formals []
in

(∗ Return the value for a variable or formal argument.
Check local names first, then global names ∗)
let rec lookup n ml = try StringMap.find n (List.hd ml) with
    Not_found -> match List.tl ml with
        [] -> raise (Failure("codegen lookup: undeclared reference
            " ^ n))
    | tail -> lookup n tail
in
let defered_el = [] in

let rec svdecl_typ_to_typ : svdecl_typ -> A.typ = function
  SInt               -> A.Int
| SBool             -> A.Bool
| SChar             -> A.Char
| SChan(t)          -> A.Chan(svdecl_typ_to_typ t)
| SArrayInit(t, _)  -> A.Array(svdecl_typ_to_typ t)
| SStruct(s)        -> A.Struct(s)
in

let default_value_of_typ (t: A.typ) builder = match t with
  A.Int | A.Bool | A.Char -> L.const_int (ltype_of_typ t) 0
| A.Chan(_)        -> L.const_pointer_null void_ptr_t
| A.Struct(n)      ->
  let (member_names, struct_t) = StringMap.find n struct_decls in
  let compare_by (n1, _) (n2, _) = compare n1 n2 in
  let members = List.sort compare_by (StringMap.bindings member_names) in
  let idxs = List.rev (generate_seq ((List.length members) - 1)) in
  let v = List.fold_left2 (fun agg i member ->
    let (n, t) = member in
    (L.build_insertvalue agg (L.const_int (ltype_of_typ t) 0) i n builder))
    (L.const_null struct_t) idxs members in v
| A.Array(t')      -> L.const_pointer_null (L.pointer_type (ltype_of_typ t'))
in

(* Construct code for an expression; return its value *)
let rec expr m builder ((_, e) : sexpr) = match e with
  SIntLit i       -> L.const_int i32_t i
| SStrLit s      -> L.build_global_stringptr s "strlit" builder
| SCharLit c      -> L.const_int i8_t c
| SBoolLit b      -> L.const_int il_t (if b then 1 else 0)
| SStructLit(sn, ml) ->
  let (_, struct_t) = StringMap.find sn struct_decls in
  let compare_by (n1, _) (n2, _) = compare n1 n2 in
  let sorted_vals = List.map (fun (_, sexpr) -> expr m builder sexpr) (List.sort compare_by ml) in
  let idxs = List.rev (generate_seq ((List.length sorted_vals) - 1)) in
  let v = List.fold_left2 (fun agg i v ->
```plaintext
L.build_insertvalue agg v i "tmp" builder)
(L.const_null struct_t) idxs sorted_vals in
let local = L.build_malloc struct_t sn builder in
ignore(L.build_store v local builder); local
| SArrLit(t, el) ->
  let arr_len = List.length el in
  let arr_t = ltype_of_typ t in
  let arr = L.build_array_malloc arr_t (L.const_int i32_t
    arr_len) "arrlit" builder in
  let elems = List.map (expr m builder) el in
  let idxs = List.rev (generate_seq (List.length el - 1)) in
  List.iter2 (fun i elem −>
    let arr_ptr = L.build_gep arr [| (L.const_int i32_t i) |]
      "" builder in
    let elem_ptr = L.build_pointercast arr_ptr (L.pointer_type
      arr_t) "" builder in
    ignore(L.build_store elem elem_ptr builder)
  ) idxs elems; arr
| SNoexpr -> L.const_int i32_t 0
| SID s -> L.build_load (lookup s m) s builder
| SBinop (e1, op, e2) ->
  let e1' = expr m builder e1
  and e2' = expr m 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.Mod  -> L.build_srem
    A.And  -> L.build_and
    A.Or   -> L.build_or
    A.Equal -> L.build_icmp L.Icmp.Eq
    A.Less -> L.build_icmp L.Icmp.Slt
    A.Leq  -> L.build_icmp L.Icmp.Sle
  ) e1' e2' "tmp" builder
| SUnop (op, ((_, _) as e)) ->
  let e' = expr m builder e in
  (match op with
    A.Neg     -> L.build_neg
    A.Not     -> L.build_not) e' "tmp" builder
| SCall ("printi", [e]) ->
L.build_call printf_func [| int_format_str ; (expr m builder
  e) |]
"printf" builder
```

Riya Chakraborty, Justin Chen, Yuanyuting Wang, Caroline Hoang
| SCall ("printc", [e]) -> |
| L.build_call printf_func [| char_format_str ; (expr m builder e) |] |
| "printf" builder |

| SCall ("prints", [e]) -> |
| L.build_call printf_func [| str_format_str ; (expr m builder e) |] |
| "printf" builder |

| SCall ("printb", [e]) -> |
| L.build_call printbool_func [| (expr m builder e) |] |
| "printbool" builder |

| SCall ("time", _) -> |
| L.build_call gettime_func [|] |
| "get_time" builder |

| SCall (f, args) -> |
| let (fdef, local, result) = construct_func_call f args m builder in |
| L.build_call fdef [| local |] result builder |

| SMake (t, buf) -> |
| let t_char = typ_to_typ_char t in |
| let ll_buf = expr m builder buf in |
| L.build_call makechan_func [| t_char ; ll_buf |] "makechan" builder |

| SSend (n, e) -> |
| let chan = expr m builder n in |
| let value = expr m builder e in |
| L.build_call send_func [| chan ; typ_to_typ_char (fst e) ; L.build_intcast value i32_t "tmp" builder |] "" builder |

| SRecv (n, t) -> |
| let chan = expr m builder n in |
| let recv_by_typ (t : A.typ) = match t with |
| | A.Int -> L.build_call recv_int_func [| chan |] "recv_int" builder |
| | A.Bool -> L.build_call recv_bool_func [| chan |] "recv_bool" builder |
| | A.Char -> L.build_call recv_char_func [| chan |] "recv_int" builder |
| | _ -> raise(Failure("trying to receive non-basic type from channel; should've checked in parser")) in |
| recv_by_typ t |
| SClose (n, t) ->
let chan = expr m builder n in
L.build_call closechan_func [| chan ; typ_to_typ_char t |]
"" builder
| SAccess (sx, sn, mn) ->
let struct_val = expr m builder (A.Struct(sn), sx) in
let (member_names, _) = StringMap.find sn struct_decls in
let compare_by (n1, _) (n2, _) = compare n1 n2 in
let sorted_names = List.map (fun (n, _) -> n) (List.sort compare_by (StringMap.bindings member_names)) in
let name_idx_pairs = List.map (fun i n -> (n, i)) sorted_names in
let idx = snd (List.hd (List.filter (fun (n, i) -> n = mn) name_idx_pairs)) in
L.build_extractvalue struct_val idx mn builder
| SSubscript (n, e) ->
let arr = expr m builder (A.Int, SId(n)) in
let idx = expr m builder e in
L.build_load (L.build_gep arr [ | idx | ] "" builder) ""
and
construct_func_call f args m builder =
let (fdef, fdecl) = StringMap.find f function_decls in
let args_t = StringMap.find f function_arg_structs in
let llargs = List.rev (List.map (fun arg -> match arg with
(_, SStructLit(s, _)) -> let p = expr m builder arg in
L.build_load p (s ^ "_lit") builder
| (_, SStrLit l) ->
let arr_len = (String.length l) + 1 in
let arr_t = (ltype_of_typ A.Char) in
let arr = L.build_array_malloc arr_t (L.const_int i32_t arr_len) "arrlit" builder in
let idxs = List.rev (generate_seq (arr_len - 1)) in
List.iter (fun i ->
let c = (if i = (String.length l) then 0 else Char. code (String.get l i)) in
let elem = L.const_int i8_t c in
let arr_ptr = L.build_gep arr [ | (L.const_int i32_t i) | ] "" builder in
let elem_ptr = L.build_pointercast arr_ptr (L. pointer_type arr_t) "" builder in
ignore(L.build_store elem elem_ptr builder)
let local = L.build_malloc args_t (f ^ "_args") builder in
let idxs = List.rev (generate_seq ((List.length fdecl.
sformals) - 1)) in
let args = List.fold_left2 (fun agg i llarg ->
    (L.build_insertvalue agg llarg i ("arg_" ^ string_of_int i)
        builder))
(L.const_null args_t) idxs llargs in
ignore(L.build_store args local builder);
(* TODO: fix multiple return types later *)
let result = (match fdecl.stypes with
    [] -> ""
| _ :: [] -> f ^ "_result"
| _ -> raise(Failure("Multiple return types not implemented yet"))) in

let void_ptr_arg = L.build_bitcast local void_ptr_t (f ^ 
    _arg_pointer") builder in
(fdef, void_ptr_arg, result)
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 m dl builder = function
    SBlock sl ->
let stmt_list builder ml dl sl =
    let helper (bldr, map, dl) = stmt map dl bldr in
    let (b, mm, ndl) = List.fold_left helper (builder, ml, dl) sl
    in (b, List.tl mm, ndl)
in
stmt_list builder (StringMap.empty::m) dl sl

| SVdeclStmt vdl ->
    let declare_var (builder, mm, dl) ((n, t) : (string * svdecl_typ)) =
        let store_default_val = function
        | SInt | SBool | SChar | SChan(_) ->
            let local = L.build_alloca (ltype_of_typ (svdecl_typ_to_typ t)) n builder in
            let default_value = (default_value_of_typ (svdecl_typ_to_typ t) builder) in
            ignore (L.build_store default_value local builder); local
        | SStruct(_) ->
            let local = L.build_malloc (ltype_of_typ (svdecl_typ_to_typ t)) n builder in
            let v = default_value_of_typ (svdecl_typ_to_typ t) builder in
            ignore (L.build_store v local builder); local
        | SArrayInit(t, e) ->
            let array_t = ltype_of_typ (svdecl_typ_to_typ t) in
            let array_len = expr m builder e in
            let ptr = L.build_array_malloc array_t array_len builder in
            let local = L.build_alloca (L.pointer_type array_t) n builder in
            ignore (L.build_store ptr local builder); local in
            let local = store_default_val t in
            (** TODO: add to the symbol table? / manage scope? *)
            let new_m = StringMap.add n local (List.hd mm) in
            (builder, new_m::(List.tl mm), dl)
        in List.fold_left declare_var (builder, m, dl) vdl
| SExpr e -> ignore (expr m builder e); (builder, m, dl)
| SReturn e -> (** TODO: fix multiple return types later *)
    ignore (List.map (fun (fdef, llargs, result) ->
    L.build_call fdef llargs result builder) dl);
ignore (match fdecl.stypes with
| Special "return nothing" instr *)
[] → L.build_ret (L.const_int i32_t 0) builder
    (* Build return statement *)
| _ :: [] → L.build_ret (expr m builder (List.hd e))
    builder
| _ → raise(Failure("Multiple return types not implemented yet"));
    (builder, m, [])
| SYeet(SCall(f, args)) →
    let (fdef, local, _) = construct_func_call f args m builder
    in
    (* let local_void_ptr = L.build_bitcast local void_ptr_t (f
     "_arg_ptr") builder in *)
    ignore(L.build_call yeet_func [| fdef ; local || ""
     builder); (builder, m, dl)
| SAssignStmt s → let assign_stmt builder = function
    SAssign sl →
    let extract_value e = match e with
    (_, SStructLit(_, _)) → let ptr = expr m builder e
    in
    L.build_load ptr "tmp"
    builder
| (_, SStrLit l) →
    let arr_len = (String.length l) + 1 in
    let arr_t = (ltype_of_typ A.Char) in
    let arr = L.build_array_malloc arr_t (L.const_int
    i32_t arr_len) "arrlit" builder
    in
    List.iter (fun i →
      let c = (if i = (String.length l) then 0 else Char
       .code (String.get l i)) in
      let elem = L.const_int i8_t c in
      let arr_ptr = L.build_gep arr [| (L.const_int
       i32_t i) || ""
      builder); (builder
      in
      let elem_ptr = L.build POINTERCAST arr_ptr (L.
       pointer_type arr_t) "" builder); (builder
      in
      ignore(L.build_store elem elem_ptr builder)
    ) idsxs; arr
| _ → expr m builder e
in
let eval_assign_lhs ee e = match ee with
| (_, Sid(s)) →
    let value = extract_value e in
    ignore(L.build_store value (lookup s m) builder); builder
| (_, SSubscript(an, index)) -> |

  let value = extract_value e in
  let arr = L.build_load (lookup an m) "tmp" builder in
  let idx = expr m builder index in
  let ptr = L.build_gep arr [| idx |] "" builder in
  ignore(L.build_store value ptr builder); builder

| (_, SAccess(sx, sn, mn)) -> |

  let struct_val = expr m builder (A.Struct(sn), sx) in
  let (member_names, _) = StringMap.find sn struct_decls in
  let compare_by (n1, _) (n2, _) = compare n1 n2 in
  let sorted_names = List.map (fun (n, _) -> n) (List.sort compare_by (StringMap.bindings member_names) ) in
  let name_idx_pairs = List.mapi (fun i n -> (n, i)) sorted_names in
  let idx = snd (List.hd (List.filter (fun (n, _) -> n = mn) name_idx_pairs)) in
  let e' = expr m builder e in
  let v = L.build_insertvalue struct_val e' idx mn builder in

  let insert_value sx = match sx with
    SId(n) -> L.build_store v (lookup n m) builder
    | SSubscript(an, index) ->
      let arr = L.build_load (lookup an m) "tmp" builder in
      let idx = expr m builder index in
      L.build_store v (L.build_gep arr [| idx |] "" builder) builder
    | _ -> raise(Failure("invalid access; should've checked in semant!"))

  in
  ignore(insert_value sx); builder
  | _ -> raise(Failure("invalid assign lhs; should've checked in semant!"))

  in
  ignore(List.map (fun (ee, e) -> eval_assign_lhs ee e) sl ); (builder, m, dl)

| SDeclAssign (vdl, assl) -> let (_, mm, dl) = stmt m dl builder (SVdeclStmt vdl) in
let new_assl = List.map (fun (s, e) -> ((
svdecl_typ_to_typ (snd (List.hd vdl)), SId(s)), e)) assl in
stmt mm dl builder (SAssignStmt(SAssign(new_assl)))
| SInit dal -> List.fold_left (fun (builder, m, dl) da ->
stmt m dl builder(SAssignStmt da)) (builder, m, dl) dal
in assign_stmt builder

| SIf (predicate, then_stmt, else_stmt) ->
  let bool_val = expr m 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
  let (builder1, _ , dl) = stmt m dl (L.builder_at_end context
  then_bb) then_stmt in
  add_terminal builder1 build_br_merge;

  let else_bb = L.append_block context "else" the_function in
  let (builder2, _, dl) = stmt m dl (L.builder_at_end context
  else_bb) else_stmt in
  add_terminal builder2 build_br_merge;

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

| 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
  let (builder, mm, dl) = stmt m dl (L.builder_at_end context
  body_bb) body in
  add_terminal builder (L.build_br pred_bb);

  let pred_builder = L.builder_at_end context pred_bb in
  let bool_val = expr mm 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, mm, dl)

| SFor (e1, e2, e3, body) -> stmt m dl builder
  ( SBlock [e1 ; SWhile (e2, SBlock [body ; e3]) ] ) |
| SDefer(_, SCall(f, args)) -> |

  let construct_call = function
    "printb" -> (printbool_func, [\ (expr m builder (List.hd args)) []], "printbool")
  | "prints" -> (printf_func, [\ str_format_str ; (expr m builder (List.hd args)) []], "printf")
  | "printc" -> (printf_func, [\ char_format_str ; (expr m builder (List.hd args)) []], "printf")
  | "printi" -> (printf_func, [\ int_format_str ; (expr m builder (List.hd args)) []], "printf")
  | "time" -> (gettime_func, [\ []], "get_time")
  | _ -> |
  |
  let (fdef, local, result) = construct_func_call f args m builder in
  |
  (builder, m, (construct_call f)::dl)

| SSelect cl1 -> |

  (* create array of clause structs *)
  |
  let num elems = List.length cl1 in
  |
  let ptr = L.build_array_malloc clause_t
    (L.const_int i32_t num elems) "" builder in
  |
  let val_ptrs = List.mapi (fun i clause -> |
    |
    let op = (function
      (SSend(_, _)) -> L.const_int i8_t (Char.code 's')
    | (SRecv(_, _)) -> L.const_int i8_t (Char.code 'r')
    | _ -> raise(Failure("invalid select clause in codegen "))
    ) clause in
  |
  let chexpr = (function
    (SSend(c, _)) -> c
  | (SRecv(c, _)) -> c
  | _ -> raise(Failure("invalid select clause in codegen "))
  ) clause in
  |
  let chan = expr m builder chexpr in
  |
  let chtyp = (function
(SSend(_, (t, _)), _) -> t
| (SRecv(_, t), _) -> t
| _ -> raise(Failure("invalid select clause in codegen "))

) clause in
let local = L.buildalloca (ltype_of_typ chtyp) ""
builder in
let value = (function
  (SSend(_, e), _) -> expr m builder e
| (SRecv(_, _), _) -> L.const_int (ltype_of_typ chtyp) 0
| _ -> raise(Failure("invalid select clause in codegen "))
) clause in
ignore(L.buildstore value local builder);
let void_ptr_val = L.buildbitcast local void_ptr_t ""
builder in
let len = L.sizeof (ltype_of_typ chtyp) in
let idxs = List.rev (generate_seq (4 - 1)) in
let elem = List.fold_left2 (fun agg i v ->
  L.buildinsertvalue agg v i "clause" builder)
  (L.constnull clause_t) idxs [op ; chan ; void_ptr_val ; len] in
let idx = L.constint i32_t i in
let eptr = L.buildgep ptr [|idx|] "" builder in
let cptr = L.buildpointercast eptr
  (L.pointer_type (L.type_of elem)) "" builder in
let _ = (L.buildstore elem cptr builder)
in local)
cl
in
(* call C function *)
let selected_clause = L.buildcall select_func
  [| ptr ; L.constint i32_t num_elems |] "select" builder
in
(* branch to selected case *)
let merge_bb = L.appendblock context "merge" the_function
in
let build_br_merge = L.buildbr merge_bb in
let (_, case_bbs) = List.fold_left2 (fun (i, cases) clause
  val_ptr ->
  let case_bb = L.appendblock context "case"
    the_function in
  let case_stmt, mm = (function
    SBlock(SAssignStmt(sass) :: case_block) ->
    (* assign received value *)
  )
  in case_bb)
in
let e' = L.build_load val_ptr "recv_val" builder in
let rec handle_case_pred = function
  SAssign([([_, SId(id)], _)]) ->
    ignore(L.build_store e' (lookup id m) builder); m
| SDeclAssign([([id, vdt], _)] = stmt m dl builder {
  SVdeclStmt [(id, vdt)]
} in
  ignore(L.build_store e' (lookup id mm) builder); mm
| SInit([sa]) -> handle_case_pred(sa)
| _ -> raise(Failure("invalid case predicate detected in codegen"))

let mm = handle_case_pred sass in
SBlock(case_block), mm
| SBlock(_ :: case_block) -> SBlock(case_block), m
| _ -> raise(Failure("internal error: invalid case block in codegen"))

let (builder1, _, _) = stmt mm dl (L.builder_at_end context case_bb) in
add_terminal builder1 build_br_merge;
(i+1, case_bb :: cases)) (0, []) cl val_ptrs in
let sw = L.build_switch selected_clause merge_bb numelems builder in
List.iteri (fun i case_bb ->
  L.add_case sw (L.const_int i32_t i) case_bb;
) (List.rev case_bbs);
((L.builder_at_end context merge_bb), m, dl)
| _ -> raise(Failure("statement not implemented; should've checked in semant!"))

let (builder1, __, ndl) = stmt [local_vars ; global_vars] defered_el builder1 build_br_merge;
match L.block_terminator (L.insertion_block builder) with
Some _ -> ()
| None -> ignore(List.map (fun (fdef, llargs, result) ->
L.build_call fdef llargs result builder) ndl);

(* Add a return if the last block falls off the end *)
(* TODO: fix later *)
add_terminal builder (L.build_ret (L.const_int 132_t 0))
(* (match fdecl.styp with
A.Void -> L.build_ret_void
| t -> L.build_ret (L.const_int (ltype_of_typ t) 0)) *)
in
ignore(List.iter build_function_body functions);
the_module

9.8 concurrency.c

/*
 * RJEC concurrency functions
 * Written by Justin Chen and Elaine Wang
 * in collaboration with Riya and Caroline
 */

#include "libmill/libmill.h"
#include "libmill/chan.h"
#include <stdbool.h>
#include <string.h>
#include <sys/time.h>
#include <stdint.h>

void yeet(int (*start_routine)(void *), void *args)
{
    go(start_routine(args));
}

void *makechan(char typ, int buf)
{
    chan ch;
    switch(typ) {
    case 'i':
        ch = chmake(int, buf);
        break;
    case 'c':
        ch = chmake(char, buf);
        break;
    }
case 'b':
    ch = chmake(bool, buf);
    break;
}
return ch;
}

void send(void *ch_void_ptr, char typ, int val)
{
    chan ch = ch_void_ptr;
    switch(typ) {
        case 'i':
            chs(ch, int, val);
            break;
        case 'c':
            chs(ch, char, (char)val);
            break;
        case 'b':
            chs(ch, bool, (bool)val);
            break;
    }
}

int recv_int(void *ch_void_ptr)
{
    chan ch = ch_void_ptr;
    return chr(ch, int);
}

bool recv_bool(void *ch_void_ptr)
{
    chan ch = ch_void_ptr;
    return chr(ch, bool);
}

char recv_char(void *ch_void_ptr)
{
    chan ch = ch_void_ptr;
    return chr(ch, char);
}

void closechan(void *ch_void_ptr, char typ)
{
    chan ch = ch_void_ptr;
    switch(typed) {
case 'i':
    chdone(ch, int, 0);
    break;

case 'c':
    chdone(ch, char, 0);
    break;

case 'b':
    chdone(ch, bool, 0);
    break;
}
}

struct select_clause {
    char op;
    void *ch;
    void *val;
    long len;
};

int selectchan(struct select_clause *clauses, int nclauses) {
    mill_choose_init_(MILL_HERE_);
    int mill_idx = -2;
    while(1) {
        struct mill_clause mill_clauses[nclauses];
        for (int i = 0; i < nclauses; ++i) {
            if (clauses[i].op == 's') {
                mill_choose_out_((void *) &mill_clauses[i], clauses[i].ch,
                                clauses[i].val, clauses[i].len, i);
            } else if (clauses[i].op == 'r') {
                mill_choose_in_((void *) &mill_clauses[i], clauses[i].ch,
                                clauses[i].len, i);
            } else {
                mill_panic("invalid clause type");
            }
        }
        mill_idx = mill_choose_wait_();

        if (mill_idx != -2) {
            if (mill_idx < 0 || mill_idx >= nclauses) {
                mill_panic("invalid clause index");
            }
        }
    }
}
if (clauses[mill_idx].op == 'r') {
    memcpy(clauses[mill_idx].val, 
           mill_choose_val_(clauses[mill_idx].len), 
           clauses[mill_idx].len);
}
break;
}

// printf("select: got index %d\n", mill_idx);
return mill_idx;

int get_time()
{
    struct timeval tv;
    gettimeofday(&tv, NULL);
    return (((int64_t)tv.tv_sec * 1000000) + tv.tv_usec);
}

// test function for select
int food(chan ch1, chan ch2, chan quit)
{
    while (1) {
        bool q = 0;
        char val1 = 'a';
        int val2;

        /* libmill syntax:
         * choose {
         *     out(ch1, char, val1):
         *         printf("Value %c sent.\n", val1);
         *     in(ch2, int, val2):
         *         printf("Value %d received.\n", val2);
         *     in(quit, bool, q):
         *         printf("Bool value %d received\n", q);
         *         if (q) {
         *             printf("quitting...\n");
         *         }
         *     return 0;
         * }
         */

        struct select_clause clauses[] = {
            {'s', ch1, &val1, sizeof(val1)},
        }
```c
{'r', ch2, &val2, sizeof(val2)},
{'r', quit, &q, sizeof(q)}
};
int rc = selectchan(clauses, 3);
if (rc == 0) {
    printf("Value %c sent.\n", val1);
}
if (rc == 1) {
    printf("Value %d received.\n", val2);
}
if (rc == 2) {
    printf("Bool value %d received\n", q);
    if (q) {
        printf("quitting...\n");
    }
    return 0;
}
return 0;
}

/* test driver for select test:
int main()
{
    chan ch1 = chmake(char, 0);
    chan ch2 = chmake(int, 0);
    chan quit = chmake(bool, 10);
    go(food(ch1, ch2, quit));
    for (int i = 0; i < 5; ++i) {
        printf("%c\n", chr(ch1, char));
        chs(ch2, int, i);
    }
    chs(quit, bool, true);
}*/

9.9 printbool.c

/*
 * RJEC printbool function
 * Written by Riya Chakraborty
 * in collaboration with Elaine, Caroline, and Justin
 */
#include <stdio.h>

void printbool(int b)
{
    if (b) {
        printf("true\n");
    } else {
        printf("false\n");
    }
}

9.10 Makefile

# Makefile for building RJEC compiler
# Initially based on MicroC
# Written by Justin Chen, Elaine Wang, Riya Chakraborty, and Caroline Hoang

LDFLAGS = -L libmill/
LDLIBS = -lmill

# "make test" Compiles everything and runs the regression tests
.PHONY : test
test : all testall.sh
    ./testall.sh

# "make all" builds the executable as well as the "printbig" library designed
# to test linking external code
.PHONY : all
all : rjec.native printbool.o libmill/libmill.a concurrency.o

# builds the libmill library used for concurrency in RJEC
libmill/libmill.a :
    ./buildlibmill.sh

# "make rjec.native" compiles the compiler
#
# The _tags file controls the operation of ocamlbuild, e.g., by including
# packages, enabling warnings
#
# See https://github.com/ocaml/ocamlbuild/blob/master/manual/manual.adoc

rjec.native :
opam config exec -- \ocamlbuild -X libmill/ -use-ocamlfind rjec.native

# "make clean" removes all generated files

.PHONY : clean
clean :
ocamlbuild -clean
rm -rf testall.log ocamlllvm *.diff
rm *.o *.ll *.s *.exe

9.11 buildlibmill.sh

#!/bin/sh

# builds the libmill library used for concurrency in RJEC
# written by Justin Chen

cd libmill
cmake --configure .
cmake --build .
cd ..

9.12 rjec.sh

#!/bin/sh

# adapted from testall.sh, which is based on MicroC
# Written by Justin Chen

# Path to the LLVM interpreter
LLI="lli"

# Path to the LLVM compiler
LLC="llc"

# Path to the C compiler
CC="cc"

# Path to the rjec compiler. Usually "./rjec.native"
# Try "/build/rjec.native" if ocamlbuild was unable to create a
# symbolic link.
RJEC="/rjec.native"

if [ −z "$1" ]
then
    echo "Usage: ./rjec.sh <filename.rjec>"
    exit 1
fi

keep=0

if [ −z "$LD_LIBRARY_PATH" ]
then
    LD_LIBRARY_PATH=$(pwd)/libmill
    export LD_LIBRARY_PATH
else
    echo "check that LD_LIBRARY_PATH is exported!"
fi

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

generatedfiles=""
generatedfiles="$generatedfiles ${basename}.ll ${basename}.s ${basename}.exe ${basename}.out" &&
eval "$RJEC" "$1" ">${basename}.ll" &&
eval "$LLC" "-relocation-model=pic" ">${basename}.ll" ">${basename}.s" &&
eval "$CC" "-o" "${basename}.exe" "${basename}.s" "printbool.o" "concurrency.o" "-L" "libmill/" "-lmill" &&
eval "./${basename}.exe"

if [ $keep −eq 0 ] ; then
    rm −f $generatedfiles
fi
9.13 testall.sh

#!/bin/sh

# Regression testing script for RJEC
# Initially based on MicroC
# Written by Justin Chen, Elaine Wang, Riya Chakraborty, and
  Caroline Hoang
# 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"

# Path to the C compiler
CC="cc"

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

# Set time limit for all operations
ulimit -t 30

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

keep=0

if [ -z "LD_LIBRARY_PATH" ]
then
  LD_LIBRARY_PATH=$(pwd)/libmill
  export LD_LIBRARY_PATH
  echo "export LD_LIBRARY_PATH=$(pwd)/libmill"
else

Riya Chakraborty, Justin Chen, Yuanyuting Wang, Caroline Hoang 134
echo "check that LD_LIBRARY_PATH is exported!"
fi

Usage() {
    echo "Usage: testall.sh [options] [.rjec 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: $1 did not report an error"
        return 1
    )
    return 0
}

Check() {
    error=0
    basename=`echo $1 | sed 's/.*\///
        s/.rjec//'`
    reffile=`echo $1 | sed 's/.rjec$//'`
    basedir=`echo $1 | sed 's/\([^/]*\)//'`
    echo −n "$basename..."
    echo 1>&2
    echo "##### Testing $basename" 1>&2
    generatedfiles=""
    generatedfiles="$generatedfiles ${basename}.ll ${basename}.s ${basename}.exe ${basename}.out"
        && Run "$RJEC" "$1" '>' "${basename}.ll" &&
    Run "$LLC" "-relocation-model=pic" "${basename}.ll" '>' "${basename}.s" &&
    Run "$CC" "-o" "${basename}.exe" "${basename}.s" "printbool.o" "concurrency.o" "-L" "libmill/" "-lmill" &&
    Run "./${basename}.exe" > "${basename}.out" &&
    Compare ${basename}.out ${reffile}.out ${basename}.diff
    # Report the status and clean up the generated files
    if [ $error -eq 0 ] ; then
        if [ $keep -eq 0 ] ; then
            rm -f $generatedfiles
        fi
        echo "OK"
        echo "##### SUCCESS" 1>&2
    else
        echo "##### FAILED" 1>&2
        globalerror=$error
        fi
    }

CheckFail() {

Riya Chakraborty, Justin Chen, Yuanyuting Wang, Caroline Hoang
error=0
basename=`echo $1 | sed 's/.*\///
 s/rjec//'`
reffile=`echo $1 | sed 's/.rjec$//'`
basedir=`echo $1 | sed 's/\([^/\]*/\///
'

echo −n "$basename..."
echo 1>&2
echo "##### Testing $basename" 1>&2
generatedfiles=""
generatedfiles="$generatedfiles ${basename}.err ${basename}.diff
" &
RunFail "$RJEC" "<" $1 "2>" "${basename}.err ">>" $globallog &
Compare ${basename}.err ${reffile}.err ${basename}.diff

# Report the status and clean up the generated files
if [ $error =eq 0 ]; then
  if [ $keep =eq 0 ]; then
    rm −f $generatedfiles
  fi
echo "OK"
echo "##### SUCCESS" 1>&2
else
  echo "##### FAILED" 1>&2
  globalerror=$error
  fi
}

while getopts kdpsh c; do
  case $c in
    k) # Keep intermediate files
       keep=1
       ;;
    h) # Help
       Usage
       ;;
    esac
done
shift `expr $OPTIND − 1`
LLIFail() {
    echo "Could not find the LLVM interpreter "$LLI"."
    echo "Check your LLVM installation and/or modify the LLI variable in testall.sh"
    exit 1
}

which "$LLI" >> $globallog || LLIFail

if [ $# -ge 1 ]
then
    files=$@
else
    files="tests/test-*.rjec tests/fail-*.rjec"
fi

for file in $files
do
    case $file in
        *test-*)
        Check $file 2>> $globallog ;;
    *fail-*)
        CheckFail $file 2>> $globallog ;;
    *)
        echo "unknown file type $file"
        globalerror=1
        ;;
    esac
done

exit $globalerror

9.14 tests/

9.14.1 fail-array1.err

Fatal error: exception Parsing.Parse_error

9.14.2 fail-array1.rjec

Riya Chakraborty, Justin Chen, Yuanyuting Wang, Caroline Hoang
func main() {
    var x [10][]int;
}

9.14.3 fail-array2.err

Fatal error: exception Failure("array size can only be integer expressions")

9.14.4 fail-array2.rjec

func main() {
    var x [true]int;
}

9.14.5 fail-array3.err

Fatal error: exception Failure("illegal assignment of element to array")

9.14.6 fail-array3.rjec

struct foo {
    x int;
    y char;
    z bool;
}

func main () {
    var a [10]struct foo;
    for i := 0; i < 10; i = i + 1 {
        a[i] = i;
    }
}
9.14.7 fail-array4.err

Fatal error: exception Failure("array element doesn't match declared array type")

9.14.8 fail-array4.rjec

```go
func main() {
    x := []bool {1, 2, 3, 4, 5};
}
```

9.14.9 fail-chan1.err

Fatal error: exception Failure("tried to perform channel operation through a non-channel variable")

9.14.10 fail-chan1.rjec

```go
func main () {
    ch := 5;
    ch <= 10;
    println(<-ch);
}
```

9.14.11 fail-decl1.err

Fatal error: exception Failure("duplicate global i")

9.14.12 fail-decl1.rjec

```go
var i int;
var i bool;
```
func main() {
    printi(2);
}

9.14.13 fail-decl2.err

Fatal error: exception Failure("duplicate global i")


var i int;
var i int;

func main() {
    printi(2);
}

9.14.15 fail-decl3.err

Fatal error: exception Failure("local variable i has already been declared")

9.14.16 fail-decl3.rjec

func main() {
    i, i := 2, 3;
}


Fatal error: exception Failure("duplicate struct foo")
9.14.18 fail-decl4.rjec

```go
struct foo {
    x bool;
    y int;
}

struct foo {
    x int;
    y int;
}

func main() {
}
```

9.14.19 fail-decl5.err

```
Fatal error: exception Failure("duplicate member in definition of struct foo : x")
```

9.14.20 fail-decl5.rjec

```go
struct foo {
    x bool;
    x int;
}

func main() {
}
```

9.14.21 fail-decl6.err

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

```go
func foo(a int, a bool) bool {
    return true;
}
func main() {
}
```

9.14.23  fail-defer1.err

```
Fatal error: exception Failure("defer statement inside of while block")
```

9.14.24  fail-defer1.rjec

```go
func main() {
    for i := 2; i < 3; i = i + 1 {
        defer printi(i);
    }
}
```

9.14.25  fail-defer2.err

```
Fatal error: exception Failure("defer statement inside of while block")
```

9.14.26  fail-defer2.rjec

```go
func main() {
    i := 1;
    for i < 3 {
        defer printi(i);
        i = i + 1;
    }
```
9.14.27 fail-defer3.err

Fatal error: exception Failure("defer statement inside of while block")

9.14.28 fail-defer3.rjec

```go
func main() {
    for i := 2; i < 3; i = i + 1 {
        j := 0;
        for j < 5 {
            defer printi(j);
        }
        j = j + 1;
    }
}
```

9.14.29 fail-func1.err

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

9.14.30 fail-func1.rjec

```go
func main() {
    foo();
}
```

9.14.31 fail-func2.err

Fatal error: exception Failure("duplicate function baz")
9.14.32  fail-func2.rjec

```go
func baz(x int, y bool) bool {
    return x;
}

func baz(z int) {
    printi(z);
}

func main(){
}
```

9.14.33  fail-func3.err

```
Fatal error: exception Failure("built-in function printi is already defined")
```

9.14.34  fail-func3.rjec

```go
func printi(x int) {
    printi(x);
}

func main() {
}
```

9.14.35  fail-struct1.err

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

9.14.36  fail-struct1.rjec

```go
func main() {
```
var i struct foo;
i = struct foo {
    x : 1,
    y : true,
    z : 'a'
};
printi(i.x);
printb(i.y);
printc(i.z);
}

9.14.37 fail-struct2.err

Fatal error: exception Failure("undeclared reference foo")

9.14.38 fail-struct2.rjec

func main() {
    printi(foo.x);
}


Fatal error: exception Failure("unknown field v in struct foo")


struct foo {
    x int;
    y bool;
    z char;
}

func main() {
    var i struct foo;
i = struct foo {
    x : 1,
y: true,
z: 'a'
};
print(i.v);

9.14.41 fail-yeet1.err

Fatal error: exception Failure("a yeet function call can only return int or nothing")

9.14.42 fail-yeet1.rjec

func isEven (x int) bool {
    return x % 2 == 0;
}
func main () {
    yeet isEven(4);
}

9.14.43 fail-yeet2.err

Fatal error: exception Failure("can't yeet a non-function call")

9.14.44 fail-yeet2.rjec

func main () {
    yeet 1 + 2;
}

9.14.45 test-array1.out

0
9.14.46  test-array1.rjec

```go
func main () {
    var x, y [10]int;
    for i := 0; i < 10; i = i + 1 {
        printi(x[i]);
    }
}
```

9.14.47  test-array10.out

```
10
10
10
10
```

9.14.48  test-array10.rjec

```go
func foo (x []int) []int{
    y := x;
    return y;
}

func main () {
    x := []int {1, 2, 3, 4, 5};
    y := foo(x);
    y[1] = 10;
    printi(y[1]);
}
```
printi(x[1]);

z := []int {1, 2, 3, 4, 5};
t := z;
z[1] = 10;
printi(z[1]);
printi(t[1]);

9.14.49 test-array11.out

5
0

9.14.50 test-array11.rjec

func main() {
    n := 5;
    printi(n);
    var arr [n]int;
    printi(arr[2]);
}

9.14.51 test-array2.out

0
false
0
false
0
false
0
false
0
false
0
false

Riya Chakraborty, Justin Chen, Yuanyuting Wang, Caroline Hoang
9.14.52 test-array2.rjec

```go
struct foo {
    x int;
    y char;
    z bool;
}

func main () {
    var a [10]struct foo;
    for i := 0; i < 10; i = i + 1 {
        s := a[i];
        println(s.x);
        printb(s.z);
    }
}
```

9.14.53 test-array3.out

```
a
```
9.14.54  test-array3.rjec

```go
struct foo {
  x int;
  y char;
  z bool;
}

func my_func (n int, c chan int) {
  var a [n]struct foo;
  for i := 0; i < n; i = i + 1 {
    s := a[i];
    printi(s.x);
    printb(s.z);
  }
  for {
    s := a[<- c];
    printc('a');
  }
}

func main() {
  c := make(chan int);
  yeet my_func(5, c);
  c <- 1;
  c <- 2;
  c <- 3;
}
```

9.14.55  test-array4.out

```
0
2
4
6
8
```

9.14.56  test-array4.rjec
```go
func foo (n int) {
    var x [n]int;
    for i := 0; i < n ; i = i + 1 {
        x[i] = i * 2;
    }
    for i := 0; i < n ; i = i + 1 {
        printi(x[i]);
    }
}

func main () {
    foo(5);
}
```

9.14.57  test-array5.out

```
0
false
0
false
0
false
0
false
0
false
0
false
a
a
a
```

9.14.58  test-array5.rjec

```go
struct foo {
    x int;
    y char;
    z bool;
}

func my_func (n int, c chan int) {
    var a [n]struct foo;
```
for $i := 0; i < n; i = i + 1$ {
    printi(a[i].x);
    a[i].y = 'a';
    printb(a[i].z);
}
for {
    printc(a[<- c].y);
}

func main() {
    c := make(chan int);
    yeet my_func(5, c);
    c <- 1;
    c <- 2;
    c <- 3;
}


6

9.14.60 test-array6.rjec

func foo(x []int) {
    x[0] = x[0] + 2;
}

func main() {
    var x [10]int;
    x[0] = 4;
    foo(x);
    printi(x[0]);
}

9.14.61 test-array7.out

```go
struct foo {
    x int;
    y int;
    z bool;
}

func change_struct(x []struct foo) {
    x[0].x = x[1].x + 1;
    x[1].y = x[2].y - 2;
    x[2].z = true;
}

func main() {
    var x [10]struct foo;
    x[0] = struct foo { x: 1, y: 2 };  
    x[1] = struct foo { x: 3, y: 5, z: true };  
    x[2] = struct foo { x: 0, y: 7};
    change_struct(x);
    printi(x[0].x);
    printi(x[1].y);
    printb(x[2].z);
}
```

9.14.63  test-array8.out

```
1
2
3
4
5
```

9.14.64  test-array8.rjec
func main () {
    x := []int{1, 2, 3, 4, 5};
    for i := 0; i < 5; i = i + 1 {
        printi(x[i]);
    }
}


120


func factorial (x []int, l int) int{
    ret := 1;
    for i := 0; i < l; i = i + 1 {
        ret = ret \times x[i];
    }
    return ret;
}
func main () {
    printi(factorial([]int{1, 2, 3, 4, 5}, 5));
}

9.14.67 test-assign1.out

52
false
true

9.14.68 test-assign1.rjec

var i int;
var b bool;
var j int;
func main() {
    i = 42;
    j = 10;
    printi(i + j);
    printb(b);
    b = true;
    printb(b);
}

9.14.69  test-assign2.out

52

9.14.70  test-assign2.rjec

var i, j int;

func main() {
    i = 10;
    j = 42;
    printi(i + j);
}

9.14.71  test-assign3.out

52

9.14.72  test-assign3.rjec

var i, j int;

func main() {
    i, j = 10, 42;
    printi(i + j);
}
9.14.73  test-assign4.out

0
0
5
6
6
11

9.14.74  test-assign4.rjec

```go
var i, j, k int;

func main() {
    printi(i);
    printi(j);
    i, j = 5, 6;
    k = i+j;
    printi(i);
    i = i+1;
    printi(i);
    printi(j);
    printi(k);
}
```

9.14.75  test-assign5.out

0
0
5
6
5
6

9.14.76  test-assign5.rjec

```go
/* TO DO: either implement or raise error */
var i, j int;
```
func main() {
    printi(i);
    printi(j);
    i, j = 5, 6;
    printi(i);
    printi(j);
    /* i, j = j, i; */
    printi(i);
    printi(j);
}

9.14.77 test-chan1.out

9.14.78 test-chan1.rjec

func main() {
    var foo chan int;
    foo = make(chan int);
    foo2 := make(chan bool, 10);
}

9.14.79 test-chan2.out

10
2
8

9.14.80 test-chan2.rjec

func main () {
    ch := make(chan int, 3);
    ch <- 10;
    ch <- 2;
    ch <- 8;
    printi(<-ch);
}
9.14.81 test-chan3.out

```
r
j
e
c
```

9.14.82 test-chan3.rjec

```go
func foo(x chan char)
    for {
        a := <- x;
        printc(a);
    }
}
func main () {
    x := make(chan char);
    yeet foo(x);
    x <- 'r';
    x <- 'j';
    x <- 'e';
    x <- 'c';
}
```

9.14.83 test-chan4.out

```
receiver:
  2
sender:
  1
receiver:
  3
sender:
  2
```
9.14.84 test-chan4.rjec

```go
def foo1(r chan int, s chan int) {
    for {
        a := <- r;
        println("receiver:");
        printi(a + 1);
        s <- (a * 2);
    }
}
```
func main () {
    s, r := make(chan int), make(chan int);
    yeet fool(s, r);
    for a := 1; a < 1000 ; a = (<− r) {
        s <- a;
        prints("sender:");
        printi(a);
    }
}

9.14.85  test-chan5.out

1
2
3
0
0

9.14.86  test-chan5.rjec

func foo (c chan int) {
    c <- 1;
    c <- 2;
    c <- 3;
    close(c);
}
func main () {
    c := make(chan int, 3);
    yeet foo(c);
    for i := 0; i < 5; i = i + 1 {
        a := <- c;
        printi(a);
    }
}

9.14.87  test-decl-assign1.out
9.14.88 test-decl-assign1.rjec

```go
cfunc main() { 
    var i int = 42; 
    var j int = 30; 

    printi(i); 
    printi(j); 
}
```

9.14.89 test-decl-assign2.out

```
42
30
```

9.14.90 test-decl-assign2.rjec

```go
cfunc main() { 
    var x, y int = 42, 30; 
    printi(x); 
    printi(y); 

    var r, j, e, c char = 'r', 'j', 'e', 'c'; 
    printc(r); 
    printc(j); 
    printc(e); 
    printc(c); 
}
```
9.14.91  test-decl-assign3.out

true
12


```go
struct foo {
    x bool;
    y int;
}

func main() {
    var i struct foo = struct foo {
        x : true,
        y : 12
    };
    printf(i.x);
    printf(i.y);
}
```

9.14.93  test-decl-assign4.out

42
30
60
52
42
60

9.14.94  test-decl-assign4.rjec

Riya Chakraborty, Justin Chen, Yuanyuting Wang, Caroline Hoang
```go
func main() {
    var i, j int = 42, 30;
    println(i);
    println(j);

    if true {
        j = 60;
        println(j);
        var i int = 52;
        println(i);
    }

    println(i);
    println(j);
}
```

**9.14.95 test-decl1.out**

```
52
false
true
```

**9.14.96 test-decl1.rjec**

```go
func main() {
    var i int;
    var b bool;
    var j int;

    i = 42;
    j = 10;
    println(i + j);
    println(b);
    b = true;
    println(b);
}
```

**9.14.97 test-decl2.out**
9.14.98  test-decl2.rjec

```go
var i int;

func main() {
    i = 42;
    printi(i);
    var i bool;
    printb(i);
    i = true;
    printb(i);
}
```

9.14.99  test-decl3.out

```
42
30
60
52
42
60
```

9.14.100 test-decl3.rjec

```go
func main() {
    var i int;
    var j int;

    i = 42;
    printi(i);
    j = 30;
    printi(j);

    if true {
```
```go
j = 60;
printi(j);
var i int;
i = 52;
printi(i);
}

printi(i);
printi(j);
```

**9.14.101 test-defer1.out**

```
2
3
1
```

**9.14.102 test-defer1.rjec**

```go
func foo (x int)
{
    printi(x);
}

func main() {
    defer foo(1);
    foo(2);
    foo(3);
}
```

**9.14.103 test-defer2.out**

```
4
5
3
2
1
```

**9.14.104 test-defer2.rjec**

---

Riya Chakraborty, Justin Chen, Yuanyuting Wang, Caroline Hoang
func foo (x int){
    printi(x);
}

func main() {
    defer printi(1);
    defer printi(2);
    defer foo(3);

    foo(4);
    printi(5);
}

9.14.105  test-defer3.out

3
8
2
8
18
7
126
16


func foo (x int, y int) int {
    defer printi(x);
    defer printi(y);

    z := x * y;
    x = x + 1;
    printi(x);
    return z;
}

func main() {
    x, y := 2, 8;
    defer printi(foo(x, y));
for i := 0; i < 5; i = i + 1 {
    x = x + 1;
    y = y + 2;
}
printi(foo(x, y));

9.14.107 test-defer4.out

3
2
1


func main() {
    if true {
        defer printi(1);
        if true {
            defer printi(2);
        }
        defer printi(3);
    }
}

9.14.109 test-for1.out

5
4
3
2
1

9.14.110 test-for1.rjec

func main() {
i := 5;
for 0 < i {
    printi(i);
    i = i - 1;
}

9.14.111 test-for2.out
5
4
3
2
1

9.14.112 test-for2.rjec
func main() {
    for i := 5; 0 < i; i = i - 1 {
        printi(i);
    }
}

9.14.113 test-for3.out
5
4
3
2
1
5
4
3
2
1

Riya Chakraborty, Justin Chen, Yuanyuting Wang, Caroline Hoang

```go
func main() {
    i := 5;
    for ; 0 < i; i = i - 1 {
        println(i);
    }
    for j := 5; 0 < j; {
        println(j);
        j = j - 1;
    }
}
```

9.14.115 test-for4.out

```
2
4
6
8
10
12
14
16
18
20
```


```go
func foo(x int, y int) bool {
    return !(y % x == 0);
}
func main() {
    i, j := 11, 2;
    for ; foo(i, j); j = j + 2 {
```
func main() {
    for i:=0; i<4; i=i+1 {
        for j:=0; j<4; j=j+1 {
            printi(i+j);
        }
    }
}
9.14.120  test-func1.rjec

```go
func foo (x int) int {
    return x + 2;
}

func main() {
    var i int;
    i = foo(1);
    printi(i);
```

9.14.121  test-func2.out

```
true
false
false
3
0
```

9.14.122  test-func2.rjec

```go
func isEven (x int, y bool) bool {
    if (x % 2 == 0 && y) {
        return true;
    } else {
        return false;
    }
    return true;
}

func foo (x int) {
    printi(x % 4);
}

func main() {
    printb(isEven(2, true));
    printb(isEven(3, true));
    printb(isEven(4, false));
```
9.14.123  test-func3.out

1
8
11
1

9.14.124  test-func3.rjec

```go
func gcd (a int, b int) int {
    if (a < b) {
        return gcd(b, a);
    }
    if (b == 0) {
        return a;
    }
    return gcd(b, a % b);
}

func main () {
    printi(gcd(32, 3));
    printi(gcd(16, 56));
    printi(gcd(869, 143));
    printi(gcd(349, 1837));
}
```

9.14.125  test-func4.out

2
3

9.14.126  test-func4.rjec

Riya Chakraborty, Justin Chen, Yuanyuting Wang, Caroline Hoang
```go
struct dummy {
    x int;
    y int;
}

func foo (x int, y int) struct dummy {
    var i struct dummy;
    i = struct dummy {
        x : x,
        y : y
    };
    return i;
}

func main () {
    var i struct dummy;
    i = foo(2, 3);
    printi(i.x);
    printi(i.y);
}
```

9.14.127 test-5.out

```
2
3
false
z
10
```

9.14.128 test-5.rjec

```go
struct dummy {
    x int;
    y int;
    z bool;
    t char;
}

func foo (x int, y int, z struct dummy) struct dummy {
    var i struct dummy;
}
```go
func main () {
    var i, j struct dummy;
    j = struct dummy {
        x : 10,
        t : 'z'
    };
    i = foo(2, 3, j);

    printi(i.x);
    printi(i.y);
    printb(i.z);
    printc(i.t);

    printi(j.x);
}
```

9.14.129  test-hello.out

```
hello world
321
888
ttrue
false
r
J
e
C
```

9.14.130  test-hello.rjec

Riya Chakraborty, Justin Chen, Yuanyuting Wang, Caroline Hoang
func main() {
    println("hello world");
    printi(321);  
    printi(888);  
    printb(true); 
    printb(false);  
    printc('r'); 
    printc('J');  
    printc('e');  
    printc('C'); 
}

9.14.131 test-if1.out

321
888
Hi friends

9.14.132 test-if1.rjec

func main() {
    if true {
        printi(321); 
        printi(888); 
    } 
    println("Hi friends"); 
}

9.14.133 test-if2.out

42
Hi friends

9.14.134 test-if2.rjec
```go
func main() {
    if true {
        printi(42);
    } else {
        printi(8);
    }

    prints("Hi friends");
}
```

**9.14.135 test-if3.out**

```
Hi friends
```

**9.14.136 test-if3.rjec**

```go
func main() {
    if false {
        prints("no");
    }

    prints("Hi friends");
}
```

**9.14.137 test-if4.out**

```
8
Hi friends
```

**9.14.138 test-if4.rjec**

```go
func main() {
    if false {
```
prints("no");
}
else {
    printi(8);
}

prints("Hi friends");

9.14.139 test-if5.out

8
Hi friends

9.14.140 test-if5.rjec

func main() {
    if false {
        prints("no");
    }
    else if true {
        printi(8);
    }

    prints("Hi friends");
}

9.14.141 test-if6.out

2
Hi friends

9.14.142 test-if6.rjec
func main() {
    if false {
        println("no");
    } else if true {
        if 1+1==2 {
            println(2);
        } else {
            println(100);
        }
    } else {
        println("bye");
    }

    println("Hi friends");
}

9.14.143 test-init1.out

42
30
true
60
52
42
60

9.14.144 test-init1.rjec

func main() {
    i, j := 42, 30;
    b := true;
    println(i);
    println(j);
    printb(b);

    if true {
j = 60;
printi(j);
i := 52;
printi(i);
}
printi(i);
printi(j);
}

9.14.145  test-init2.out

1
r
false
2
j
false
1
r
false
2
j
false
3
1
r
false

9.14.146  test-init2.rjec

struct my_struct {
  a int;
  b char;
  c bool;
}

func foo(x int, y char) struct my_struct {
  i := struct my_struct {
    a : x,
    b : y
  }
}
};
printi(i.a);
printc(i.b);
printb(i.c);
return i;
}

func main() {

t, k, l := foo(1, 'r'), foo(2, 'j'), 3;
printi(t.a);
printc(t.b);
printb(t.c);
printi(k.a);
printc(k.b);
printb(k.c);
printi(l);

a, b, c := t.a, t.b, t.c;
printi(a);
printc(b);
printb(c);
}

9.14.147 test-ops1.out

3
-1
2
50
0
99
false
t
true
99
t
true
false
99
t
true

```go
func main() {
    printi(1 + 2);
    printi(1 - 2);
    printi(1 * 2);
    printi(100 / 2);
    printi(4 % 2);
    printi(99);
    printb(1 == 2);
    printb(1 == 1);
    printi(99);
    printb(1 < 2);
    printb(2 < 1);
    printi(99);
    printb(1 <= 2);
    printb(1 <= 1);
    printb(2 <= 1);
    printi(99);
}
```

9.14.149  test-ops2.out

```text
true
false
99
true
false
true
false
false
false
true
true
true
false
true
false
-10
```
9.14.150  test-ops2.rjec

```go
func 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);
    printi(-10);
}
```

9.14.151  test-select1.out

```
777
888
1313
```

9.14.152  test-select1.rjec

```go
func foo(a chan int, b chan int, c chan int) {
    for {
        select {
            case i := <- a:
                printi(i);
            case j := <- b:
                printi(j);
            case k := <- c:
                printi(k);
        }
    }
}
```
9.14.153 test-select2.out

777
true
a

9.14.154 test-select2.rjec

func foo(a chan int, b chan bool, c chan char) {
    for {
        select {
            case i := <- a:
                println(i);
            case j := <- b:
                println(j);
            case k := <- c:
                println(k);
        }
    }
}

func main() {
    a := make(chan int);
    b := make(chan bool);
    c := make(chan char);
    yeet foo(a, b, c);
    a <- 777;
    b <- true;
    c <- 'a';
}
func foo(a chan int, b chan int, c chan int) {
    for {
        var i int;
        select {
            case i = <- a:
                printi(i);
            case var j int = <- b:
                printi(j);
            case k := <- c:
                printi(k);
        }
    }
}

func main() {
    a := make(chan int);
    b := make(chan int);
    c := make(chan int);
    yeet foo(a, b, c);
    a <- 777;
    b <- 888;
    c <- 1313;
}
func foo(a chan int, b chan int, c chan int) {
    val := 1313;
    for {
        select {
            case a <- 777:
                printi(777);
            case b <- 888:
                printi(888);
            case c <- val:
                printi(val);
        }
    }
}

func main() {
    a := make(chan int);
    b := make(chan int);
    c := make(chan int);
    yeet foo(a, b, c);
    printi(<-a);
    printi(<-b);
    printi(<-c);
}

9.14.159  test-select5.out

0
1

9.14.160  test-select5.rjec

func foo(a chan int, b chan int) {
    idx := -1;
}
for {
    select {
        case i := <- a:
            idx = 0;
        case j := <- b:
            idx = 1;
    }
    printi(idx);
}

func main() {
    a := make(chan int);
    b := make chan int);
    yeet foo(a, b);
    a <- 0;
    b <- 1;
}

9.14.161 test-select6.out

sending:
a

receiving:
a

sending:
0

sending:
a

receiving:
1

sending:
a

receiving:
2

sending:
a

receiving:
3

sending:
a

Riya Chakraborty, Justin Chen, Yuanyuting Wang, Caroline Hoang
func foo(ch1 chan char, ch2 chan int, quit chan bool) {
    for {
        select {
            case ch1 <- 'a':
                println("sending:");
                printc('a');
            case val2 := <- ch2:
                println("receiving:");
                printi(val2);
            case q := <- quit:
                println(q);
                if q {
                    println("quitting...");
                }
                return;
        }
    }
}

func main() {
    ch1 := make(chan char);
    ch2 := make(chan int);
    quit := make(chan bool, 10);
    yeet foo(ch1, ch2, quit);
    for i := 0; i < 5; i = i + 1 {
        printc(<<-ch1);
        ch2 <- i;
    }
    quit <- true;
}
9.14.164 test-select7.rjec

```go
func producer (data chan int, quit chan int) {
    i := 0;
    for {
        i = i + 1;
        select {
            case <- i:
            case <- quit:
                close(data);
                return;
        }
    }
}

func main () {
    data := make(chan int);
    quit := make(chan int);

    yeet producer(data, quit);

    for i := <-data; !(i == 0); i = <-data {
        printi(i);
        if i == 10 {
            quit <- 1;
            close(quit);
        }
    }
}
```
9.14.165 test-select8.out

sent 1 through c2
sent 1 through c2
1
2
10
2
2

9.14.166 test-select8.rjec

/* NOTE: might be race conditions causing variations of the test
   output */

func foo (x []chan int){
  c0, c1, c2, c3 := x[0], x[1], x[2], x[3];
  for {
    select {
      case i := <- c0:
        printi(i);
      case j := <- c1:
        printi(j);
      case c2 <- 1:
        prints("sent 1 through c2");
      case c3 <- 2:
    }
  }
}

func main () {
  x := []chan int {
    make(chan int),
    make(chan int, 2),
    make(chan int, 2),
    make(chan int)
  };

  yeet foo(x);
  c0, c1, c2, c3 := x[0], x[1], x[2], x[3];

  c0 <- 1;
c0 <- 2;
c1 <- 10;
printi(<- c3);
printi(<- c3);
}


sent 1 through c2
sent 1 through c2
1
2
10
2
2


/* NOTE: might be race conditions causing variations of the test output */

func foo (x []chan int){
    for {
        select {
            case i := <- x[0]:
                printi(i);
            case j := <- x[1]:
                printi(j);
            case x[2] <- 1:
                prints("sent 1 through c2");
            case x[3] <- 2:
            }
    }
}

func main () {
    x := []chan int {
        make(chan int),
        make(chan int, 2),
        make(chan int, 2),
        make(chan int)
yeet foo(x);
x[0] <- 1;
x[0] <- 2;
x[1] <- 10;
printi(<- x[3]);
printi(<- x[3]);

9.14.169  test-string1.out

e
l
l
o

9.14.170  test-string1.rjec

func main () {
    var x [5]char;
    x = "ello";
    for i := 0; i < 4; i = i + 1 {
        printc(x[i]);
    }
}

9.14.171  test-string2.out

e
l
l
o
ello
e
l
l
9.14.172 test-string2.rjec

```go
func main () {
    x := "ello";
    for i := 0; i < 4; i = i + 1 {
        printc(x[i]);
    }
    prints(x);

    x[1] = x[2];
    x[3] = x[0];
    for i := 0; i < 4; i = i + 1 {
        printc(x[i]);
    }
    prints(x);
}
```

9.14.173 test-string3.out

```
hello world
```

9.14.174 test-string3.rjec

```go
func foo (x []char) {
    prints(x);
}

func main () {
    foo("hello world");
}
```

9.14.175 test-string4.out
9.14.176  test-string4.rjec

```go
func main () {
    x := "ello";
    for i := 0; !(x[i] == '\0'); i = i + 1 {
        printc(x[i]);
    }
    prints(x);
    x[1] = x[2];
    x[3] = '\0';
    for i := 0; !(x[i] == '\0'); i = i + 1 {
        printc(x[i]);
    }
    prints(x);
}
```

9.14.177  test-struct1.out

```
true
```

9.14.178  test-struct1.rjec

```go
struct foo {
    x int;
    y bool;
}```
```go
z char;
}

func main() {
    var i struct foo;
    i = struct foo {
        x : 1,
        y : true,
        z : 'a'
    };
    printi(i.x);
    printb(i.y);
    printc(i.z);
}
```

### 9.14.179 test-struct2.out

```
false
0
```

### 9.14.180 test-struct2.rjec

```go
struct foo {
    x bool;
    y int;
}

func main() {
    var i struct foo;

    printb(i.x);
    printi(i.y);
}
```

### 9.14.181 test-struct3.out

```
false
0
true
```
9.14.182  test-struct3.rjec

```go
struct foo {
    x bool;
    y int;
    z bool;
}

func main() {
    var i struct foo;
i = struct foo {
        x : false,
        z : true
    };
pintf(i.x);
pintf(i.y);
pintf(i.z);

    i = struct foo {
        y : 10
    };
pintf(i.x);
pintf(i.y);
pintf(i.z);
}
```

9.14.183  test-struct4.out

```
true
9
true
```

9.14.184  test-struct4.rjec
```go
struct foo {
    x bool;
    y int;
    z bool;
}

func main() {
    var i struct foo;
    i.x = true;
    i.y = 9;
    i.z = true;

    printb(i.x);
    printi(i.y);
    printb(i.z);
}
```

**9.14.185 test-struct5.out**

false
0
true
false
10
true

**9.14.186 test-struct5.rjec**

```go
struct foo {
    x bool;
    y int;
    z bool;
}

func main() {
    var i struct foo;
    i = struct foo {
        x : false,
        z : true
    }
```
printb(i.x);
printi(i.y);
printb(i.z);

i.y = 10;

printb(i.x);
printi(i.y);
printb(i.z);


10
r
false
20
j
false
10
r
false

9.14.188 test-struct6.rjec

struct my_struct {
    a int;
    b char;
    c bool;
}

func foo(i struct my_struct, x int, y char) {
    i.a = x;
    i.b = y;

    printi(i.a);
    printc(i.b);
    printb(i.c);
}
func main() {
    j := struct my_struct {
        a : 10,
        b : 'r'
    };

    printj(j.a);
    printc(j.b);
    printb(j.c);

    foo(j, 20, 'j');

    printi(j.a);
    printc(j.b);
    printb(j.c);
}

9.14.189 test-time1.out

time elapsed greater than 0:
true

9.14.190 test-time1.rjec

func main()
{
    i := time();
    prints("time elapsed greater than 0:");
    printb(i < time());
}

9.14.191 test-yeet1.out

3
4
7

Riya Chakraborty, Justin Chen, Yuanyuting Wang, Caroline Hoang
9.14.192  test-yeet1.rjec

```go
func foo (x int, y int) int {
    printi(x);
    printi(y);
    printi(x + y);
    return x + y;
}

func main() {
    yeet foo(3, 4);
}
```

9.14.193  test-yeet2.out

```
1
3
false
```

9.14.194  test-yeet2.rjec

```go
struct lol {
    x int;
    y int;
    z bool;
}

func foo (x struct lol) {
    y := struct lol {
        x : x.x,
        y : x.y + 1
    };
    printi(y.x);
    printi(y.y);
    printb(y.z);
}

func main() {
    yeet foo(struct lol {
```
9.14.195  test-yeet3.out

2
2
2

9.14.196  test-yeet3.rjec

```go
func foo1 () int {
    return 1 + 2;
}
func foo2 (x int, y bool) int {
    if (x == 2 && y) {
        printi(x);
    } else {
        printi(x + 1);
    }
    return x;
}

func main() {
    yeet foo1();
    yeet foo2(2, true);
    yeet foo2(1, true);
    yeet foo2(1, false);
}
```

9.15  demo/
9.15.1  mapreduce.rjec

```bash
/* by Justin Chen
 * mapreduce to count randomly generated primes vs composite ints
```
* compares time length of iterative vs concurrent solutions
* based on noun/verb count mapreduce by appliedgo.net */

/* mapper receives a channel of ints and counts the number of prime and
* composite ints. It sends the resulting counts to the output channels.
*/

func mapper(in chan int, out_prime chan int, out_composite chan int) {
    num_prime := 0;
    num_composite := 0;
    for i := <- in; !(i == 0); i = <- in {
        if is_prime(i) {
            num_prime = num_prime + 1;
        } else {
            num_composite = num_composite + 1;
        }
    }
    out_prime <- num_prime;
    out_composite <- num_composite;
    close(out_prime);
    close(out_composite);
}

/* checks if positive integer is prime, using pseudocode from Wikipedia */

func is_prime(n int) bool {
    if n <= 3 {
        return 1 < n;
    }
    if n % 2 == 0 || n % 3 == 0 {
        return false;
    }
    for i := 5; i * i <= n; i = i + 6 {
        if n % i == 0 || n % (i + 2) == 0 {
            return false;
        }
    }
    return true;
}

/* reducer receives a channel of ints and adds up all ints until it receives the
* quit signal */
func reducer(in chan int, out chan int, quit chan bool) {
    sum := 0;
    for {
        select {
            case n := <- in:
                sum = sum + n;
            case <- quit:
                out <- sum;
                close(out);
        }
    }
}

func rand(seed int) int {
    rand_int := (1103515245 * seed + 12345) % 2147483648;
    if rand_int < 0 {
        return -1 * rand_int;
    }
    if rand_int == 0 {
        return 1;
    }
    return rand_int;
}

func input_reader(out []chan int, num_mappers int, num_ints int, rand_ints []int) {
    for i := 0; i < num_mappers; i = i + 1 {
        yeet input_helper(out[i], num_ints / num_mappers, rand_ints, num_ints / num_mappers * i);
    }
}

func input_helper(ch chan int, num_ints int, rand_ints []int, idx int) {
    for i := 0; i < num_ints; i = i + 1 {
        ch <- rand_ints[idx + i];
    }
    close(ch);
}

/* gets two arrays of input channels containing for prime and
composite.

* the input channels are merged into single outputs for prime and
  composite.

 */
func shuffler(in_prime []chan int, in_composite []chan int, out []
  chan int, num_mappers int, quit []chan bool) {
  var done [num_mappers]chan bool;
  for i := 0; i < num_mappers; i = i + 1 {
    done[i] = make(chan bool);
    yeet shuffler_helper(in_prime[i], in_composite[i], out, done
      [i]);
  }
  for i := 0; i < num_mappers; i = i + 1 {
    <- done[i];
  }
  quit[0] <- true;
  quit[1] <- true;
  close(out[0]);
  close(out[1]);
}

func shuffler_helper(in_prime chan int, in_composite chan int, out
  []chan int, done chan bool) {
  out[0] <- (<- in_prime);
  out[1] <- (<- in_composite);
  done <- true;
}

/* writes out the counts that it receives from each channel. */
func output_writer(in []chan int) {
  prints("number of primes:");
  printi(<- in[0]);
  prints("number of composites:");
  printi(<- in[1]);
}

/* struct for iterative mapreduce */
struct count_t {
  count_prime int;
  count_composite int;
}

/* iterative mapreduce */
func primes_iterative(num_ints int, num_mappers int, rand_ints []int
var counts [num_mappers]struct count_t;
idx := 0;
for i := 0; i < num_mappers; i = i + 1 {
    var rand_ints_slice [num_ints / num_mappers]int;
    for j := 0; j < num_ints / num_mappers; j = j + 1 {
        rand_ints_slice[j] = rand_ints[idx];
        idx = idx + 1;
    }
    counts[i] = count_primes_iterative(num_ints / num_mappers,
                                       rand_ints_slice);
}

count := reduce_counts_iterative(counts, num_mappers);

prints("number of primes: ");
printi(count.count_prime);
prints("number of composites: ");
printi(count.count_composite);

func count_primes_iterative(num_ints int, rand_ints []int) struct
    count_t {
        counts := struct count_t {
            count_prime : 0,
            count_composite: 0
        };

        for i := 0; i < num_ints; i = i + 1 {
            if is_prime(rand_ints[i]) {
                counts.count_prime = counts.count_prime + 1;
            } else {
                counts.count_composite = counts.count_composite + 1;
            }
        }

        return counts;
    }

func reduce_counts_iterative(counts []struct count_t, num_mappers
int) struct count_t {
    count := struct count_t{
        count_prime : 0,
        count_composite: 0
    };

for i := 0; i < num_mappers; i = i + 1 {
    count.count_prime = count.count_prime + counts[i].count_prime;
    count.count_composite = count.count_composite + counts[i].count_composite;
}
return count;
}

/* pregenerate array of num_ints ints */
func generate_ints(num_ints int) []int {
    var rand_ints [num_ints]int;
    for i := 0; i < num_ints; i = i + 1 {
        rand_ints[i] = rand(time());
    }
    return rand_ints;
}

func main() {
    num_ints := 1000000;
    rand_ints := generate_ints(num_ints);
    prints("number of random ints generated:");
    printi(num_ints);

    /* Set up all channels used for passing data between the workers */
    size := 10;
    num_mappers := 10;
    var text [num_mappers]chan int;
    var map_prime [num_mappers]chan int;
    var map_composite [num_mappers]chan int;
    for i := 0; i < num_mappers; i = i + 1 {
        text[i] = make(chan int, size);
        map_prime[i] = make(chan int, size);
        map_composite[i] = make(chan int, size);
    }
    reduce := []chan int{make(chan int, size), make(chan int, size)};
    count := []chan int{make(chan int, size), make(chan int, size)};
    quit := []chan bool{make(chan bool), make(chan bool)};

    /* Start all workers in separate yeetroutines, chained together via channels. */
    yeet input_reader(text, num_mappers, num_ints, rand_ints);
for i := 0; i < num_mappers; i = i + 1 {
    yeet mapper(text[i], map_prime[i], map_composite[i]);
}

yeet shuffler(map_prime, map_composite, reduce, num_mappers, quit);

yeet reducer(reduce[0], count[0], quit[0]);

yeet reducer(reduce[1], count[1], quit[1]);

start := time();
																																																																																																																																																																																																																																																								
return;
} else {
    lock(mu);
    defer unlock(mu);
}

for i := n; i < n + 10; i = i + 1 {
    printi(i);
}
}

func main() {
    mu := make_mutex();
    yeet foo(0, mu);
    yeet foo(25, mu);
    yeet foo(-100, mu);
    foo(-5, mu);
}

9.15.3 producer_consumer.rjec

/* simple producer-consumer problem */
/* written by Justin Chen */

func foo(ch1 chan char, ch2 chan int, quit chan bool) {
    for {
        select {
            case ch1 <- 'a':
                prints("sending a in foo...");
            case val2 := <- ch2:
                prints("receiving in foo:");
                printi(val2);
            case q := <- quit:
                printb(q);
                if q {
                    prints("quitting...");
                    return;
                }
        }
    }
}

func main() {

```go
ch1 := make(chan char);
ch2 := make(chan int);
quit := make(chan bool, 10);
yeet foo(ch1, ch2, quit);
for i := 0; i < 5; i = i + 1 {
    c := <-ch1;
    prints("received in main:");
    printc(c);
    prints("sending i in main...");
    ch2 <- i;
}
quit <- true;

9.15.4  rand_ints.rjec

/* generates 1,000,000 random integers in an array and prints every
10,000th */
/* written by Justin Chen */

func rand(seed int) int {
    return (1103515245 * seed + 12345) % 2147483648;
}

func main() {
    start := time();
    var rand_ints [1000000]int;
    for i := 0; i < 1000000; i = i + 1 {
        rand_ints[i] = rand(time());
        if i % 10000 == 0 {
            printi(rand_ints[i]);
        }
    }
    end := time();
    prints("time taken (us): ");
    printi(end - start);
}

9.16  libmill/

Source code for the Libmill library may be found at https://github.com/sustrik/libmill.

Riya Chakraborty, Justin Chen, Yuanyuting Wang, Caroline Hoang 209