ECE30862 Compiler Project Help

Disclaimer: This is not the only way to implement this project. It is your responsibility to check the syntax. Please do not come and say the code does not compile. This is to guide and help you get started.

In public static void main

// Declare an array of string for input lines

```
ArrayList<String> source = new ArrayList<String>();
```

// string for current line

```
String statement;
```

// Get each line and push it into source array

```
BufferedReader reader;
try{
reader = new BufferedReader(new FileReader(filename));
String line;
while((line = reader.readLine())!= null) {source.add(line);}
}
catch(IOException e){e.printStackTrace();}


bytecode bc = bytecode(source);
ArrayList<byte> destination = new ArrayList<byte>();
bc.compile(destination);
```

// Prepare second argument as binary output

// Save compile result (destination) to output file (.smp)

In bytecode class

class bytecode {

```
public static final int HALT = 0;
public static final int JMP = 36;
public static final int JMPC = 40;
public static final int PUSHI = 70;
public static final int PUSHVI = 74;
public static final int POPM = 76;
public static final int POPA = 77;
public static final int POPV = 80;
public static final int PEEKI = 86;
public static final int POKEI = 90;
public static final int SWP = 94;
public static final int ADD = 100;
public static final int SUB = 104;
public static final int MUL = 108;
public static final int DIV = 112;
public static final int CMPE = 132;
public static final int CMPLT = 136;
public static final int CMPGT = 140;
public static final int PRINTI = 146;
```

```
public static final int LABEL = -1;
public static final int SHORT = 0;
public static final int INT = 1;
public static final int FLOAT = 2;

int sc = 0; // current line being compiled
int pc = -1; // program counter
int fo = -1; // number of local variables in a function

ArrayList<String> source = new ArrayList<String>(); // lines
ArrayList<Integer> mem = new ArrayList<Integer>(); // opcodes
Pair <Integer, Integer> value = new Pair<Integer, Integer>(0,0);
Map<string, value> symbol_table;
ArrayList<ArrayList<Integer>> value = new ArrayList<ArrayList<Integer>>();
Map<string, value> unknown_labels;
```

class bytecode {

```
int decl(String, int);
int lab(string);
int subr(int, string);
int printi(int);
int jmp(string);
int jmpc(string);
int cmpe();
int cmplt();
int cmpgt();
int pushi(int);
int popm(int);
int popa(int);
int popv(string);
int peek(string, int);
int poke(int, string);
int swp();
int add();
int sub();
int mul();
int div();
```

```
int parse(String);
List<String> find_tokens(String);
bool is_alpha(String);
int compile(ArrayList<byte>); // destination from slide 3 is passed in
```

int compile(ArrayList<byte> des);

// Read each line of source code

```
int i = 0;
for (String str : source){
 sc = i + 1;
 parse(str);
 // Check that source code ends with RET
 des = mem; // save mem into des (destination in slide 3 -> compile result saved in .smp file)
}
```

int parse(String statement);

// Skip comments: //

// Find operation and operands in current statement

```
String[] tokens = find_tokens(statement); (return statement.split(" ");)
for(String str: tokens){System.out.println(str + ", ");}
String operation = tokens[0];
Switch(operation)
case("add"): add(parts); break;
… CONTINUE
}
```

Statement that is an operation with no operands

cmpe, cmplt, cmpgt, swp, add, sub, mul, div (tokens.size() == 1)

Statement that is an operation with exactly one operand

lab, printi, jmp, jmpc, pushi, pushvi, popm, popv (tokens.size() == 2)

Statement that is an operation with exactly two operands

decl, subr, peek, poke (tokens.size() == 3)

```
Switch(operation)
case(”pushi"):
int value = tokens[1];
pushi(value);
break;}
```

```
pushi(int value){
Integer[] bytes = Arrays.copyof(value, value.length);
mem.add(PUSHI);
mem.add(bytes[0]);
mem.add(bytes[1]);
mem.add(bytes[2]);
mem.add(bytes[3]);
pc += 5;
}
```

```
Switch(operation){
case(”decl"):
if (tokens[2] == “int”)
{decl(tokens[1], INT);}
break;}
```

```
decl(String var, int type){
String var_name = “main” + “_” + var;
int[] value = {++fo, type};
Symbol_table.put(var_name, value);
// reserve a space in the stack for the variable by generating the following code
pushi(0);
}
```

```
Switch(operation){
case(”lab"):
lab(tokens[1]);
break;}
```

Continue for the other operations ☺