CS220 Homework 07 Josh Clemens

1.Assembler.java

```
import java.io.*;
import java.util.*;

public class Assembler {

        private final SymbolTable symbols; //stores symbols and labels
        private int countLine; //count line
        private Parser parser; //parse into segments

        //constructor
        public Assembler() {
        symbols = new SymbolTable();
        countLine = 0;
        }

        public void preAssemble(String filename) {
        try {
        final BufferedReader input = new BufferedReader(new FileReader(filename));
        boolean parseCheck;
        String line;

        while ((line = input.readLine()) != null) {
                parser = new Parser();
                parseCheck = parser.parseCheck(line);

                if (parseCheck) {
                if (line.trim().charAt(0) == '(') {
                //extract label symbol
                String symbol = line.trim().substring(line.indexOf("(") + 1, line.lastIndexOf(")"));

                if (!symbols.contains(symbol))
                        symbols.put(symbol, countLine);

                countLine--; //dont count labels
                }
                countLine++; //only count true lines
                }
        }
        input.close();
        } catch (final IOException ioe) {
```

```
System.out.println(ioe);
return;
}
}

public void assemble(String filename) {

try {
//change file from .asm to .hack
String outputFilename = filename.substring(0, filename.indexOf(".")) + ".hack";
BufferedReader input = new BufferedReader(new FileReader(filename));
PrintWriter output = new PrintWriter(outputFilename);

countLine = 0;
boolean parseCheck;
String line;

while ((line = input.readLine()) != null) {
        parser = new Parser();
        parseCheck = parser.parseCheck(line);

        if (parseCheck && line.trim().charAt(0) != '(') {
        //C-instruction
        if (parser.bit() == null) {
        String comp = Code.getCompCode(parser.comp());
        String dest = Code.getDestCode(parser.dest());
        String jump = Code.getJumpCode(parser.jump());
        output.printf("111%s%s%s\n", comp, dest, jump);
        }
        //A-instruction
        else {
        String var = parser.bit();

        Scanner sc = new Scanner(var);
        if (sc.hasNextInt()) {
                String bit = Integer.toBinaryString(Integer.parseInt(var));
                //write 16-bit to output
                output.println(checkBit(bit));
        } else {
                symbols.addVariable(var);
                final String bit = Integer.toBinaryString(symbols.get(var));
                output.println(checkBit(bit));
        }
        sc.close();
```

```
                }
                countLine++;
                }
        }
        input.close();
        output.close();
        } catch (final IOException ioe) {
        System.out.println(ioe);
        return;
        }
        }

        //adds 0's as needed
        private String checkBit(String bit) {
        String addZeros = "";
        int zerosNeeded = 16 - bit.length();

        //add needed 0's to complete bit length
        for (int i = 0; i < zerosNeeded; i++) {
        addZeros += "0";
        }

        return addZeros + bit;
        }
}
```

2. Code.java

```
 import java.util.*;

 public class Code {

        private static Hashtable<String, String> destTable = new Hashtable<String, String>(8);
        private static Hashtable<String, String> jumpTable = new Hashtable<String, String>(8);
        private static Hashtable<String, String> compTable = new Hashtable<String, String>(28);

        private static void initDestTable() {
        destTable.put("null", "000");
        destTable.put("M", "001"); //memory
        destTable.put("D", "010"); //D-register
        destTable.put("MD", "011");
        destTable.put("A", "100"); //A-register
        destTable.put("AM", "101");
        destTable.put("AD", "110");
```

```
destTable.put("AMD", "111");
}

private static void initJumpTable() {
jumpTable.put("null", "000");
jumpTable.put("JGT", "001"); //greater than zero
jumpTable.put("JEQ", "010"); //equal to zero
jumpTable.put("JGE", "011"); //greater than or equal to zero
jumpTable.put("JLT", "100"); //less than zero
jumpTable.put("JNE", "101"); //not equal to zero
jumpTable.put("JLE", "110"); //less than or equal to zero
jumpTable.put("JMP", "111"); //unconditional
}

private static void initCompTable() {
compTable.put("0", "0101010");
compTable.put("1", "0111111");
compTable.put("-1", "0111010");
compTable.put("D", "0001100");
compTable.put("A", "0110000");
compTable.put("!D", "0001101");
compTable.put("!A", "0110001");
compTable.put("-D", "0001111");
compTable.put("-A", "0110011");
compTable.put("D+1", "0011111");
compTable.put("A+1", "0110111");
compTable.put("D-1", "0001110");
compTable.put("A-1", "0110010");
compTable.put("D+A", "0000010");
compTable.put("D-A", "0010011");
compTable.put("A-D", "0000111");
compTable.put("D&A", "0000000");
compTable.put("D|A", "0010101");
compTable.put("M", "1110000");
compTable.put("!M", "1110001");
compTable.put("-M", "1110011");
compTable.put("M+1", "1110111");
compTable.put("M-1", "1110010");
compTable.put("D+M", "1000010");
compTable.put("D-M", "1010011");
compTable.put("M-D", "1000111");
compTable.put("D&M", "1000000");
compTable.put("D|M", "1010101");
}
```

```
        public static String getCompCode(String key) {
         initCompTable();
         return compTable.get(key);
  }

        public static String getDestCode(String key) {
    initDestTable();
    return destTable.get(key);
  }

        public static String getJumpCode(String key) {
         initJumpTable();
        return jumpTable.get(key);
        }
}
```

3. Parser.java

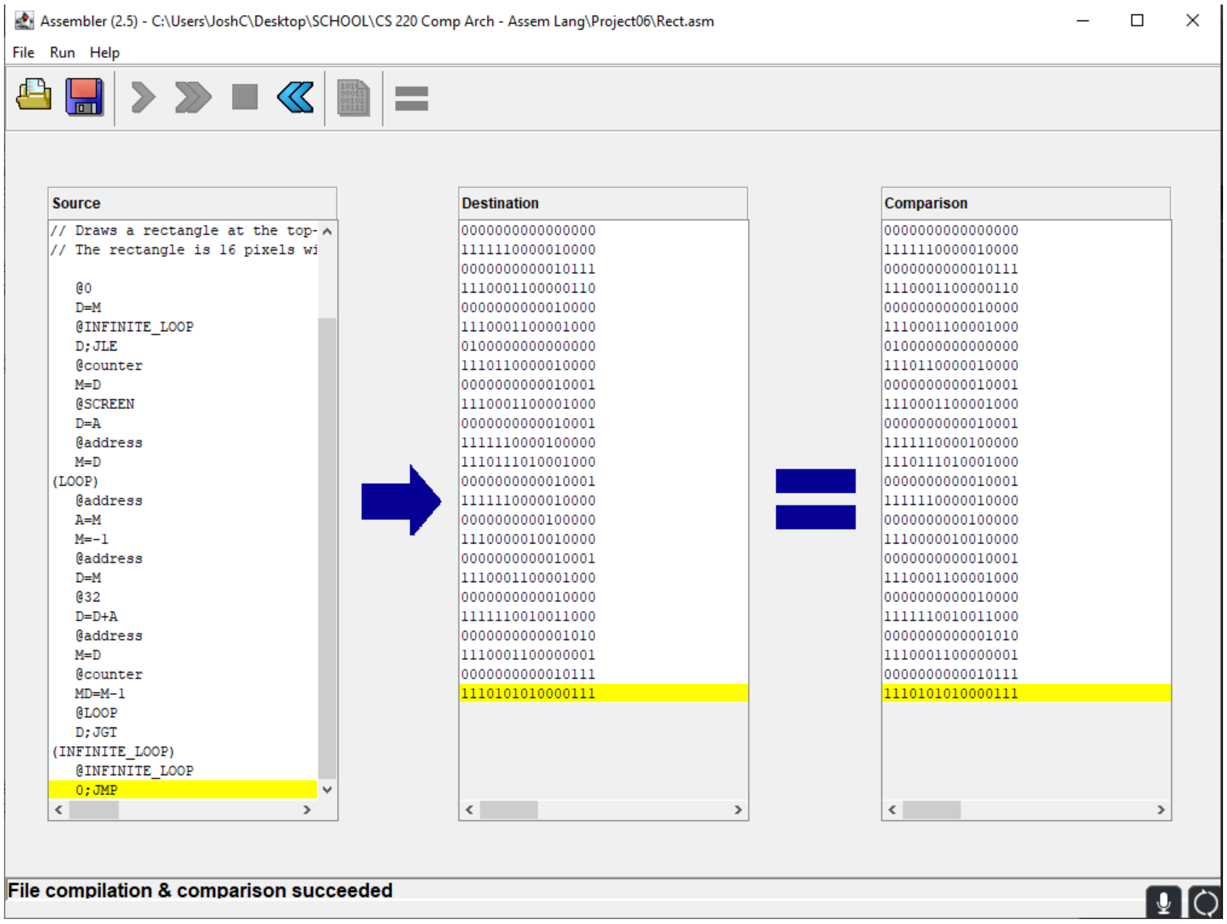
```
public class Parser {
        private String dest; //destination instruction
        private String comp; //computation instruction
        private String jump; //jump instruction
        private String bit; //16-bit address

        //constructor
        public Parser() {
        dest = "null";
        jump = "null";
        }

        public boolean parseCheck(String line) {
        //remove whitespace before and after line
        line = line.trim();

        //validate if line is empty
        if (!line.isEmpty()) {
        //validate if line is a comment
        if (line.charAt(0) != '/') {
                //A-instruction
                if (line.contains("@")) {
                bit = line.split("@")[1].trim();
                }
                //C-instruction
```

```
          else {
          //contains dest, comp or jump instruction
          if (line.contains("=")) {
          String[] segment = line.split("=");
          dest = segment[0];
          //validate jump
          if (segment[1].contains(";")) {
                    jumpCheck(segment[1]);
          } else {
                    //remove comments and whitespace
                    comp = segment[1].split("/")[0].trim();
          }
          } else if (line.contains("+") || line.contains("-")) {
          //validate jump
          if (line.contains(";")) {
                    jumpCheck(line);
          } else {
                    //remove comments and whitespace
                    comp = line.split("/")[0].trim();
          }
          } else if (line.contains(";")) {
          jumpCheck(line);
          } else {
          //remove comments and whitespace
          jump = line.split("/")[0].trim();
          }
          }
          return true;
}
}
return false;
}

private void jumpCheck(String str) {
String[] parts = str.split(";");
comp = parts[0].trim();
jump =  parts[1].split("/")[0].trim();
}

public String dest() {
return dest;
}

public String comp() {
        return comp;
        }

        public String jump() {
        return jump;
        }

        public String bit() {
        return bit;
        }
}
```

4. SymbolTable.java

```
import java.util.Hashtable;

public class SymbolTable {

        private int countRegister;
        private final Hashtable<String, Integer> symbolTable;

        //constructor
        public SymbolTable() {
        countRegister = 16;
        symbolTable = new Hashtable<String, Integer>(25);

        //initialize pre-defined variables
        for (int i = 0; i <= 15; i++) {
        final String key = "R" + i;
        symbolTable.put(key, i);
        }

        symbolTable.put("SCREEN", 16384);
        symbolTable.put("KBD", 24576);
        symbolTable.put("SP", 0);
        symbolTable.put("LCL", 1);
        symbolTable.put("ARG", 2);
        symbolTable.put("THIS", 3);
        symbolTable.put("THAT", 4);
        }

        public boolean addVariable(final String symbol) {
        if (!symbolTable.containsKey(symbol)) {
        symbolTable.put(symbol, countRegister);
        countRegister++;
        return true;
        }
        return false;
        }

        public void put(final String symbol, final int value) {
        symbolTable.put(symbol, value);
        }

        public boolean contains(final String symbol) {
        return symbolTable.containsKey(symbol);
        }

        public int get(final String symbol) {
        return symbolTable.get(symbol);
        }
}
```

5. Screenshot of Rect.hack comparison


Assembler (2.5) - C:\Users\JoshC\Desktop\SCHOOL\CS 220 Comp Arch - Assem Lang\Project06\Rect.asm
File
Run
Help

Source

```
// Draws a rectangle at the top-
// The rectangle is 16 pixels wi

   @0
   D=M
   @INFINITE_LOOP
   D;JLE
   @counter
   M=D
   @SCREEN
   D=A
   @address
   M=D
(LOOP)
   @address
   A=M
   M=-1
   @address
   D=M
   @32
   D=D+A
   @address
   M=D
   @counter
   MD=M-1
   @LOOP
   D;JGT
(INFINITE_LOOP)
   @INFINITE_LOOP
   0;JMP
```

Destination

```
0000000000000000
1111110000010000
0000000000010111
1110001100000110
0000000000010000
1110001100001000
0100000000000000
1110110000010000
0000000000010001
1110001100001000
0000000000010001
1111110000100000
1110111010001000
0000000000010001
1111110000010000
0000000000100000
1110000010010000
0000000000010001
1110001100001000
0000000000010000
1111110010011000
0000000000001010
1110001100000001
0000000000010111
1110101010000111
```

Comparison

```
0000000000000000
1111110000010000
0000000000010111
1110001100000110
0000000000010000
1110001100001000
0100000000000000
1110110000010000
0000000000010001
1110001100001000
0000000000010001
1111110000100000
1110111010001000
0000000000010001
1111110000010000
0000000000100000
1110000010010000
0000000000010001
1110001100001000
0000000000010000
1111110010011000
0000000000001010
1110001100000001
0000000000010111
1110101010000111
```

File compilation & comparison succeeded