

```

1
2 import java.io.BufferedReader;
3 import java.io.BufferedWriter;
4 import java.io.FileReader;
5 import java.io.FileWriter;
6 import java.io.IOException;
7 import java.util.ArrayList;
8 import java.util.LinkedHashMap;
9
10 public class PassOne {
11     int lc=0;
12     int libtab_ptr=0, pooltab_ptr=0;
13     int symIndex=0, litIndex=0;
14     LinkedHashMap<String, TableRow> SYMTAB;
15     ArrayList<TableRow> LITTAB;
16     ArrayList<Integer> POOLTAB;
17     private BufferedReader br;
18
19     public PassOne()
20     {
21         SYMTAB =new LinkedHashMap<>();
22         LITTAB=new ArrayList<>();
23         POOLTAB=new ArrayList<>();
24         lc=0;
25         POOLTAB.add(0);
26     }
27     public static void main(String[] args) {
28         PassOne one=new PassOne();
29         try
30         {
31             one.parseFile();
32         }
33         catch (Exception e) {
34             System.out.println("Error: "+e); //
35         }
36     }
37     public void parseFile() throws Exception
38     {
39         String prev="";
40         String line, code;
41         br = new BufferedReader(new FileReader("sample.asm"));
42         BufferedWriter bw=new BufferedWriter(new FileWriter("IC.txt"));
43         INSTtable lookup=new INSTtable();
44         while((line=br.readLine())!=null)
45         {
46             String parts[]=line.split("\\s+");
47             if(!parts[0].isEmpty()) //processing of label
48             {
49                 if(SYMTAB.containsKey(parts[0]))
50                     SYMTAB.put(parts[0], new TableRow(parts[0], lc, SYMTAB.get(parts[0]).getSymbol());
51                 else
52                     SYMTAB.put(parts[0], new TableRow(parts[0], lc, ++symIndex));
53             }
54             if(parts[1].equals("LTORG"))
55             {
56                 int ptr=POOLTAB.get(pooltab_ptr);
57                 for(int j=ptr; j<libtab_ptr; j++)
58                 {
59                     lc++;
60                     LITTAB.set(j, new TableRow(LITTAB.get(j).getSymbol(), lc));
61                     code="(DL,01)\\t(C,"+LITTAB.get(j).symbol+")";
62                     bw.write(code+"\\n");
63                 }
64                 pooltab_ptr++;
65                 POOLTAB.add(libtab_ptr);
66             }
67             if(parts[1].equals("START"))
68             {
69                 lc=expr(parts[2]);
70                 code="(AD,01)\\t(C,"+lc+")";
71                 bw.write(code+"\\n");
72                 prev="START";
73             }
74             else if(parts[1].equals("ORIGIN"))
75             {
76                 lc=expr(parts[2]);
77                 String splits[]=parts[2].split("\\s+"); //Same for - SYMBOL
78                 code="(AD,03)\\t(S,"+SYMTAB.get(splits[0]).getIndex()+")+";
79                 bw.write(code+"\\n");
80             }
81             //Now for EQU
82             if(parts[1].equals("EQU"))
83             {
84                 int loc=expr(parts[2]);
85                 //below If conditions are optional as no IC is generated for EQU
86                 if(parts[2].contains("+"))
87                 {
88                     String splits[]=parts[2].split("\\s+");
89                     code="(AD,04)\\t(S,"+SYMTAB.get(splits[0]).getIndex()+")+";
90                     bw.write(code+"\\n");
91                 }
92                 else if(parts[2].contains("-"))
93                 {
94                     String splits[]=parts[2].split("\\s-");
95                     code="(AD,04)\\t(S,"+SYMTAB.get(splits[0]).getIndex()+")+";
96                     bw.write(code+"\\n");
97                 }
98                 else
99                 {
100                     code="(AD,04)\\t(C,"+Integer.parseInt(parts[2])+")";
101                     bw.write(code+"\\n");
102                 }
103                 if(SYMTAB.containsKey(parts[0]))
104                     SYMTAB.put(parts[0], new TableRow(parts[0], loc, SYMTAB.get(parts[0]).getSymbol());
105             }
106         }
107     }
108 }

```

```

104         if (SYMTAB.containsKey(parts[0]))
105             SYMTAB.put(parts[0], new TableRow(parts[0], loc, SYMTAB.get(parts[0]).getAddress()));
106         else
107             SYMTAB.put(parts[0], new TableRow(parts[0], loc, ++symIndex));
108     }
109
110     if (parts[1].equals("DC"))
111     {
112         lc++;
113         int constant = Integer.parseInt(parts[2].replace("'", ""));
114         code = "(DL,01)\t(C," + constant + ")";
115         bw.write(code + "\n");
116     }
117     else if (parts[1].equals("DS"))
118     {
119
120         int size = Integer.parseInt(parts[2].replace("'", ""));
121
122         code = "(DL,02)\t(C," + size + ")";
123         bw.write(code + "\n");
124         /*if (prev.equals("START"))
125         {
126             lc = lc + size - 1; // System.out.println("here");
127
128         }
129         else
130             lc = lc + size;
131         prev = "";
132     }
133     if (lookup.getType(parts[1]).equals("IS"))
134     {
135         code = "(IS,0" + lookup.getCode(parts[1]) + ")\t";
136         int j = 2;
137         String code2 = "";
138         while (j < parts.length)
139         {
140             parts[j] = parts[j].replace(",", "");
141             if (lookup.getType(parts[j]).equals("RG"))
142             {
143                 code2 += lookup.getCode(parts[j]) + "\t";
144             }
145             else
146             {
147                 if (parts[j].contains("="))
148                 {
149                     parts[j] = parts[j].replace("=", "").replace("'", "");
150                     LITTAB.add(new TableRow(parts[j], -1, ++litIndex));
151                     code2 += "(L," + (litIndex) + ")";
152                 }
153                 else if (SYMTAB.containsKey(parts[j]))
154                 {
155                     int ind = SYMTAB.get(parts[j]).getIndex();
156                     code2 += "(S,0" + ind + ")";
157                 }
158                 else
159                 {
160                     SYMTAB.put(parts[j], new TableRow(parts[j], -1, ++symIndex));
161                     int ind = SYMTAB.get(parts[j]).getIndex();
162                     code2 += "(S,0" + ind + ")";
163                 }
164             }
165             j++;
166         }
167         lc++;
168         code = code + code2;
169         bw.write(code + "\n");
170     }
171 }
172
173 if (parts[1].equals("END"))
174 {
175     int ptr = POOLTAB.get(pooltab_ptr);
176     for (int j = ptr; j < libtab_ptr; j++)
177     {
178         lc++;
179         LITTAB.set(j, new TableRow(LITTAB.get(j).getSymbol(), lc, SYMTAB.get(parts[0]).getAddress()));
180         code = "(DL,01)\t(C," + LITTAB.get(j).symbol + ")";
181         bw.write(code + "\n");
182     }
183     pooltab_ptr++;
184     POOLTAB.add(libtab_ptr);
185     code = "(AD,02)";
186     bw.write(code + "\n");
187 }
188
189 }
190 bw.close();
191 printSYMTAB();
192 //Printing Literal table
193 PrintLITTAB();
194 printPOOLTAB();
195 }
196 void PrintLITTAB() throws IOException
197 {
198     BufferedWriter bw = new BufferedWriter(new FileWriter("LITTAB.txt"));
199     System.out.println("\nLiteral Table\n");
200     //Processing LITTAB
201     for (int i = 0; i < LITTAB.size(); i++)
202     {
203         TableRow row = LITTAB.get(i);
204         System.out.println(i + "\t" + row.getSymbol() + "\t" + row.getAddress());
205         bw.write((i + 1) + "\t" + row.getSymbol() + "\t" + row.getAddress() + "\n");
206     }
207     bw.close();
208 }

```



```

200     }
201     bw.close();
202 }
203 void printPOOLTAB() throws IOException
204 {
205     BufferedWriter bw=new BufferedWriter(new FileWriter("POOLTAB.txt"));
206     System.out.println("\nPOOLTAB");
207     System.out.println("Index\t#first");
208     for (int i = 0; i < POOLTAB.size(); i++) {
209         System.out.println(i+"\t"+POOLTAB.get(i));
210         bw.write((i+1)+"\t"+POOLTAB.get(i)+"\n");
211     }
212     bw.close();
213 }
214 void printSYMTAB() throws IOException
215 {
216     BufferedWriter bw=new BufferedWriter(new FileWriter("SYMTAB.txt"));
217     //Printing Symbol Table
218     java.util.Iterator<String> iterator = SYMTAB.keySet().iterator();
219     System.out.println("SYMBOL TABLE");
220     while (iterator.hasNext()) {
221         String key = iterator.next().toString();
222         TableRow value = SYMTAB.get(key);
223
224         System.out.println(value.getIndex()+"\t" + value.getSymbol()+"\n");
225         bw.write(value.getIndex()+"\t" + value.getSymbol()+"\t"+value.get
226     }
227     bw.close();
228 }
229 public int expr(String str)
230 {
231     int temp=0;
232     if(str.contains("+"))
233     {
234         String splits[]=str.split("\\+");
235         temp=SYMTAB.get(splits[0]).getIndex()+Integer.parseInt(splits[1]);
236     }
237     else if(str.contains("-"))
238     {
239         String splits[]=str.split("\\-");
240         temp=SYMTAB.get(splits[0]).getIndex()-(Integer.parseInt(splits[1]));
241     }
242     else
243     {
244         temp=Integer.parseInt(str);
245     }
246     return temp;
247 }
248 }
249 }

```

```

student@student-ASUS-EXPERTCENTER-D500MD-D500MD-IN: ~/Desktop/LP1
(base) student@student-ASUS-EXPERTCENTER-D500MD-D500MD-IN:~/Desktop/LP1$ javac PassOne.java
(base) student@student-ASUS-EXPERTCENTER-D500MD-D500MD-IN:~/Desktop/LP1$ java PassOne
SYMBOL TABLE
1      A      100
2      L1     103
3      B      108
4      C      109
5      D      101
6      L2     106

Literal Table

POOLTAB
Index  #first
0      0
1      0
(base) student@student-ASUS-EXPERTCENTER-D500MD-D500MD-IN:~/Desktop/LP1$

```

