受理号: _____ 受理签字: _____

登记号: _____ 审查签字: _____

流水号	**********

计算机软件著作权登记申请表

软件基本信息	软件全称	Hollow Jack语言编译软件	版本号	V1.0
	软件简称		分类号	10300-6200
	软件作品说明	⊙ 原创 ○ 修改(含翻译软件、合成软件) ☐ 修改软件须经原权利人授权 ☐ 原有软件已经登记 • 原登记号: • 修改（翻译或合成）软件作品说明:		
开发完成日期		2020 年 05 月 04 日		
发表状态		○ 已发表 ⊙ 未发表		
开发方式		⊙ 独立开发 ○ 合作开发 ○ 委托开发 ○ 下达任务开发		

著作权人	姓名或名称	类别	证件类型	证件号码	国籍	省份/城市	园区
	Hollow Man	自然人	居民身份证	**********	中国	**********	

流水号	**********

权利说明	权利取得方式	⊙ 原始取得 ○ 继受取得（ ○ 受让 ○ 承受 ○ 继承 ） ▬ 该软件已登记 （原登记号：__） □ 原登记做过变更或补充 （变更或补充证明编号：__）		
	权利范围	⊙ 全部 ○ 部分（ □ 发表权 □ 署名权 □ 修改权 □ 复制权 □ 发行权 □ 出租权 □ 信息网络传播权 □ 翻译权 □ 应当由著作权人享有的其他权利 ）		
软件鉴别材料	⊙ 一般交存	提交源程序前连续的30页和后连续的30页； 提交任何一种文档的前连续的30页和后连续的30页； ⊙ 一种文档 ○ ____种文档		
	○ 例外交存	○使用黑色宽斜线覆盖，页码为： ○前10页和任选连续的50页 ○目标程序的连续的前、后各30页和源程序任选连续的20页		
软件功能和技术特点	硬件环境	CPU: Intel Core i7-8550U 1.80GHZ， RAM: 8GB，硬盘: SSD 128GB 机械 1TB		
	软件环境	Ubuntu 18.04, Python 3.8		
	编程语言	Python 3	源程序量	1998
	主要功能和技术特点	Jack语言是《计算机系统要素：从零开始构建现代计算机》书中所描述的一种面向对象编程语言。此软件可以自动遍历各个文件夹下的Jack源代码文件（.jack），并通过词法分析，语法分析，语义分析步骤检查其中的错误，将Jack语言转换为Jack虚拟机所使用的汇编语言（.vm）。		

流水号	**********

申请办理方式		⊙由著作权人申请　○由代理人申请		
申请人信息	姓名或名称	Hollow Man	电话	**********
	详细地址	**********	邮编	*******
	联系人	Hollow Man	手机	**********
	E-mail	**********	传真	
代理人信息	申请人委托下述代理人办理登记事宜，具体委托事项如下：			
	姓名或名称		电话	
	详细地址		邮编	
	联系人		手机	
	E-mail		传真	

申请人认真阅读了填表说明，准确理解了所需填写的内容，保证所填写的内容真实。

申请人签章：

2020 年 05 月 25 日

流水号	**********

证书份数	1份正本	
请确认所需要的计算机软件著作权登记证书副本份数。登记证书正本和副本数量之和不能超过软件著作权人的数量。		
提交申请材料清单		
申请材料类型	申请材料名称	
申请表	打印签字或盖章的登记申请表	一份 4 页
软件鉴别材料	软件源程序	一份 48 页
	软件文档(1)	一份 13 页
	软件文档(2)	一份____页
身份证明文件	申请人身份证明复印件	一份 1 页
	代理人身份证明复印件	一份____页
权利归属证明文件	软件转让合同或协议	一份____页
	承受或继承证明文件	一份____页
其他材料		一份____页
		一份____页
		一份____页
		一份____页

填写说明：

请按照提示要求提交有关申请材料，并在提交申请材料清单中准确填写实际交存材料页数。若提示中没有的，请填写材料名称及其页数。该页是申请表的组成部分与申请表一并打印提交。

```
01 # myjc.py
02
03 import sys
04 import os
05 import lexer
06 import jcparser
07 import copy
08 import SymbolTable
09
10 if len(sys.argv) == 1:
11     print("Error, no input directory")
12 elif len(sys.argv) > 2:
13     print("Error, too many arguments, please only type your JACK progr
am directory")
14 else:
15     if not os.path.exists(sys.argv[1]):
16         print("Error, please make sure your typed JACK program directo
ry exists")
17     elif os.path.isfile(sys.argv[1]):
18         print("Error, please type your JACK program directory instead
of file name")
19     else:
20         filename = os.listdir()
21         count = 0
22         systab={}
23         for fpathe, dirs, fs in os.walk("syslib"):
24             for f in fs:
25                 file = os.path.join(fpathe, f)
26                 if os.path.splitext(f)[1] == ".jack":
27                     source = open(file)
28                     sourcecode = []
29                     temp = source.readline()
30                     while temp:
31                         sourcecode.append(temp.replace("\n", ""))
32                         temp = source.readline()
33                     source.close()
34                     tokens = lexer.Token(sourcecode)
35                     try:
36                         tokens.line = 0
37                         tokens.pointer = 0
38                         systab=SymbolTable.start(tokens,systab)
39                     except Exception as err:
40                         print(file,end="")
41                         print(err.args[0]+" error, line "+str(err.args
[1]) +
42                                 ', close to "'+str(err.args[2])+'", '+
str(err.args[3]))
43             for f in fs:
44                 file = os.path.join(fpathe, f)
45                 if os.path.splitext(f)[1] == ".jack":
```

```
46                      source = open(file)
47                      sourcecode = []
48                      temp = source.readline()
49                      while temp:
50                          sourcecode.append(temp.replace("\n", ""))
51                          temp = source.readline()
52                      source.close()
53                      print("\nCompiling System library: "+file)
54                      tokens = lexer.Token(sourcecode)
55                      try:
56                          tokens.line = 0
57                          tokens.pointer = 0
58                          systab=jcparser.start(tokens,systab,True)
59                          print("Compilation success")
60                      except Exception as err:
61                          try:
62                              print(err.args[0]+" error, line "+str(err.
args[1]) +
63                                  ', close to "'+str(err.args[2])+'", '+
str(err.args[3]))
64                          except Exception:
65                              print("Unkown Error")
66          for fpathe, dirs, fs in os.walk(sys.argv[1]):
67              stab=copy.deepcopy(systab)
68              for f in fs:
69                  file = os.path.join(fpathe, f)
70                  if os.path.splitext(f)[1] == ".jack":
71                      source = open(file)
72                      sourcecode = []
73                      temp = source.readline()
74                      while temp:
75                          sourcecode.append(temp.replace("\n", ""))
76                          temp = source.readline()
77                      source.close()
78                      tokens = lexer.Token(sourcecode)
79                      try:
80                          tokens.line = 0
81                          tokens.pointer = 0
82                          stab=SymbolTable.start(tokens,stab)
83                      except Exception as err:
84                          print(file,end="")
85                          print(err.args[0]+" error, line "+str(err.args
[1]) +
86                                  ', close to "'+str(err.args[2])+'", '+
str(err.args[3]))
87              for f in fs:
88                  file = os.path.join(fpathe, f)
89                  if os.path.splitext(f)[1] == ".jack":
90                      count += 1
91                      source = open(file)
```

```
92                      sourcecode = []
93                      temp = source.readline()
94                      while temp:
95                          sourcecode.append(temp.replace("\n", ""))
96                          temp = source.readline()
97                      source.close()
98                      print("\nCompiling "+file)
99                      tokens = lexer.Token(sourcecode)
100                      try:
101                          destcode = ""
102                          tokens.line = 0
103                          tokens.pointer = 0
104                          destcode,stab = jcparser.start(tokens,stab)
105                          print("Compilation success")
106                          with open(os.path.join(fpathe, os.path.splite
xt(f)[0]+".vm"), 'w') as dest:
107                              dest.write(destcode)
108                      except Exception as err:
109                          try:
110                              print(err.args[0]+" error, line "+str(err
.args[1]) +
111                                  ', close to "'+str(err.args[2])+'", '
+str(err.args[3]))
112                          except Exception:
113                              print("Unkown Error")
114          if count == 0:
115              print("Error, unable to find any jack source files in you
r typed JACK program directory, "
116                    "please make sure your source code files have .jack
 extension")
117          else:
118              print("\nCompilation Complete! Proceed " +
119                    str(count)+" files in total.")
120
121 # lexer.py
122
123 class Token:
124     keywords = ["class", "constructor", "method", "function", "let",
"do", "if",
125                 "int", "boolean", "char", "void", "var", "static", "f
ield",
126                 "else", "while", "return", "this"]
127     # I take "true", "false", "null" as a specific type
128     types = ["Identifier", "Integer", "String", "Boolean",
129              "Null", "Symbol", "Keyword", "Operator", "Method", "EOF"
]
130     operators = ["+", "-", "*", "/", "&", "|", "~", "<", ">"]
131     symbols = ["{", "}", "[", "]", "(", ")", ",", ";", "=", "."]
132     line = 0
133     pointer = 0
```

```
134     code = []
135 
136     def __init__(self, code):
137         self.code = code
138 
139     def GetNextToken(self):
140         lexem = ""
141         # code is empty or pointer exceeds the line size
142         if self.code == []:
143             return (None, "EOF")
144         while True:
145             if self.line >= len(self.code):
146                 return (None, "EOF")
147             if self.pointer >= len(self.code[self.line]):
148                 self.pointer = 0
149                 self.line += 1
150             else:
151                 break
152         # Consuming beginning tab and whitespace and check then consu
me comments
153         while True:
154             while self.code[self.line][self.pointer] == ' ' or self.c
ode[self.line][self.pointer] == '\t':
155                 self.pointer += 1
156                 # make sure haven't reached the end of line or file
157                 while True:
158                     if self.line >= len(self.code):
159                         return (None, "EOF")
160                     if self.pointer >= len(self.code[self.line]):
161                         self.pointer = 0
162                         self.line += 1
163                     else:
164                         break
165             while self.pointer+1 < len(self.code[self.line]) and self
.code[self.line][self.pointer] == '/':
166                 if self.code[self.line][self.pointer+1] == '/':
167                     self.line += 1
168                     self.pointer = 0
169                     if self.line >= len(self.code):
170                         return (None, "EOF")
171                 elif self.code[self.line][self.pointer+1] == '*':
172                     self.pointer += 2
173                     # make sure haven't reached the end of line or fi
le
174                     while True:
175                         if self.pointer+1 < len(self.code[self.line])
 and self.code[self.line][self.pointer] == '*':
176                             if self.code[self.line][self.pointer+1] =
= '/':
177                                 self.pointer += 2
```

```
178                                 break
179                         self.pointer += 1
180                         # make sure haven't reached the end of line o
r file
181                         while True:
182                             if self.line >= len(self.code):
183                                 raise Exception("Lexical", self.line+
1,
184                                                 "EOF", 'Comments end
without */')
185                             if self.pointer >= len(self.code[self.lin
e]):
186                                 self.pointer = 0
187                                 self.line += 1
188                             else:
189                                 break
190                 # it's divide symbol
191                 else:
192                     self.pointer += 1
193                     return ("/", "Operator")
194                 # make sure haven't reached the end of line or file
195                 while True:
196                     if self.line >= len(self.code):
197                         return (None, "EOF")
198                     if self.pointer >= len(self.code[self.line]):
199                         self.pointer = 0
200                         self.line += 1
201                     else:
202                         break
203             # make sure haven't reached the end of line or file
204             while True:
205                 if self.line >= len(self.code):
206                     return (None, "EOF")
207                 if self.pointer >= len(self.code[self.line]):
208                     self.pointer = 0
209                     self.line += 1
210                 else:
211                     break
212             # To ensure that no remained tab and whitespace and comme
nts
213             if not(self.code[self.line][self.pointer] == ' ' or self.
code[self.line][self.pointer] == '\t' or self.code[self.line][self.pointe
r] == '\\'):
214                 break
215         # begin getting a Token
216         # String
217         if self.pointer < len(self.code[self.line]) and self.code[sel
f.line][self.pointer] == '"':
218             self.pointer += 1
219             while self.pointer < len(self.code[self.line]):
```

```
220                 if self.code[self.line][self.pointer] == '"':
221                     self.pointer += 1
222                     if self.pointer >= len(self.code[self.line]):
223                         self.pointer = 0
224                         self.line += 1
225                     return (lexem, "String")
226                 else:
227                     lexem += self.code[self.line][self.pointer]
228                     self.pointer += 1
229             # Check if the string ends with "
230             raise Exception("Lexical", self.line+1,
231                             self.code[self.line][self.pointer-
1], 'a string ended without "')
232         # Others
233         while self.pointer < len(self.code[self.line]) and self.code[
self.line][self.pointer] != ' ':
234             # raise error if there exists special symbols
235             if not (self.code[self.line][self.pointer].isalnum() or s
elf.code[self.line][self.pointer] == '_' or self.code[self.line][self.poi
nter] in self.operators or self.code[self.line][self.pointer] in self.sym
bols):
236                 raise Exception("Lexical", self.line+1,
237                                 self.code[self.line][self.pointer], '
unrecognised symbol')
238             lexem += self.code[self.line][self.pointer]
239             # To identify symbols
240             if self.code[self.line][self.pointer] in self.operators:
241                 self.pointer += 1
242                 return (lexem, "Operator")
243             elif self.code[self.line][self.pointer] in self.symbols:
244                 self.pointer += 1
245                 return (lexem, "Symbol")
246             # To cut symbols with words
247             elif self.pointer+1 < len(self.code[self.line]) and not (
self.code[self.line][self.pointer+1].isalnum() or self.code[self.line][se
lf.pointer+1] == '_'):
248                 self.pointer += 1
249                 break
250             self.pointer += 1
251         # Reconize the type of lexem
252         if lexem in self.keywords:
253             return (lexem, "Keyword")
254         elif lexem == "true" or lexem == "false":
255             return (lexem, "Boolean")
256         elif lexem == "null":
257             return (lexem, "Null")
258         elif lexem.isnumeric():
259             return (int(lexem), "Integer")
260         elif lexem[0].isalpha() or lexem[0] == "_":
261             return (lexem, "Identifier")
```

```
262         # Error when don't match any type of lexem
263         else:
264             raise Exception("Lexical", self.line+1,
265                             self.code[self.line][self.pointer-
1], 'wrong identifier')
266 
267     def PeekNextToken(self):
268         oldline = self.line
269         oldpointer = self.pointer
270         token = self.GetNextToken()
271         self.line = oldline
272         self.pointer = oldpointer
273         return token
274 
275 # SymbolTable.py
276 
277 class SymbolTable:
278     table = {}
279     level = []
280 
281     def Add(self, name, dtype, kind, assign, new, firstt,offset=0):
282         result = self.Find(name)
283         if new:
284             if result:
285                 if firstt:
286                     return False
287         symbol = {}
288         info = []
289         info.append(dtype)
290         info.append(kind)
291         info.append(assign)
292         if not new:
293             if len(result) == 2:
294                 info.append(result[0][-1])
295                 symbol = self.table[str(self.level[:-result[1]])]
296                 symbol[name] = info[:]
297                 self.table[str(self.level[:-result[1]])] = symbol
298                 return True
299             else:
300                 info.append(result[-1])
301         else:
302             info.append(offset)
303         if str(self.level) in self.table:
304             symbol = self.table[str(self.level)]
305         symbol[name] = info[:]
306         self.table[str(self.level)] = symbol
307         return True
308 
309     def Find(self, name, deep=True):
310         if str(self.level) in self.table:
```

```
311             if name in self.table[str(self.level)]:
312                 return self.table[str(self.level)][name]
313         if deep:
314             loop = 1
315             while loop < len(self.level):
316                 if str(self.level[:-loop]) in self.table:
317                     if name in self.table[str(self.level[:-loop])]:
318                         return self.table[str(self.level[:-
loop])][name], loop
319                 loop += 1
320             return False
321
322 global symboltable
323 symboltable = SymbolTable()
324 ifnum=0
325 whilenum=0
326
327 def start(token, table):
328     global symboltable
329     symboltable.table = table
330     symboltable.level = []
331     count = 0
332     while True:
333         nextToken = token.PeekNextToken()
334         if nextToken[0] == "class":
335             classDeclar(token,count)
336             count += 1
337         elif nextToken[1] == "EOF":
338             break
339         else:
340             # Check whether there is code outside the class block
341             raise Exception('Semantic', token.line+1,
342                             token.code[token.line][token.pointer-
1], "unreachable code outside the class block")
343     return symboltable.table
344

345 def classDeclar(token, count):
346     nextToken = token.GetNextToken()
347     if nextToken[0] == "class":
348         nextToken = token.GetNextToken()
349         if nextToken[1] == "Identifier":
350             symboltable.level.append(nextToken[0])
351             if not symboltable.Add(nextToken[0], nextToken[0], "class
", False, True,True, count):
352                 raise Exception(
353                     'Semantic', token.line+1, token.code[token.line][
token.pointer-1], nextToken[0]+"class has declared")
354             nextToken = token.GetNextToken()
355             if nextToken[0] == "{":
```

```
356                 nextToken = token.PeekNextToken()
357                 counts = 0
358                 countf = 0
359                 while True:
360                     if nextToken[0] == 'static':
361                         classVarDeclar(token, counts)
362                         counts += 1
363                     elif nextToken[0] == 'field':
364                         classVarDeclar(token, countf)
365                         countf += 1
366                     else:
367                         break
368                     nextToken = token.PeekNextToken()
369                 countc = 0
370                 countf = 0
371                 countm = 0
372                 while True:
373                     if nextToken[0] == 'constructor':
374                         countc = subroutineDeclar(token, countc)
375                         countc += 1
376                     elif nextToken[0] == 'function':
377                         countf = subroutineDeclar(token, countf)
378                         countf += 1
379                     elif nextToken[0] == 'method':
380                         countm = subroutineDeclar(token, countm)
381                         countm += 1
382                     else:
383                         break
384                     nextToken = token.PeekNextToken()
385                 nextToken = token.GetNextToken()
386                 if nextToken[0] == "}":
387                     del symboltable.level[-1]
388                 else:
389                     raise Exception(
390                         'Syntax', token.line+1, token.code[token.line
][token.pointer-1], "'}' expected at this area")
391             else:
392                 raise Exception(
393                     'Syntax', token.line+1, token.code[token.line][to
ken.pointer-1], "'{' expected at this area")
394         else:
395             raise Exception(
396                 'Syntax', token.line+1, token.code[token.line][token.
pointer-1], "an identifier expected at this area")
397     else:
398         raise Exception('Syntax', token.line+1,
399                         token.code[token.line][token.pointer-
1], "'class' expected at this area")
400
```

```
401 def classVarDeclar(token, count):
402     nextToken = token.GetNextToken()
403     if nextToken[0] == 'static' or nextToken[0] == 'field':
404         kind = nextToken[0]
405         nextToken = token.GetNextToken()
406         if nextToken[0] == "void" or nextToken[0] == "int" or nextTok
en[0] == "char" or nextToken[0] == "boolean" or nextToken[1] == "Identifi
er":
407             type = nextToken[0]
408             nextToken = token.GetNextToken()
409             if nextToken[1] == "Identifier":
410                 if not symboltable.Add(nextToken[0], type, kind, Fals
e, True,True, count):
411                     raise Exception(
412                         'Semantic', token.line+1, token.code[token.li
ne][token.pointer-1], nextToken[0]+" has declared in this class")
413                 nextToken = token.GetNextToken()
414                 while nextToken[0] == ',':
415                     nextToken = token.GetNextToken()
416                     if nextToken[1] == "Identifier":
417                         count += 1
418                         if not symboltable.Add(nextToken[0], type, ki
nd, False, True,True, count):
419                             raise Exception(
420                                 'Semantic', token.line+1, token.code[
token.line][token.pointer-1], nextToken[0]+" has declared in this class")
421                     else:
422                         raise Exception(
423                             'Syntax', token.line+1, token.code[token.
line][token.pointer-1], "an identifier expected at this area")
424                     nextToken = token.GetNextToken()
425                 if nextToken[0] == ";":
426                     return count
427                 else:
428                     raise Exception(
429                         'Syntax', token.line+1, token.code[token.line
][token.pointer-1], "';' expected at this area")
430             else:
431                 raise Exception(
432                     'Syntax', token.line+1, token.code[token.line][to
ken.pointer-1], "an identifier expected at this area")
433         else:
434             raise Exception(
435                 'Syntax', token.line+1, token.code[token.line][token.
pointer-1], "expect a type at this area")
436     else:
437         raise Exception('Syntax', token.line+1,
438                         token.code[token.line][token.pointer-
1], "'static' or 'field' expected at this area")
439
```

```
440 def subroutineDeclar(token, count):
441     nextToken = token.GetNextToken()
442     if nextToken[0] == 'constructor' or nextToken[0] == 'function' or
 nextToken[0] == 'method':
443         kind = nextToken[0]
444         nextToken = token.GetNextToken()
445         if nextToken[0] == "void" or nextToken[0] == "int" or nextTok
en[0] == "char" or nextToken[0] == "boolean" or nextToken[1] == "Identifi
er":
446             type = nextToken[0]
447             nextToken = token.GetNextToken()
448             if nextToken[1] == "Identifier":
449                 if not symboltable.Add(nextToken[0], type, kind, Fals
e, True,True, count):
450                     raise Exception(
451                         'Semantic', token.line+1, token.code[token.li
ne][token.pointer-1], nextToken[0]+" has declared in this class")
452                 symboltable.level.append(nextToken[0])
453                 nextToken = token.GetNextToken()
454                 if nextToken[0] == "(":
455                     paramList(token)
456                     nextToken = token.GetNextToken()
457                     if nextToken[0] == ")":
458                         nextToken = token.GetNextToken()
459                         if nextToken[0] == "{":
460                             while statementTest(token):
461                                 statement(token)
462                             nextToken = token.GetNextToken()
463                             if nextToken[0] == "}":
464                                 del symboltable.level[-1]
465                                 return count
466                             else:
467                                 raise Exception(
468                                     'Syntax', token.line+1, token.cod
e[token.line][token.pointer-1], "'}' expected at this area")
469                         else:
470                             raise Exception(
471                                 'Syntax', token.line+1, token.code[to
ken.line][token.pointer-1], "'{' expected at this area")
472                     else:
473                         raise Exception(
474                             'Syntax', token.line+1, token.code[token.
line][token.pointer-1], "')' expected at this area")
475                 else:
476                     raise Exception(
477                         'Syntax', token.line+1, token.code[token.line
][token.pointer-1], "'(' expected at this area")
478             else:
479                 raise Exception(
```

```
480                     'Syntax', token.line+1, token.code[token.line][to
ken.pointer-1], "an identifier expected at this area")
481         else:
482             raise Exception(
483                 'Syntax', token.line+1, token.code[token.line][token.
pointer-1], "expect a type at this area")
484     else:
485         raise Exception('Syntax', token.line+1,
486                         token.code[token.line][token.pointer-
1], "'static' or 'field' expected at this area")
487
488
# using to test whether it's possibly a statement by checking it's head
489

490 def statementTest(token):
491     nextToken = token.PeekNextToken()
492     if nextToken[0] == 'if' or nextToken[0] == 'var' or nextToken[0]
== 'let' or nextToken[0] == 'while' or nextToken[0] == 'do' or nextToken[
0] == 'return':
493         return True
494     else:
495         return False
496

497 def statement(token):
498     nextToken = token.PeekNextToken()
499     if nextToken[0] == 'if':
500         ifStatement(token)
501     elif nextToken[0] == 'var':
502         varDeclarStatement(token)
503     elif nextToken[0] == 'let':
504         letStatement(token)
505     elif nextToken[0] == 'while':
506         whileStatement(token)
507     elif nextToken[0] == 'do':
508         doStatement(token)
509     elif nextToken[0] == 'return':
510         returnStatement(token)
511     else:
512         raise Exception('Syntax', token.line+1,
513                         token.code[token.line][token.pointer-
1], "expect a statement at this area")
514

515 def paramList(token):
516     nextToken = token.PeekNextToken()
517     if nextToken[0] == "void" or nextToken[0] == "int" or nextToken[0
] == "char" or nextToken[0] == "boolean" or nextToken[1] == "Identifier":
518         type = nextToken[0]
```

```
519         nextToken = token.GetNextToken()
520         nextToken = token.GetNextToken()
521         count = 0
522         if nextToken[1] == "Identifier":
523             symboltable.Add(nextToken[0], type, "argument",False,True
,True, count)
524             count += 1
525             nextToken = token.PeekNextToken()
526             while nextToken[0] == ',':
527                 nextToken = token.GetNextToken()
528                 nextToken = token.GetNextToken()
529                 if nextToken[0] == "void" or nextToken[0] == "int" or
 nextToken[0] == "char" or nextToken[0] == "boolean" or nextToken[1] == "
Identifier":
530                     type = nextToken[0]
531                     nextToken = token.GetNextToken()
532                     if nextToken[1] == "Identifier":
533                         symboltable.Add(nextToken[0], type, "argument
",False,True,True, count)
534                         count += 1
535                     else:
536                         raise Exception(
537                             'Syntax', token.line+1, token.code[token.
line][token.pointer-1], "an identifier expected at this area")
538                 else:
539                     raise Exception(
540                         'Syntax', token.line+1, token.code[token.line
][token.pointer-1], "expect a type at this area")
541                 nextToken = token.PeekNextToken()
542         else:
543             raise Exception(
544                 'Syntax', token.line+1, token.code[token.line][token.
pointer-1], "an identifier expected at this area")
545

546 def varDeclarStatement(token):
547     nextToken = token.GetNextToken()
548     if nextToken[0] == "var":
549         kind = nextToken[0]
550         nextToken = token.GetNextToken()
551         if nextToken[0] == "void" or nextToken[0] == "int" or nextTok
en[0] == "char" or nextToken[0] == "boolean" or nextToken[1] == "Identifi
er":
552             type = nextToken[0]
553             nextToken = token.GetNextToken()
554             numLocalVariables = 0
555             if nextToken[1] == "Identifier":
556                 if not symboltable.Add(nextToken[0], type, kind, Fals
e, True,True, numLocalVariables):
557                     raise Exception(
```

```
558                         'Semantic', token.line+1, token.code[token.li
ne][token.pointer-1], nextToken[0]+" has declared")
559                 numLocalVariables += 1
560                 nextToken = token.GetNextToken()
561                 while nextToken[0] == ',':
562                     nextToken = token.GetNextToken()
563                     if nextToken[1] == "Identifier":
564                         if not symboltable.Add(nextToken[0], type, ki
nd, False, True,True, numLocalVariables):
565                             raise Exception(
566                                 'Semantic', token.line+1, token.code[
token.line][token.pointer-1], nextToken[0]+" has declared")
567                         numLocalVariables += 1
568                     else:
569                         raise Exception(
570                             'Syntax', token.line+1, token.code[token.
line][token.pointer-1], "an identifier expected at this area")
571                     nextToken = token.GetNextToken()
572                 if nextToken[0] == ";":
573                     pass
574                 else:
575                     raise Exception(
576                         'Syntax', token.line+1, token.code[token.line
][token.pointer-1], "';' expected at this area")
577             else:
578                 raise Exception(
579                     'Syntax', token.line+1, token.code[token.line][to
ken.pointer-1], "an identifier expected at this area")
580         else:
581             raise Exception(
582                 'Syntax', token.line+1, token.code[token.line][token.
pointer-1], "expect a type at this area")
583     else:
584         raise Exception('Syntax', token.line+1,
585                         token.code[token.line][token.pointer-
1], "'var' expected at this area")
586
587 def letStatement(token):
588     nextToken = token.GetNextToken()
589     if nextToken[0] == "let":
590         nextToken = token.GetNextToken()
591         if nextToken[1] == "Identifier":
592             res=symboltable.Find(nextToken[0])
593             if not res:
594                 raise Exception('Semantic', token.line+1,
595                         token.code[token.line][token.pointer-
1], nextToken[0]+" hasn't declared")
596             else:
597                 if len(res) == 2:
```

```
598                     if res[0][2]:
599                         pass
600                     else:
601                         symboltable.Add(nextToken[0], res[0][0], res[
0][1], True, False, True)
602                 else:
603                     if res[2]:
604                         pass
605                     else:
606                         symboltable.Add(nextToken[0], res[0], res[1],
 True, False, True)
607                     ftype = res[0]
608             nextToken = token.PeekNextToken()
609             if nextToken[0] == "[":
610                 nextToken = token.GetNextToken()
611                 expression(token)
612                 nextToken = token.GetNextToken()
613                 if nextToken[0] == "]":
614                     nextToken = token.PeekNextToken()
615                 else:
616                     raise Exception(
617                         'Syntax', token.line+1, token.code[token.line
][token.pointer-1], "']' expected at this area")
618             if nextToken[0] == "=":
619                 nextToken = token.GetNextToken()
620                 expression(token)
621                 nextToken = token.GetNextToken()
622                 if nextToken[0] == ";":
623                     pass
624                 else:
625                     raise Exception(
626                         'Syntax', token.line+1, token.code[token.line
][token.pointer-1], "';' expected at this area")
627             else:
628                 raise Exception(
629                     'Syntax', token.line+1, token.code[token.line][to
ken.pointer-1], "'=' expected at this area")
630         else:
631             raise Exception(
632                 'Syntax', token.line+1, token.code[token.line][token.
pointer-1], "an identifier expected at this area")
633     else:
634         raise Exception('Syntax', token.line+1,
635                         token.code[token.line][token.pointer-
1], "'let' expected at this area")
636

637 def ifStatement(token):
638     nextToken = token.GetNextToken()
639     if nextToken[0] == "if":
```

```
640         nextToken = token.GetNextToken()
641         if nextToken[0] == "(":
642             expression(token)
643             nextToken = token.GetNextToken()
644             if nextToken[0] == ")":
645                 global ifnum
646                 symboltable.level.append("if"+str(ifnum))
647                 ifnum+=1
648                 nextToken = token.GetNextToken()
649                 if nextToken[0] == "{":
650                     while statementTest(token):
651                         statement(token)
652                     nextToken = token.GetNextToken()
653                     if nextToken[0] == "}":
654                         nextToken = token.PeekNextToken()
655                         if nextToken[0] == "else":
656                             nextToken = token.GetNextToken()
657                             nextToken = token.GetNextToken()
658                             if nextToken[0] == "{":
659                                 while statementTest(token):
660                                     statement(token)
661                                 nextToken = token.GetNextToken()
662                                 if nextToken[0] == "}":
663                                     pass
664                                 else:
665                                     raise Exception(
666                                         'Syntax', token.line+1, token
.code[token.line][token.pointer-1], "'}' expected at this area")
667                             else:
668                                 raise Exception(
669                                     'Syntax', token.line+1, token.cod
e[token.line][token.pointer-1], "'{' expected at this area")
670                         del symboltable.level[-1]
671                     else:
672                         raise Exception(
673                             'Syntax', token.line+1, token.code[token.
line][token.pointer-1], "'}' expected at this area")
674                 else:
675                     raise Exception(
676                         'Syntax', token.line+1, token.code[token.line
][token.pointer-1], "'{' expected at this area")
677             else:
678                 raise Exception(
679                     'Syntax', token.line+1, token.code[token.line][to
ken.pointer-1], "')' expected at this area")
680         else:
681             raise Exception(
682                 'Syntax', token.line+1, token.code[token.line][token.
pointer-1], "'(' expected at this area")
683     else:
```

```
684         raise Exception('Syntax', token.line+1,
685                         token.code[token.line][token.pointer-
1], "'if' expected at this area")
686
687 def whileStatement(token):
688     nextToken = token.GetNextToken()
689     if nextToken[0] == "while":
690         nextToken = token.GetNextToken()
691         if nextToken[0] == "(":
692             expression(token)
693             nextToken = token.GetNextToken()
694             if nextToken[0] == ")":
695                 global whilenum
696                 symboltable.level.append("while"+str(whilenum))
697                 whilenum+=1
698                 nextToken = token.GetNextToken()
699                 if nextToken[0] == "{":
700                     while statementTest(token):
701                         statement(token)
702                     nextToken = token.GetNextToken()
703                     if nextToken[0] == "}":
704                         del symboltable.level[-1]
705                     else:
706                         raise Exception(
707                             'Syntax', token.line+1, token.code[token.
line][token.pointer-1], "'}' expected at this area")
708                 else:
709                     raise Exception(
710                         'Syntax', token.line+1, token.code[token.line
][token.pointer-1], "'{' expected at this area")
711             else:
712                 raise Exception(
713                     'Syntax', token.line+1, token.code[token.line][to
ken.pointer-1], "')' expected at this area")
714         else:
715             raise Exception(
716                 'Syntax', token.line+1, token.code[token.line][token.
pointer-1], "'(' expected at this area")
717     else:
718         raise Exception('Syntax', token.line+1,
719                         token.code[token.line][token.pointer-
1], "'while' expected at this area")
720
721 def doStatement(token):
722     nextToken = token.GetNextToken()
723     if nextToken[0] == "do":
724         subroutineCall(token)
725         nextToken = token.GetNextToken()
```

```
726         if nextToken[0] == ";":
727             pass
728         else:
729             raise Exception(
730                 'Syntax', token.line+1, token.code[token.line][token.
pointer-1], "';' expected at this area")
731     else:
732         raise Exception('Syntax', token.line+1,
733                         token.code[token.line][token.pointer-
1], "'do' expected at this area")
734 

735 def subroutineCall(token):
736     nextToken = token.GetNextToken()
737     if nextToken[1] == "Identifier":
738         nextToken = token.GetNextToken()
739         if nextToken[0] == ".":
740             nextToken = token.GetNextToken()
741             if nextToken[1] == "Identifier" or nextToken[1] == "Metho
d":
742                 nextToken = token.GetNextToken()
743             else:
744                 raise Exception(
745                     'Syntax', token.line+1, token.code[token.line][to
ken.pointer-1], "an identifier expected at this area")
746         if nextToken[0] == "(":
747             expressionList(token)
748             nextToken = token.GetNextToken()
749             if nextToken[0] == ")":
750                 pass
751             else:
752                 raise Exception(
753                     'Syntax', token.line+1, token.code[token.line][to
ken.pointer-1], "')' expected at this area")
754         else:
755             raise Exception(
756                 'Syntax', token.line+1, token.code[token.line][token.
pointer-1], "'(' expected at this area")
757     else:
758         raise Exception('Syntax', token.line+1,
759                         token.code[token.line][token.pointer-
1], "an identifier expected at this area")
760 

761 def expressionList(token):
762     if factorTest(token):
763         expression(token)
764         while True:
765             nextToken = token.PeekNextToken()
766             if nextToken[0] == ",":
```

```
767                 nextToken = token.GetNextToken()
768                 expression(token)
769             else:
770                 break
771 
772 def returnStatement(token):
773     nextToken = token.GetNextToken()
774     if nextToken[0] == "return":
775         if factorTest(token):
776             expression(token)
777         nextToken = token.GetNextToken()
778         if nextToken[0] == ";":
779             pass
780         else:
781             raise Exception(
782                 'Syntax', token.line+1, token.code[token.line][token.
pointer-1], "'return' expected at this area")
783         nextToken = token.PeekNextToken()
784         if nextToken[0] == "}" or nextToken[0] == "else":
785             pass
786         else:
787             raise Exception('Semantic', token.line+1,
788                             token.code[token.line][token.pointer-
1], "unreachable code after return statement")
789     else:
790         raise Exception('Syntax', token.line+1,
791                         token.code[token.line][token.pointer-
1], "'return' expected at this area")
792 
793 def expression(token):
794     if factorTest(token):
795         relationalExpression(token)
796         nextToken = token.PeekNextToken()
797         while nextToken[0] == '&' or nextToken[0] == '|':
798             nextToken = token.GetNextToken()
799             relationalExpression(token)
800             nextToken = token.PeekNextToken()
801     else:
802         raise Exception('Syntax', token.line+1,
803                         token.code[token.line][token.pointer-
1], "expect a relational expression at this area")
804 
805 def relationalExpression(token):
806     if factorTest(token):
807         arithmeticExpression(token)
808         nextToken = token.PeekNextToken()
809         while True:
```

```
            # I think the provided full jack grammar make a mistake h
ere, so I correct it.
            if nextToken[0] == "=":
                nextToken = token.GetNextToken()
                arithmeticExpression(token)
                nextToken = token.PeekNextToken()
            elif nextToken[0] == '>' or nextToken[0] == '<':
                nextToken = token.GetNextToken()
                nextToken = token.PeekNextToken()
                if nextToken[0] == "=":
                    nextToken = token.GetNextToken()
                arithmeticExpression(token)
                nextToken = token.PeekNextToken()
            else:
                break
    else:
        raise Exception('Syntax', token.line+1,
                        token.code[token.line][token.pointer-
1], "expect a arithmetic expression at this area")

def arithmeticExpression(token):
    if factorTest(token):
        term(token)
        nextToken = token.PeekNextToken()
        while nextToken[0] == '+' or nextToken[0] == '-':
            nextToken = token.GetNextToken()
            term(token)
            nextToken = token.PeekNextToken()
    else:
        raise Exception('Syntax', token.line+1,
                        token.code[token.line][token.pointer-
1], "expect a term at this area")

def term(token):
    if factorTest(token):
        factor(token)
        nextToken = token.PeekNextToken()
        while nextToken[0] == '*' or nextToken[0] == '/':
            nextToken = token.GetNextToken()
            factor(token)
            nextToken = token.PeekNextToken()
    else:
        raise Exception('Syntax', token.line+1,
                        token.code[token.line][token.pointer-
1], "expect a factor at this area")

def factor(token):
```

```
853     nextToken = token.PeekNextToken()
854     if nextToken[0] == '-' or nextToken[0] == '~':
855         nextToken = token.GetNextToken()
856     nextToken = token.GetNextToken()
857     if nextToken[1] == 'Integer' or nextToken[1] == 'String' or nextT
oken[0] == 'true' or nextToken[0] == 'false' or nextToken[0] == 'null' or
 nextToken[0] == 'this':
858         pass
859     elif nextToken[1] == 'Identifier':
860         nextToken = token.PeekNextToken()
861         if nextToken[0] == '.':
862             nextToken = token.GetNextToken()
863             nextToken = token.GetNextToken()
864             if nextToken[1] == 'Identifier':
865                 pass
866             else:
867                 raise Exception(
868                     'Syntax', token.line+1, token.code[token.line][to
ken.pointer-1], "expect a identifier at this area")
869             nextToken = token.PeekNextToken()
870         if nextToken[0] == '[':
871             nextToken = token.GetNextToken()
872             expression(token)
873             nextToken = token.GetNextToken()
874             if nextToken[0] == ']':
875                 pass
876             else:
877                 raise Exception(
878                     'Syntax', token.line+1, token.code[token.line][to
ken.pointer-1], "']' expected at this area")
879         if nextToken[0] == '(':
880             nextToken = token.GetNextToken()
881             expressionList(token)
882             nextToken = token.GetNextToken()
883             if nextToken[0] == ')':
884                 pass
885             else:
886                 raise Exception(
887                     'Syntax', token.line+1, token.code[token.line][to
ken.pointer-1], "')' expected at this area")
888     elif nextToken[0] == '(':
889         expressionList(token)
890         nextToken = token.GetNextToken()
891         if nextToken[0] == ')':
892             pass
893         else:
894             raise Exception(
895                 'Syntax', token.line+1, token.code[token.line][token.
pointer-1], "')' expected at this area")
896
```

```
897
# using to test whether it's possibly an expression by checking it's head
898

899 def factorTest(token):
900     nextToken = token.PeekNextToken()
901     if nextToken[1] == 'Integer' or nextToken[0] == '-
' or nextToken[0] == '~' or nextToken[1] == 'String' or nextToken[0] == '
true' or nextToken[0] == 'false' or nextToken[0] == 'null' or nextToken[0
] == 'this' or nextToken[1] == 'Identifier' or nextToken[0] == '(':
902         return True
903     else:
904         return False
905
906
907 # jcparser.py
908
909 import SymbolTable
910
911 global symboltable, tempexp
912 generatedcode = ""
913 symboltable = SymbolTable.SymbolTable()
914 labelNum = 0
915 numExpressions = 0
916 fieldCount = 0
917 tempexp = ""
918 tempclassN = ""
919 subName = ""
920 isSubroutineBody = False
921 isConstructor = False
922 isMethod = False
923 VM_OPERATORS = {'+': 'add', '-': 'sub', '*': 'call Math.multiply 2',
924                 '/': 'call Math.divide 2', '|': 'or', '&': 'and', '<'
: 'lt', '>': 'gt', '=': 'eq', }
925 UNARY_OPERATORS = {'~': 'not', '-': 'neg'}
926
927 # Code Generation
928

929 def writePush(segment, index):
930     global generatedcode
931     # possible segments: const, arg, local, static, this, that, point
er, temp
932     generatedcode += "push " + segment + " " + str(index)+"\n"
933

934 def writePop(segment, index):
935     global generatedcode
936     generatedcode += "pop " + segment + " " + str(index)+"\n"
937
```

```
938 def writeArithmetic(command):
939     global generatedcode
940     # possible commands: add, sub, neg, eq, gt, lt, and, or, not
941     generatedcode += command+"\n"
942 
943 def writeLabel(label):
944     global generatedcode
945     generatedcode += "label " + label+"\n"
946 
947 def writeGoto(label):
948     global generatedcode
949     generatedcode += "goto " + label+"\n"
950 
951 def writeIf(label):
952     global generatedcode
953     generatedcode += "if-goto " + label+"\n"
954 
955 def writeCall(name, nArgs):
956     global generatedcode
957     generatedcode += "call " + name + " " + str(nArgs)+"\n"
958 
959 def writeFunction(name, nLocals):
960     global generatedcode
961     generatedcode += "function " + name + " " + str(nLocals)+"\n"
962 
963 def writeReturn():
964     global generatedcode
965     generatedcode += "push constant 0\nreturn\n"
966 
967 # Parser
968 
969 def start(token, table, system=False):
970     global symboltable, generatedcode
971     symboltable.table = table
972     symboltable.level = []
973     generatedcode = ""
974     count = 0
975     while True:
976 
977         nextToken = token.PeekNextToken()
978         if nextToken[0] == "class":
979             classDeclar(token, count)
980             count += 1
981         elif nextToken[1] == "EOF":
```

```
982             break
983         else:
984             # Check whether there is code outside the class block
985             raise Exception('Semantic', token.line+1,
986                             token.code[token.line][token.pointer-
1], "unreachable code outside the class block")
987     if system:
988         return symboltable.table
989     else:
990         return generatedcode, symboltable.table
991 

992 def classDeclar(token, count):
993     nextToken = token.GetNextToken()
994     if nextToken[0] == "class":
995         nextToken = token.GetNextToken()
996         if nextToken[1] == "Identifier":
997             global tempclassN
998             tempclassN = nextToken[0]
999             if not symboltable.Add(nextToken[0], nextToken[0], "class
", False, True, False, count):
1000                 raise Exception(
1001                     'Semantic', token.line+1, token.code[token.line]
[token.pointer-1], nextToken[0]+" class has declared")
1002             symboltable.level.append(nextToken[0])
1003             symboltable.Add(nextToken[0], nextToken[0],
1004                             "class", False, True, False, count)
1005             nextToken = token.GetNextToken()
1006             if nextToken[0] == "{":
1007                 nextToken = token.PeekNextToken()
1008                 counts = 0
1009                 countf = 0
1010                 while True:
1011                     if nextToken[0] == 'static':
1012                         classVarDeclar(token, counts)
1013                         counts += 1
1014                     elif nextToken[0] == 'field':
1015                         classVarDeclar(token, countf)
1016                         countf += 1
1017                     else:
1018                         break
1019                     nextToken = token.PeekNextToken()
1020                 countc = 0
1021                 countf = 0
1022                 countm = 0
1023                 while True:
1024                     if nextToken[0] == 'constructor':
1025                         countc = subroutineDeclar(token, countc)
1026                         countc += 1
1027                     elif nextToken[0] == 'function':
```

```
1028                            countf = subroutineDeclar(token, countf)
1029                            countf += 1
1030                        elif nextToken[0] == 'method':
1031                            countm = subroutineDeclar(token, countm)
1032                            countm += 1
1033                        else:
1034                            break
1035                        nextToken = token.PeekNextToken()
1036                    nextToken = token.GetNextToken()
1037                    if nextToken[0] == "}":
1038                        del symboltable.level[-1]
1039                    else:
1040                        raise Exception(
1041                            'Syntax', token.line+1, token.code[token.lin
e][token.pointer-1], "'}' expected at this area")
1042                else:
1043                    raise Exception(
1044                        'Syntax', token.line+1, token.code[token.line][t
oken.pointer-1], "'{' expected at this area")
1045            else:
1046                raise Exception(
1047                    'Syntax', token.line+1, token.code[token.line][token
.pointer-1], "an identifier expected at this area")
1048        else:
1049            raise Exception('Syntax', token.line+1,
1050                            token.code[token.line][token.pointer-
1], "'class' expected at this area")
1051
1052 def classVarDeclar(token, count):
1053        nextToken = token.GetNextToken()
1054        if nextToken[0] == 'static' or nextToken[0] == 'field':
1055            kind = nextToken[0]
1056            if nextToken[0] == 'field':
1057                global fieldCount
1058                fieldCount += 1
1059            nextToken = token.GetNextToken()
1060            if nextToken[0] == "void" or nextToken[0] == "int" or nextTo
ken[0] == "char" or nextToken[0] == "boolean" or nextToken[1] == "Identif
ier":
1061                type = nextToken[0]
1062                nextToken = token.GetNextToken()
1063                if nextToken[1] == "Identifier":
1064                    if not symboltable.Add(nextToken[0], type, kind, Fal
se, True, False, count):
1065                        raise Exception(
1066                            'Semantic', token.line+1, token.code[token.l
ine][token.pointer-1], nextToken[0]+" has declared in this class")
1067                    nextToken = token.GetNextToken()
1068                    while nextToken[0] == ',':
```

```
1069                    nextToken = token.GetNextToken()
1070                    if nextToken[1] == "Identifier":
1071                        count += 1
1072                        if not symboltable.Add(nextToken[0], type, k
ind, False, True, False, count):
1073                            raise Exception(
1074                                'Semantic', token.line+1, token.code
[token.line][token.pointer-
1], nextToken[0]+" has declared in this class")
1075                    else:
1076                        raise Exception(
1077                            'Syntax', token.line+1, token.code[token
.line][token.pointer-1], "an identifier expected at this area")
1078                    nextToken = token.GetNextToken()
1079                if nextToken[0] == ";":
1080                    return count
1081                else:
1082                    raise Exception(
1083                        'Syntax', token.line+1, token.code[token.lin
e][token.pointer-1], "';' expected at this area")
1084            else:
1085                raise Exception(
1086                    'Syntax', token.line+1, token.code[token.line][t
oken.pointer-1], "an identifier expected at this area")
1087        else:
1088            raise Exception(
1089                'Syntax', token.line+1, token.code[token.line][token
.pointer-1], "expect a type at this area")
1090    else:
1091        raise Exception('Syntax', token.line+1,
1092                        token.code[token.line][token.pointer-
1], "'static' or 'field' expected at this area")
1093

1094 def subroutineDeclar(token, count):
1095    global generatedcode
1096    nextToken = token.GetNextToken()
1097    if nextToken[0] == 'constructor' or nextToken[0] == 'function' o
r nextToken[0] == 'method':
1098        kind = nextToken[0]
1099        global isSubroutineBody, isConstructor, isMethod
1100        isSubroutineBody = True
1101        if nextToken[0] == 'constructor':
1102            isConstructor = True
1103        elif nextToken[0] == 'method':
1104            isMethod = True
1105        generatedcode += kind+" "
1106        nextToken = token.GetNextToken()
```

```
1107         if nextToken[0] == "void" or nextToken[0] == "int" or nextTo
ken[0] == "char" or nextToken[0] == "boolean" or nextToken[1] == "Identif
ier":
1108             type = nextToken[0]
1109             nextToken = token.GetNextToken()
1110             if nextToken[1] == "Identifier":
1111                 global subName
1112                 subName = nextToken[0]
1113                 if not symboltable.Add(nextToken[0], type, kind, Fal
se, True, False, count):
1114                     raise Exception(
1115                         'Semantic', token.line+1, token.code[token.l
ine][token.pointer-1], nextToken[0]+" has declared in this class")
1116                 symboltable.level.append(nextToken[0])
1117                 symboltable.Add(nextToken[0], type,
1118                                 kind, False, True, False, count)
1119                 generatedcode += nextToken[0]+" "+str(count)+"\n"
1120                 nextToken = token.GetNextToken()
1121                 if nextToken[0] == "(":
1122                     paramList(token)
1123                     nextToken = token.GetNextToken()
1124                     if nextToken[0] == ")":
1125                         nextToken = token.GetNextToken()
1126                         if nextToken[0] == "{":
1127                             while statementTest(token):
1128                                 statement(token)
1129                             nextToken = token.GetNextToken()
1130                             if nextToken[0] == "}":
1131                                 isSubroutineBody = False
1132                                 isConstructor = False
1133                                 isMethod = False
1134                                 del symboltable.level[-1]
1135                                 return count
1136                             else:
1137                                 raise Exception(
1138                                     'Syntax', token.line+1, token.co
de[token.line][token.pointer-1], "'}' expected at this area")
1139                         else:
1140                             raise Exception(
1141                                 'Syntax', token.line+1, token.code[t
oken.line][token.pointer-1], "'{' expected at this area")
1142                     else:
1143                         raise Exception(
1144                             'Syntax', token.line+1, token.code[token
.line][token.pointer-1], "')' expected at this area")
1145                 else:
1146                     raise Exception(
1147                         'Syntax', token.line+1, token.code[token.lin
e][token.pointer-1], "'(' expected at this area")
1148             else:
```

```
1149                 raise Exception(
1150                     'Syntax', token.line+1, token.code[token.line][t
oken.pointer-1], "an identifier expected at this area")
1151         else:
1152             raise Exception(
1153                 'Syntax', token.line+1, token.code[token.line][token
.pointer-1], "expect a type at this area")
1154     else:
1155         raise Exception('Syntax', token.line+1,
1156                         token.code[token.line][token.pointer-
1], "'static' or 'field' expected at this area")
1157
1158
# using to test whether it's possibly a statement by checking it's head
1159

1160 def statementTest(token):
1161     nextToken = token.PeekNextToken()
1162     if nextToken[0] == 'if' or nextToken[0] == 'var' or nextToken[0]
 == 'let' or nextToken[0] == 'while' or nextToken[0] == 'do' or nextToken
[0] == 'return':
1163         return True
1164     else:
1165         return False
1166

1167 def statement(token):
1168     nextToken = token.PeekNextToken()
1169     if nextToken[0] == 'if':
1170         ifStatement(token)
1171     elif nextToken[0] == 'var':
1172         varDeclarStatement(token)
1173     elif nextToken[0] == 'let':
1174         letStatement(token)
1175     elif nextToken[0] == 'while':
1176         whileStatement(token)
1177     elif nextToken[0] == 'do':
1178         doStatement(token)
1179     elif nextToken[0] == 'return':
1180         returnStatement(token)
1181     else:
1182         raise Exception('Syntax', token.line+1,
1183                         token.code[token.line][token.pointer-
1], "expect a statement at this area")
1184

1185 def paramList(token):
1186     nextToken = token.PeekNextToken()
```

```
1187     if nextToken[0] == "void" or nextToken[0] == "int" or nextToken[
0] == "char" or nextToken[0] == "boolean" or nextToken[1] == "Identifier"
:
1188         type = nextToken[0]
1189         nextToken = token.GetNextToken()
1190         nextToken = token.GetNextToken()
1191         count = 0
1192         if nextToken[1] == "Identifier":
1193             if not symboltable.Add(nextToken[0], type, "argument", F
alse, True, False, count):
1194                 raise Exception(
1195                     'Semantic', token.line+1, token.code[token.line]
[token.pointer-1], nextToken[0]+", fobid using the same parameter")
1196             count += 1
1197             nextToken = token.PeekNextToken()
1198             while nextToken[0] == ',':
1199                 nextToken = token.GetNextToken()
1200                 nextToken = token.GetNextToken()
1201                 if nextToken[0] == "void" or nextToken[0] == "int" o
r nextToken[0] == "char" or nextToken[0] == "boolean" or nextToken[1] ==
"Identifier":
1202                     type = nextToken[0]
1203                     nextToken = token.GetNextToken()
1204                     if nextToken[1] == "Identifier":
1205                         if not symboltable.Add(nextToken[0], type, "
argument", False, True, False, count):
1206                             raise Exception(
1207                                 'Semantic', token.line+1, token.code
[token.line][token.pointer-
1], nextToken[0]+"fobid using the same parameter")
1208                         count += 1
1209                         global numExpressions
1210                         numExpressions += count
1211                     else:
1212                         raise Exception(
1213                             'Syntax', token.line+1, token.code[token
.line][token.pointer-1], "an identifier expected at this area")
1214                 else:
1215                     raise Exception(
1216                         'Syntax', token.line+1, token.code[token.lin
e][token.pointer-1], "expect a type at this area")
1217                 nextToken = token.PeekNextToken()
1218         else:
1219             raise Exception(
1220                 'Syntax', token.line+1, token.code[token.line][token
.pointer-1], "an identifier expected at this area")
1221

1222 def varDeclarStatement(token):
1223     nextToken = token.GetNextToken()
```

```
1224     if nextToken[0] == "var":
1225         kind = nextToken[0]
1226         nextToken = token.GetNextToken()
1227         if nextToken[0] == "void" or nextToken[0] == "int" or nextTo
ken[0] == "char" or nextToken[0] == "boolean" or nextToken[1] == "Identif
ier":
1228             type = nextToken[0]
1229             nextToken = token.GetNextToken()
1230             numLocalVariables = 0
1231             if nextToken[1] == "Identifier":
1232                 if not symboltable.Add(nextToken[0], type, kind, Fal
se, True, False, numLocalVariables):
1233                     raise Exception(
1234                         'Semantic', token.line+1, token.code[token.l
ine][token.pointer-1], nextToken[0]+" has declared")
1235                 numLocalVariables += 1
1236                 nextToken = token.GetNextToken()
1237                 while nextToken[0] == ',':
1238                     nextToken = token.GetNextToken()
1239                     if nextToken[1] == "Identifier":
1240                         if not symboltable.Add(nextToken[0], type, k
ind, False, True, False, numLocalVariables):
1241                             raise Exception(
1242                                 'Semantic', token.line+1, token.code
[token.line][token.pointer-1], nextToken[0]+" has declared")
1243                         numLocalVariables += 1
1244                     else:
1245                         raise Exception(
1246                             'Syntax', token.line+1, token.code[token
.line][token.pointer-1], "an identifier expected at this area")
1247                     nextToken = token.GetNextToken()
1248                 if nextToken[0] == ";":
1249                     if isSubroutineBody:
1250                         global tempclassN, subName, fieldCount
1251                         writeFunction(tempclassN + '.' +
1252                                       subName, numLocalVariables)
1253                         if isConstructor:
1254                             writePush("constant", fieldCount)
1255                             # allocate space for this object
1256                             writeCall("Memory.alloc", 1)
1257                             writePop("pointer", 0)  # assign object
to 'this'
1258                         elif isMethod:
1259                             writePush("argument", 0)
1260                             writePop("pointer", 0)
1261                 else:
1262                     raise Exception(
1263                         'Syntax', token.line+1, token.code[token.lin
e][token.pointer-1], "';' expected at this area")
1264             else:
```

```
1265                 raise Exception(
1266                     'Syntax', token.line+1, token.code[token.line][t
oken.pointer-1], "an identifier expected at this area")
1267         else:
1268             raise Exception(
1269                 'Syntax', token.line+1, token.code[token.line][token
.pointer-1], "expect a type at this area")
1270     else:
1271         raise Exception('Syntax', token.line+1,
1272                         token.code[token.line][token.pointer-
1], "'var' expected at this area")
1273

1274 def letStatement(token):
1275     global generatedcode,tempexp
1276     nextToken = token.GetNextToken()
1277     if nextToken[0] == "let":
1278         nextToken = token.GetNextToken()
1279         if nextToken[1] == "Identifier":
1280             ftype = ""
1281             fkind = ""
1282             identi = nextToken[0]
1283             templev = []
1284             res = symboltable.Find(nextToken[0])
1285             if not res:
1286                 raise Exception('Semantic', token.line+1,
1287                                 token.code[token.line][token.pointer
-1], nextToken[0]+" hasn't declared")
1288             else:
1289                 if len(res) == 2:
1290                     if res[0][2]:
1291                         pass
1292                     else:
1293                         symboltable.Add(
1294                             nextToken[0], res[0][0], res[0][1], True
, False, False)
1295                     ftype = res[0][0]
1296                     fkind = res[0][1]
1297                 else:
1298                     if res[2]:
1299                         pass
1300                     else:
1301                         symboltable.Add(
1302                             nextToken[0], res[0], res[1], True, Fals
e, False)
1303                     ftype = res[0]
1304             nextToken = token.PeekNextToken()
1305             containsList = False
1306             if nextToken[0] == "[":
1307                 nextToken = token.GetNextToken()
```

```
1308                    expression(token)
1309                    nextToken = token.GetNextToken()
1310                    containsList = True
1311                    if len(res) == 2:
1312                        writePush(res[0][1], res[0][3])
1313                    else:
1314                        writePush(res[1], res[3])
1315                    writeArithmetic('add')
1316                    mark = False
1317                    if nextToken[0] == "]":
1318                        if str(symboltable.level) in symboltable.table:
1319                            if identi in symboltable.table[str(symboltab
le.level)]:
1320                                templev = symboltable.level[:]
1321                                mark = True
1322                        if not mark:
1323                            loop = 1
1324                            while loop < len(symboltable.level):
1325                                if str(symboltable.level[:-
loop]) in symboltable.table:
1326                                    if identi in symboltable.table[str(s
ymboltable.level[:-loop])]:
1327                                        templev = symboltable.level[:-
loop]
1328                                        break
1329                                    loop += 1
1330                        if templev == []:
1331                            raise Exception('Semantic', token.line+1,
1332                                            token.code[token.line][token
.pointer-1], identi+" can't be found")
1333                        else:
1334                            templev.append(identi)
1335                        nextToken = token.PeekNextToken()
1336                    else:
1337                        raise Exception(
1338                            'Syntax', token.line+1, token.code[token.lin
e][token.pointer-1], "']' expected at this area")
1339                if nextToken[0] == "=":
1340                    nextToken = token.GetNextToken()
1341                    tempexpss=tempexp
1342                    tempexp = ""
1343                    expression(token)
1344                    if containsList:
1345                        writePop('temp', 0)
1346                        writePop('pointer', 1)
1347                        writePush('temp', 0)
1348                        writePop('that', 0)
1349                    else:
1350                        if len(res) == 2:
1351                            writePop(res[0][1], res[0][3])
```

```
1352                        else:
1353                            writePop(res[1], res[3])
1354                    if not containsList:
1355                        try:
1356                            if tempexp == ftype:
1357                                pass
1358                            elif str(type(eval(tempexp))) == "<class '"+
ftype+"'>":
1359                                pass
1360                            elif str(type(eval(tempexp))) == "<class 'bo
ol'>" and ftype == "boolean":
1361                                pass
1362                            elif str(type(eval(tempexp))) == "<class 'in
t'>" and ftype == "char":
1363                                pass
1364                            elif str(type(eval(tempexp))) == "<class 'fl
oat'>" and ftype == "int":
1365                                pass
1366                            else:
1367                                raise Exception('Semantic', token.line+1
,
1368                                                token.code[token.line][t
oken.pointer-1], "wrong type for assignment")
1369                        except Exception:
1370                            raise Exception('Semantic', token.line+1,
1371                                            token.code[token.line][token
.pointer-1], "wrong type for assignment")
1372                    else:
1373                        ftype = ""
1374                        try:
1375                            if eval(tempexp) == None:
1376                                result = 0
1377                                tempexps = tempexp
1378                                tempexp = "result="+tempexp
1379                                eval(tempexp)
1380                                ftype = str(type(result)).replace(
1381                                    "<class '", "").replace("'>", "")
1382                            else:
1383                                ftype = tempexp.replace(
1384                                    "<class '", "").replace("'>", "")
1385                        except Exception:
1386                            ftype = tempexp
1387                        templ = symboltable.level[:]
1388                        symboltable.level = templev[:]
1389                        symboltable.Add("Array", ftype, fkind, True, Tru
e, False)
1390                        symboltable.level = templ[:]
1391                    nextToken = token.GetNextToken()
1392                    if nextToken[0] == ";":
1393                        pass
```

```
1394                 else:
1395                     raise Exception(
1396                         'Syntax', token.line+1, token.code[token.lin
e][token.pointer-1], "';' expected at this area")
1397                 tempexp=tempexpss
1398             else:
1399                 raise Exception(
1400                     'Syntax', token.line+1, token.code[token.line][t
oken.pointer-1], "'=' expected at this area")
1401         else:
1402             raise Exception(
1403                 'Syntax', token.line+1, token.code[token.line][token
.pointer-1], "an identifier expected at this area")
1404     else:
1405         raise Exception('Syntax', token.line+1,
1406                         token.code[token.line][token.pointer-
1], "'let' expected at this area")
1407

1408 def ifStatement(token):
1409     global labelNum
1410     nextToken = token.GetNextToken()
1411     if nextToken[0] == "if":
1412         trueLabel = "IF_TRUE" + str(labelNum)
1413         falseLabel = "IF_FALSE" + str(labelNum)
1414         endLabel = "IF_END" + str(labelNum)
1415         nextToken = token.GetNextToken()
1416         if nextToken[0] == "(":
1417             expression(token)
1418             nextToken = token.GetNextToken()
1419             writeIf(trueLabel)
1420             writeGoto(falseLabel)
1421             writeLabel(trueLabel)
1422             if nextToken[0] == ")":
1423                 symboltable.level.append("if"+str(labelNum))
1424                 labelNum += 1
1425                 nextToken = token.GetNextToken()
1426                 if nextToken[0] == "{":
1427                     while statementTest(token):
1428                         statement(token)
1429                     nextToken = token.GetNextToken()
1430                     if nextToken[0] == "}":
1431                         del symboltable.level[-1]
1432                         nextToken = token.PeekNextToken()
1433                         if nextToken[0] == "else":
1434                             writeGoto(endLabel)
1435                             writeLabel(falseLabel)
1436                             writeLabel(endLabel)
1437                             symboltable.level.append("else"+str(labe
lNum-1))
```

```
1438                                nextToken = token.GetNextToken()
1439                                nextToken = token.GetNextToken()
1440                                if nextToken[0] == "{":
1441                                    while statementTest(token):
1442                                        statement(token)
1443                                    nextToken = token.GetNextToken()
1444                                    if nextToken[0] == "}":
1445                                        del symboltable.level[-1]
1446                                    else:
1447                                        raise Exception(
1448                                            'Syntax', token.line+1, toke
n.code[token.line][token.pointer-1], "'}' expected at this area")
1449                                else:
1450                                    raise Exception(
1451                                        'Syntax', token.line+1, token.co
de[token.line][token.pointer-1], "'{' expected at this area")
1452                            else:
1453                                writeLabel(falseLabel)
1454                        else:
1455                            raise Exception(
1456                                'Syntax', token.line+1, token.code[token
.line][token.pointer-1], "'}' expected at this area")
1457                    else:
1458                        raise Exception(
1459                            'Syntax', token.line+1, token.code[token.lin
e][token.pointer-1], "'{' expected at this area")
1460                else:
1461                    raise Exception(
1462                        'Syntax', token.line+1, token.code[token.line][t
oken.pointer-1], "')' expected at this area")
1463            else:
1464                raise Exception(
1465                    'Syntax', token.line+1, token.code[token.line][token
.pointer-1], "'(' expected at this area")
1466        else:
1467            raise Exception('Syntax', token.line+1,
1468                            token.code[token.line][token.pointer-
1], "'if' expected at this area")
1469
1470 def whileStatement(token):
1471     global labelNum
1472     nextToken = token.GetNextToken()
1473     if nextToken[0] == "while":
1474         nextToken = token.GetNextToken()
1475         if nextToken[0] == "(":
1476             writeLabel('WHILE_EXP'+str(labelNum))
1477             expression(token)
1478             writeArithmetic('not')
1479             nextToken = token.GetNextToken()
```

```
1480            if nextToken[0] == ")":
1481                writeIf('WHILE_END'+str(labelNum))
1482                symboltable.level.append("while"+str(labelNum))
1483                labelNum += 1
1484                nextToken = token.GetNextToken()
1485                if nextToken[0] == "{":
1486                    while statementTest(token):
1487                        statement(token)
1488                    nextToken = token.GetNextToken()
1489                    writeGoto('WHILE_EXP'+str(labelNum))
1490                    writeLabel('WHILE_END'+str(labelNum))
1491                    if nextToken[0] == "}":
1492                        del symboltable.level[-1]
1493                    else:
1494                        raise Exception(
1495                            'Syntax', token.line+1, token.code[token
.line][token.pointer-1], "'}' expected at this area")
1496                else:
1497                    raise Exception(
1498                        'Syntax', token.line+1, token.code[token.lin
e][token.pointer-1], "'{' expected at this area")
1499            else:
1500                raise Exception(
1501                    'Syntax', token.line+1, token.code[token.line][t
oken.pointer-1], "')' expected at this area")
1502        else:
1503            raise Exception(
1504                'Syntax', token.line+1, token.code[token.line][token
.pointer-1], "'(' expected at this area")
1505    else:
1506        raise Exception('Syntax', token.line+1,
1507                        token.code[token.line][token.pointer-
1], "'while' expected at this area")
1508

1509 def doStatement(token):
1510    nextToken = token.GetNextToken()
1511    if nextToken[0] == "do":
1512        subroutineCall(token)
1513        writePop('temp', 0)
1514        nextToken = token.GetNextToken()
1515        if nextToken[0] == ";":
1516            pass
1517        else:
1518            raise Exception(
1519                'Syntax', token.line+1, token.code[token.line][token
.pointer-1], "';' expected at this area")
1520    else:
1521        raise Exception('Syntax', token.line+1,
```

```
1522                         token.code[token.line][token.pointer-
1], "'do' expected at this area")
1523

1524 def subroutineCall(token):
1525     global tempexp
1526     nextToken = token.GetNextToken()
1527     isObjorClass = False
1528     isExpress=False
1529     if nextToken[1] == "Identifier":
1530         ident = [nextToken[0]]
1531         sub_identifier = ""
1532         nextToken = token.GetNextToken()
1533         if nextToken[0] == ".":
1534             isObjorClass = True
1535             nextToken = token.GetNextToken()
1536             if nextToken[1] == "Identifier":
1537                 if str([ident[0]]) in symboltable.table:
1538                     pass
1539                 else:
1540                     res=symboltable.Find(ident[0])
1541                     if not res:
1542                         raise Exception('Semantic', token.line+1,
1543                                         token.code[token.line][token
.pointer-1], ident[0]+" can't be found")
1544                     elif len(res) == 2:
1545                         ident[0]= res[0][0]
1546                         isExpress=True
1547                     else:
1548                         ident[0] = res[0]
1549                         isExpress=True
1550                 if nextToken[0] in symboltable.table[str(ident)]:
1551                     sub_identifier = nextToken[0]
1552                     ident.append(nextToken[0])
1553                 else:
1554                     raise Exception('Semantic', token.line+1,
1555                                     token.code[token.line][token.poi
nter-1], nextToken[0]+" can't be found")
1556                 nextToken = token.GetNextToken()
1557             else:
1558                 raise Exception(
1559                     'Syntax', token.line+1, token.code[token.line][t
oken.pointer-1], "an identifier expected at this area")
1560         if nextToken[0] == "(":
1561             tempexpss=tempexp
1562             tempexp = ""
1563             expressionList(token, ident)
1564             nextToken = token.GetNextToken()
1565             if nextToken[0] == ")":
1566                 callName = ""
```

```
1567                 if isObjorClass:
1568                     global numExpressions
1569                     callName = ident[0] + "." + sub_identifier
1570                     if isExpress:
1571                         numExpressions += 1
1572                     writeCall(callName, numExpressions)
1573                     # if there is only 1 identifer and it is a metho
d,
1574                     # push it on to the stack first as first param
1575                 else:
1576                     if isExpress:
1577                         res=symboltable.Find(ident[0])
1578                         if len(res) == 2:
1579                             writePush(res[0][1], res[0][4])
1580                         else:
1581                             writePush(res[1], res[4])
1582                     else:
1583                         writePush('pointer', 0)
1584             else:
1585                 raise Exception(
1586                     'Syntax', token.line+1, token.code[token.line][t
oken.pointer-1], "')' expected at this area")
1587             tempexp=tempexpss
1588         else:
1589             raise Exception(
1590                 'Syntax', token.line+1, token.code[token.line][token
.pointer-1], "'(' expected at this area")
1591     else:
1592         raise Exception('Syntax', token.line+1,
1593                         token.code[token.line][token.pointer-
1], "an identifier expected at this area")
1594

1595 def expressionList(token, function):
1596     global tempexp
1597     tempexpss = tempexp
1598     tempexp = ""
1599     argu = {}
1600     argulist = {}
1601     if str(function) in symboltable.table:
1602         argulist = symboltable.table[str(function)]
1603         for key in argulist:
1604             if argulist[key][1] != 'argument':
1605                 continue
1606             argu[str(argulist[key][3])] = argulist[key][0]
1607         argcount = 0
1608         if factorTest(token):
1609             expression(token)
1610             if len(argu) > argcount:
1611                 try:
```

```
1612                    if tempexp == argu[str(argcount)]:
1613                        pass
1614                    elif str(type(eval(tempexp))) == "<class '"+argu
[str(argcount)]+"'>":
1615                        pass
1616                    elif str(type(eval(tempexp))) == "<class 'bool'>
" and argu[str(argcount)] == "boolean":
1617                        pass
1618                    elif str(type(eval(tempexp))) == "<class 'int'>"
 and argu[str(argcount)] == "char":
1619                        pass
1620                    elif str(type(eval(tempexp))) == "<class 'float'
>" and argu[str(argcount)] == "int":
1621                        pass
1622                    else:
1623                        raise Exception('Semantic', token.line+1,
1624                                        token.code[token.line][token
.pointer-1], "wrong given argument value type")
1625                except Exception:
1626                    raise Exception('Semantic', token.line+1,
1627                                    token.code[token.line][token.poi
nter-1], "wrong given argument value type")
1628            else:
1629                raise Exception('Semantic', token.line+1,
1630                                token.code[token.line][token.pointer
-1], "wrong given argument number")
1631
1632            while True:
1633                nextToken = token.PeekNextToken()
1634                if nextToken[0] == ",":
1635                    argcount += 1
1636                    tempexp = ""
1637                    nextToken = token.GetNextToken()
1638                    expression(token)
1639                    if len(argu) > argcount:
1640                        try:
1641                            if tempexp == argu[str(argcount)]:
1642                                pass
1643                            elif str(type(eval(tempexp))) == "<class
 '"+argu[str(argcount)]+"'>":
1644                                pass
1645                            elif str(type(eval(tempexp))) == "<class
 'bool'>" and argu[str(argcount)] == "boolean":
1646                                pass
1647                            elif str(type(eval(tempexp))) == "<class
 'int'>" and argu[str(argcount)] == "char":
1648                                pass
1649                            elif str(type(eval(tempexp))) == "<class
 'float'>" and argu[str(argcount)] == "int":
1650                                pass
```

```
1651                                else:
1652                                    raise Exception('Semantic', token.li
ne+1,
1653                                                    token.code[token.lin
e][token.pointer-1], "wrong given argument value type")
1654                            except Exception:
1655                                raise Exception('Semantic', token.line+1
,
1656                                                token.code[token.line][t
oken.pointer-1], "wrong given argument value type")
1657                        else:
1658                            raise Exception('Semantic', token.line+1,
1659                                                token.code[token.line][token
.pointer-1], "wrong given argument number")
1660                    else:
1661                        break
1662        else:
1663            pass
1664        tempexp = tempexpss
1665
1666 def returnStatement(token):
1667        global tempexp
1668        nextToken = token.GetNextToken()
1669        ftype = ""
1670        if nextToken[0] == "return":
1671            writeReturn()
1672            if str(symboltable.level) in symboltable.table:
1673                if symboltable.level[-
1] in symboltable.table[str(symboltable.level)]:
1674                    ftype = symboltable.table[str(symboltable.level)][sy
mboltable.level[-1]][0]
1675            loop = 1
1676            while loop < len(symboltable.level) and ftype=="":
1677                if str(symboltable.level[:-loop]) in symboltable.table:
1678                    if symboltable.level[-loop-
1] in symboltable.table[str(symboltable.level[:-loop])]:
1679                        ftype = symboltable.table[str(symboltable.level[
:-loop])][symboltable.level[-loop-1]][0]
1680                        break
1681                loop += 1
1682            tempexpss=tempexp
1683            tempexp = ""
1684            if factorTest(token):
1685                expression(token)
1686            try:
1687                if tempexp == "" and ftype == "void":
1688                    pass
1689                elif tempexp == ftype:
1690                    pass
```

```
1691                elif str(type(eval(tempexp))) == "<class '"+ftype+"'>":
1692                    pass
1693                elif str(type(eval(tempexp))) == "<class 'bool'>" and ft
ype == "boolean":
1694                    pass
1695                elif str(type(eval(tempexp))) == "<class 'int'>" and fty
pe == "char":
1696                    pass
1697                elif str(type(eval(tempexp))) == "<class 'float'>" and f
type == "int":
1698                    pass
1699                else:
1700                    raise Exception('Semantic', token.line+1,
1701                                    token.code[token.line][token.pointer
-1], "wrong return type")
1702            except Exception:
1703                raise Exception('Semantic', token.line+1,
1704                                token.code[token.line][token.pointer-
1], "wrong return type")
1705            tempexp=tempexpss
1706            nextToken = token.GetNextToken()
1707            if nextToken[0] == ";":
1708                pass
1709            else:
1710                raise Exception(
1711                    'Syntax', token.line+1, token.code[token.line][token
.pointer-1], "'return' expected at this area")
1712            nextToken = token.PeekNextToken()
1713            if nextToken[0] == "}" or nextToken[0] == "else":
1714                pass
1715            else:
1716                raise Exception('Semantic', token.line+1,
1717                                token.code[token.line][token.pointer-
1], "unreachable code after return statement")
1718        else:
1719            raise Exception('Syntax', token.line+1,
1720                            token.code[token.line][token.pointer-
1], "'return' expected at this area")
1721

1722 def expression(token):
1723        global tempexp
1724        if factorTest(token):
1725            relationalExpression(token)
1726            nextToken = token.PeekNextToken()
1727            while nextToken[0] == '&' or nextToken[0] == '|':
1728                tempexp += nextToken[0]
1729                writeArithmetic(VM_OPERATORS[nextToken[0]])
1730                nextToken = token.GetNextToken()
1731                relationalExpression(token)
```

```
1732            nextToken = token.PeekNextToken()
1733    else:
1734        raise Exception('Syntax', token.line+1,
1735                        token.code[token.line][token.pointer-
1], "expect a relational expression at this area")
1736

1737 def relationalExpression(token):
1738    global tempexp
1739    if factorTest(token):
1740        arithmeticExpression(token)
1741        nextToken = token.PeekNextToken()
1742        while True:
1743            # I think the provided full jack grammar make a mistake
here, so I correct it.
1744            if nextToken[0] == "=":
1745                tempexp += nextToken[0]
1746                writeArithmetic(VM_OPERATORS[nextToken[0]])
1747                nextToken = token.GetNextToken()
1748                arithmeticExpression(token)
1749                nextToken = token.PeekNextToken()
1750            elif nextToken[0] == '>' or nextToken[0] == '<':
1751                tempexp += nextToken[0]
1752                writeArithmetic(VM_OPERATORS[nextToken[0]])
1753                nextToken = token.GetNextToken()
1754                nextToken = token.PeekNextToken()
1755                if nextToken[0] == "=":
1756                    tempexp += nextToken[0]
1757                    writeArithmetic(VM_OPERATORS[nextToken[0]])
1758                    nextToken = token.GetNextToken()
1759                arithmeticExpression(token)
1760                nextToken = token.PeekNextToken()
1761            else:
1762                break
1763    else:
1764        raise Exception('Syntax', token.line+1,
1765                        token.code[token.line][token.pointer-
1], "expect a arithmetic expression at this area")
1766

1767 def arithmeticExpression(token):
1768    global tempexp
1769    if factorTest(token):
1770        term(token)
1771        nextToken = token.PeekNextToken()
1772        while nextToken[0] == '+' or nextToken[0] == '-':
1773            tempexp += nextToken[0]
1774            writeArithmetic(VM_OPERATORS[nextToken[0]])
1775            nextToken = token.GetNextToken()
1776            term(token)
```

```
1777             nextToken = token.PeekNextToken()
1778     else:
1779         raise Exception('Syntax', token.line+1,
1780                         token.code[token.line][token.pointer-
1], "expect a term at this area")
1781 

1782 def term(token):
1783     global tempexp
1784     if factorTest(token):
1785         factor(token)
1786         nextToken = token.PeekNextToken()
1787         while nextToken[0] == '*' or nextToken[0] == '/':
1788             tempexp += nextToken[0]
1789             writeArithmetic(VM_OPERATORS[nextToken[0]])
1790             nextToken = token.GetNextToken()
1791             factor(token)
1792             nextToken = token.PeekNextToken()
1793     else:
1794         raise Exception('Syntax', token.line+1,
1795                         token.code[token.line][token.pointer-
1], "expect a factor at this area")
1796 

1797 def factor(token):
1798     global tempexp
1799     nextToken = token.PeekNextToken()
1800     if nextToken[0] == '-' or nextToken[0] == '~':
1801         tempexp += nextToken[0]
1802         writeArithmetic(UNARY_OPERATORS[nextToken[0]])
1803         nextToken = token.GetNextToken()
1804     nextToken = token.GetNextToken()
1805     if nextToken[1] == 'Integer' or nextToken[1] == 'String' or next
Token[0] == 'true' or nextToken[0] == 'false' or nextToken[0] == 'null':
1806         if nextToken[0] == 'true':
1807             tempexp += 'True'
1808             writePush('constant', 0)
1809             writeArithmetic('not')
1810         elif nextToken[0] == 'false':
1811             tempexp += 'False'
1812             writePush('constant', 1)
1813         elif nextToken[0] == 'null':
1814             tempexp += 'None'
1815         else:
1816             if nextToken[1] == 'String':
1817                 if tempexp != "":
1818                     raise Exception('Semantic', token.line+1,
1819                                     token.code[token.line][token.poi
nter-1], " wrong expression!")
1820                 else:
```

```
1821                    tempexp = "String"
1822            else:
1823                tempexp += str(nextToken[0])
1824        if nextToken[1] == 'Integer':
1825            writePush('constant', nextToken[0])
1826        elif nextToken[1] == 'String':
1827            writePush('constant', len(nextToken[0]))
1828            writeCall('String.new', 1)
1829            for i in range(len(nextToken[0])):
1830                writePush('constant', ord(nextToken[0][i]))
1831                writeCall('String.appendChar', 2)
1832    elif nextToken[1] == 'Identifier' or nextToken[0] == 'this':
1833        ident = []
1834        if nextToken[1] == 'Identifier':
1835            ident = [nextToken[0]]
1836        else:
1837            global tempclassN
1838            ident = [tempclassN]
1839        nextToken = token.PeekNextToken()
1840        dot = False
1841        if nextToken[0] == '.':
1842            nextToken = token.GetNextToken()
1843            nextToken = token.GetNextToken()
1844            if nextToken[1] == "Identifier":
1845                dot = True
1846                if str([ident[0]]) in symboltable.table:
1847                    pass
1848                else:
1849                    res=symboltable.Find(ident[0])
1850                    if not res:
1851                        raise Exception('Semantic', token.line+1,
1852                                        token.code[token.line][token
.pointer-1], ident[0]+" can't be found")
1853                    elif len(res) == 2:
1854                        ident[0] = res[0][0]
1855                    else:
1856                        ident[0] = res[0]
1857                if nextToken[0] in symboltable.table[str(ident)]:
1858                    ident.append(nextToken[0])
1859                else:
1860                    raise Exception('Semantic', token.line+1,
1861                                    token.code[token.line][token.poi
nter-1], nextToken[0]+" can't be found")
1862                nextToken = token.PeekNextToken()
1863                if nextToken[0] == "(":
1864                    nextToken = token.GetNextToken()
1865                    if nextToken[0] == ")":
1866                        pass
1867                    else:
1868                        expressionList(token, ident)
```

```
1869                        nextToken = token.GetNextToken()
1870                        if nextToken[0] == ")":
1871                            pass
1872                        else:
1873                            raise Exception('Syntax', token.line+1,
1874                                            token.code[token.line][t
oken.pointer-1], " ')' expect here")
1875                    if ident[-
1] in symboltable.table[str([ident[0]])]:
1876                        res = symboltable.table[str([ident[0]])][ide
nt[-1]]
1877                        if res[0] == "int" or res[0] == "char":
1878                            tempexp += "1"
1879                        elif res[0] == "boolean":
1880                            tempexp += "True"
1881                        else:
1882                            tempexp += res[0]
1883                    else:
1884                        raise Exception('Semantic', token.line+1,
1885                                        token.code[token.line][token
.pointer-1], ident[-1]+" can't be found")
1886                else:
1887                    if ident[-
1] in symboltable.table[str([ident[0]])]:
1888                        res = symboltable.table[str([ident[0]])][ide
nt[-1]]
1889                        if res[0] == "int" or res[0] == "char":
1890                            tempexp += "1"
1891                        elif res[0] == "boolean":
1892                            tempexp += "True"
1893                        else:
1894                            tempexp += res[0]
1895                    else:
1896                        raise Exception('Semantic', token.line+1,
1897                                        token.code[token.line][token
.pointer-1], ident[-1]+" can't be found")
1898            else:
1899                raise Exception(
1900                    'Syntax', token.line+1, token.code[token.line][t
oken.pointer-1], "expect a identifier at this area")
1901        if nextToken[0] == '[':
1902            ident.append("Array")
1903            tempexps = tempexp
1904            tempexp = ""
1905            nextToken = token.GetNextToken()
1906            expression(token)
1907            tempexp = tempexps
1908            nextToken = token.GetNextToken()
1909            if nextToken[0] == ']':
1910                pass
```

```
1911             else:
1912                 raise Exception(
1913                     'Syntax', token.line+1, token.code[token.line][t
oken.pointer-1], "']' expected at this area")
1914         if nextToken[0] == '(':
1915             tempexps = tempexp
1916             tempexp = ""
1917             nextToken = token.GetNextToken()
1918             expressionList(token, ident)
1919             tempexp = tempexps
1920             nextToken = token.GetNextToken()
1921             if nextToken[0] == ')':
1922                 pass
1923             else:
1924                 raise Exception(
1925                     'Syntax', token.line+1, token.code[token.line][t
oken.pointer-1], "')' expected at this area")
1926         if not dot:
1927             if str(ident) in symboltable.table:
1928                     tempexp += ident[0]
1929             elif len(ident) == 1 or (len(ident) == 2 and ident[-
1] == "Array"):
1930                 res = symboltable.Find(ident[0])
1931                 templev = []
1932                 if len(ident) != 1:
1933                     mark = False
1934                     if str(symboltable.level) in symboltable.table:
1935                         if ident[0] in symboltable.table[str(symbolt
able.level)]:
1936                             templev = symboltable.level[:]
1937                             mark = True
1938                     if not mark:
1939                         loop = 1
1940                         while loop < len(symboltable.level):
1941                             if str(symboltable.level[:-
loop]) in symboltable.table:
1942                                 if ident[0] in symboltable.table[str
(symboltable.level[:-loop])]:
1943                                     templev = symboltable.level[:-
loop]
1944                                     break
1945                                 loop += 1
1946
1947                     if templev == []:
1948                         raise Exception('Semantic', token.line+1,
1949                                         token.code[token.line][token
.pointer-1], ident[-1]+" can't be found")
1950                     else:
1951                         templev.append(ident[0])
1952                         if str(templev) in symboltable.table:
```

```
1953                                if "Array" in symboltable.table[str(temp
lev)]:
1954                                    res = symboltable.table[str(templev)
]["Array"]
1955                                else:
1956                                    raise Exception('Semantic', token.li
ne+1,
1957                                                    token.code[token.lin
e][token.pointer-1], "Array can't be found")
1958                            else:
1959                                raise Exception('Semantic', token.line+1
,
1960                                                token.code[token.line][t
oken.pointer-1], ident[0]+" can't be found")
1961                    if not res:
1962                        raise Exception('Semantic', token.line+1,
1963                                        token.code[token.line][token.poi
nter-1], ident[-1]+" can't be found")
1964                    else:
1965                        if len(res) == 2:
1966                            if res[0][0] == "int" or res[0][0] == "char"
:
1967                                tempexp += "1"
1968                            elif res[0][0] == "boolean":
1969                                tempexp += "True"
1970                            else:
1971                                tempexp += res[0][0]
1972                        else:
1973                            if res[0] == "int" or res[0] == "char":
1974                                tempexp += "1"
1975                            elif res[0] == "boolean":
1976                                tempexp += "True"
1977                            else:
1978                                tempexp += res[0]
1979                else:
1980                    raise Exception('Semantic', token.line+1,
1981                                    token.code[token.line][token.pointer
-1], ident[-1]+" can't be found")
1982        elif nextToken[0] == '(':
1983            tempexp += "("
1984            expression(token)
1985            nextToken = token.GetNextToken()
1986            if nextToken[0] == ')':
1987                tempexp += ")"
1988            else:
1989                raise Exception(
1990                    'Syntax', token.line+1, token.code[token.line][token
.pointer-1], "')' expected at this area")
1991
```

```
1992
# using to test whether it's possibly an expression by checking it's head
1993 def factorTest(token):
1994     nextToken = token.PeekNextToken()
1995     if nextToken[1] == 'Integer' or nextToken[0] == '-
' or nextToken[0] == '~' or nextToken[1] == 'String' or nextToken[0] == '
true' or nextToken[0] == 'false' or nextToken[0] == 'null' or nextToken[0
] == 'this' or nextToken[1] == 'Identifier' or nextToken[0] == '(':
1996         return True
1997     else:
1998         return False
```

目录

一、 引言

Jack 语言是《计算机系统要素：从零开始构建现代计算机》书中所描述的一种面向对象编程语言。然而书中并没有提供此语言的编译器，而市面上也没有使用 Python 3 语言编写的能够完全实现必要语义分析的 Jack 语言编译软件，因而本人通过独立手动编程，完成了这一软件的创作。此软件可以自动遍历各个文件夹下的 Jack 源代码文件（.jack），并通过词法分析，语法分析，语义分析步骤检查其中的错误，将 Jack 语言文件转换为 Jack 虚拟机所使用的汇编语言文件（.vm）。

Jack 语言的系统库文件和 Jack 虚拟机都可以到这里进行下载：
https://www.nand2tetris.org/software

二、 软件概述

1. 软件结构

```
.
│   jcparser.py
│   lexer.py
│   myjc.py
│   SymbolTable.py
│
└──syslib
        Array.jack
        Keyboard.jack
        Math.jack
        Memory.jack
        Output.jack
        Screen.jack
        String.jack
        Sys.jack
```

其中 syslib 文件下下存放的是 Jack 语言的系统库文件。

2. 功能

1. Jack 语言编译软件

Jack 程序是一个或多个类的集合。每个类都在自己的源代码文件中定义。Jack 源代码文件（可以称为类文件）必须具有.jack 扩展名。一个 Jack 程序的所有源文件程序应该存储在同一个目录中。

此编译软件(myjc.py)接受包含一个或多个 JACK 的目录作为输入源文件。对于每个源文件，编译器生成一个等效的 VM 代码文件（类似于汇编语言），与源文件同名，但扩展名为.vm（vm=virtual machine）。目标代码文件在与源文件相同的目录中被创建。编译后，目录将包含.jack 源文件和相应的.vm 文件。使用 Jack 官方提供的运行虚拟机，将提供的系统（库）文件复制到 syslib 目录中，然后加载整个目录进入虚拟机运行程序。

2. 调用编译器

此软件使用命令行调用编译器。编译器接受一个命令参数，表示包 Jack 源文件的目录的地址或者名称。例如，该编译器经过封装打包后可执行文件名为 myjc，Jack 程序的目录名为 myprog，编译器可以通过在终端键入以下命令来调用：

myjc myprog

3. 错误报告

在编译过程的任何阶段，如果遇到错误，编译器将会打印出提示性的错误消息，并引用遇到错误的行号。消息还会引用发生错误的关键词。下面是一个典型错误的例子：

```
Error, line 103, close to ";", an identifier is expected at this position
```

4. 词法分析

词法分析器（lexer.py）模块用来读取 Jack 源文件（扩展名为.JACK）并从该文件中提取生成标记。lexer 主要通过两种方法向其他模块（解析器）提供标记：

- Token GetNextToken()：每当调用此方法时，它将返回下一个可用的标记，并将指针移动到下一个标记。
- Token PeekNextToken()：每当调用此方法时，它将返回一个可用的标记，但指针并不发生移动。所以，下次解析器调用 GetNextToken 或 PeekNextToken，它得到与这一次相同的标记。

Lexer 同时能够成功地从输入文件中删除空白和注释，并且正确地提取源代码的所有标记，报告源文件包含任何类型的词法错误。

5. 语法分析

Jack 的语法请参见《计算机系统要素：从零开始构建现代计算机》，这里不再叙述。

此软件实现了一个递归下降解析器（jcparser.py）。此递归下降解析器是递归函数的集合，各种语法错误都可以正地报告。当编译器在遇到源文件中的第一个错误时即会停止编译此文件。

6. 符号表

符号表(SymbolTable.py)来存储所有程序标识符及其属性。这里的标识符指的是程序中定义的任何标识符，如变量或方法名。

7. 语义分析

语义分析器用来查找和报告 JACK 程序中可能的语义错误。此分析器不是一个独立的模块，我在解析器函数的适当位置插入了语义分析语句。

下面是 JACK 编译器应该能够执行的语义检查任务的列表。

1. 变量在使用前必须声明。例如，以下 JACK 代码片段不正确，因为变量 y 既没有在函数 f 的本地作用域中声明，也没有在类 X 的作用域中声明：

```
class X{
  field int x;
  function void f (int a){
    let x = a/2;
    let y = x+1;  // 错误, y 尚未被申明
    }
}
```

即使编译器尚未遇到来自其他类的变量声明，也允许使用这些变量。但是，这些声明必须在以后编译所有源代码时解析。因此，这个 JACK 代码片段是正确的：

```
// 一个文件
class X {
    function void f (int a){
        var Y y;
        char c;
        let y =Y.new ();
        let c=y.v;
      }
    }
// 另一个文件
class Y{
  field char v;
    //等等
}
```

2. 一个标识符只能在同一范围内声明一次（不能对同一标识符重新声明）。这适用于所有作用域，无论它是子例程的本地作用域、类的作用域还是整个程序作用域。例如，不允许在同一程序中声明两个同名的类（尽管它们在不同的文件中声明）。变量在用于表达式之前必须初始化（赋值）。因此，以下 JACK 代码片段不正确，因为变量 b 在初始化之前正在使用：

```
    method int m ()
    {
      var boolean b;
      if (b) {                  // 错误, b 在被初始化之前就被使用
      // … etc
      }
    }
```

3. 赋值语句的右侧必须与左侧类型兼容，例如，如果变量 x 是 BankAccount（对象）类型，y 是 char 类型，则不允许使用以下赋值语句：

```
var BankAccount x;
var char y;
let x = BankAcount.new ();
let y= 100;
let x = y+1;          // 错误, 左侧是类引用，右侧是 char 类型
```

4. 用作数组索引的表达式必须求值为整数值，因此以下错误：

```
var bool b;
var Array a;
let a = Array. new (10);
let b = false;
a[b] = 10;
```

5. 一个标识符只能在同一范围内声明一次（不能对同一标识符重新声明）。这适用于所有作用域，无论它是子例程的本地作用域、类的作用域还是整个程序作用域。例如，不允许在同一程序中声明两个同名的类（尽管它们在不同的文件中声明）。
6. 子例程调用必须是可解析的，即如果类 g 中有对方法 g 的调用，则源代码中的某个地方必须有此方法的声明。
7. 被调用的子例程必须具有与其声明相同数量和类型的参数，例如以下函数调用错误：

```
function int g (int a, char b) {
//…等等
}
do g (3); //  错误，g 需要两个参数
```

8. 函数应返回与函数返回类型兼容的值，例如以下返回语句错误：

```
function int g (int a, char b)
{
   BankAccount b;
   let b = BankAccount.new ();
   return b; // b  返回类型错误
}
```

9. 函数中的所有代码路径都必须返回一个值，例如不应允许以下函数实现：

```
Function int f(in a)
{
    If (a==0)
    return 0;
//  错误, a!=0 时无返回值
}
```

10. 无法访问的代码（例如，在函数体中的非条件返回之后）：

```
Function int f(in a)
{
    If (a==0)
       return 0;
    return 1;
     let a = 4; //  错误，此代码无法被执行
}
```

8. 代码生成

代码生成语句被插入到了解析器函数的适当位置。

三、 运行环境

1. 硬件环境

本软件的编写与测试运行在作者本人的个人笔记本电脑上。由于条件限制，作者未测试在其它硬件环境下的运行效果，理论上符合运行 Python 解释器硬件条件的，有 1G 以上的 RAM 和硬盘存储空间的所有 PC 机上都能运行本程序。

作者本人个人笔记本电脑配置：

CPU: Intel Core i7-8550U 1.80GHZ
RAM: 8GB
硬盘: SSD 128GB 机械 1TB

2. 软件环境

因为本程序为 Python 3 语言编写，因而凡是支持 Python 3 解释器的操作系统都可以编译本程序。作者的编译运行环境如下：

Ubuntu 18.04（64 位）
Python 3.7.8（32 位）

在进行发布的时候，作者使用了 Pyinstaller 工具进行了打包，因为打包时使用的是 32 位的 Python，所以程序同时支持 32 位和 64 位的操作系统。

四、 技术细节

1. 词法分析器

此软件使用嵌套的 case 语句（if…elif…因为 Python 不支持 case）来实现词法有限自动机逻辑。此部分位于“lexer.py”。如果你想使用该词法分析器，可以把它作为一个库导入。根据需要，我将标识分为 10 种类型：Identifier、Integer、String、Boolean、Null、Symbol、Keyword、Operator、Method、EOF。我创建了一个类，它包括 GetNextToken（）和 PeekNextToken（）方法，Jack 源代码以列表 list 的形式存储在代码参数中，这样的列表由字符串组成，列表中的字符串是源代码文件中一行的内容。例如，code[line]表示源文件中的行字符串，而 code[line][pointer]表示指定行号中的指针字符。使用 line 参数存储已使用的行号，pointer 参数用于存储正在使用的行中指定的字符。

当第一次输入 GetNextToken（）方法时，它将判断源文件是空的还是指针或行参数超过实际大小。然后开始跳过制表符、空白和注释，直到下一个标记出现。如果使用了/*而不使用*/，lexer 将停止，并显示错误消息。最后，我们开始识别这些标记，如果一个字符串的结尾没有在一行中，或者有一个不允许的符号或一个错误的标识符，则会提示错误。如果成功，函数将首先增加指针，然后返回一个包含符号和类型的元组。

PeekNextToken（）方法先另存行号和指针号，然后执行 GetNextToken（），最后恢复行号和指针号。

我使用一个空文件，一个只有空行的文件，一个缺少*/注释的文件，以及书中给定的测试集。我的 lexer 在那些测试中表现得很好。

2. 语法分析器

此软件使用如下表达式进行了语法分析器（jcparser.py）的有限自动机逻辑构建：

```
classDeclar → class identifier { {memberDeclar} }
memberDeclar → classVarDeclar | subroutineDeclar
classVarDeclar → (static | field) type identifier {, identifier} ;
type → int | char | boolean | identifier
subroutineDeclar → (constructor | function | method) (type|void) identifier (paramList) subroutineBody
paramList → type identifier {, type identifier} | ε
subroutineBody → { {statement} }
statement → varDeclarStatement | letStatemnt | ifStatement | whileStatement | doStatement | returnStatemnt
varDeclarStatement → var type identifier { , identifier } ;
letStatemnt → let identifier [ [ expression ] ] = expression ;
ifStatement → if ( expression ) { {statement} } [else { {statement} }]
whileStatement → while ( expression ) { {statement} }
doStatement → do subroutineCall ;
subroutineCall → identifier [ . identifier ] ( expressionList )
expressionList → expression { , expression } | ε
returnStatemnt → return [ expression ] ;
expression → relationalExpression { ( & | | ) relationalExpression }
relationalExpression → ArithmeticExpression { ( = | > | < ) ArithmeticExpression }
ArithmeticExpression → term { ( + | - ) term }
term → factor { ( * | / ) factor }
factor → ( - | ~ | ε ) operand
operand → integerConstant | identifier [.identifier ] [ [ expression ] | (expressionList ) ] | (expression) | stringLiteral | true | false | null | this
```

在对应函数部分我适当插入了符号表的修改以及查询语句，语义分析功能和代码生成功能。

我还创建了 9 个用于写入生成代码的方法：writePush(), writePop(), writeArithmetic(), writeLabel(), writeGoto(), writeIf(), writeCall(), writeFunction(), writeReturn(). 它们被放置在解析器的适当部分以生成代码。

3. 符号表

符号表部分位于（SymbolTable.py）我使用 Python 中的 dictionary 字典来存储符号表。主符号表的键是一个名为 level 的列表，它表示子表的名称，并反映其层次结构。主符号表的值是使用字典存储标识符的子表。子符号表的键是标识符的名称。子符号表的值是一个列表，包括类型、类型和是否已分配标识符的布尔记录。

符号表使用主表的键来查找确切的子表。其工作原理如下。当进入一个新的作用域时，编译器将把类/函数等的名称推到 level 中，并将其用作创建新子表的键。如果作用域结束，level 将弹出它的最后一个元素以进入父作用域。我还设置了 if 和 while 编号，以帮助识别这些不同的作用域，并通过将“if/while”+编号推入 level 来创建新表。

此部分还包含了一个纯粹的语法分析器，以帮助事先建立符号表，为后面预先判断是否存在引用助力。

4. 主编译程序

编译器首先读取和编译系统库 Jack 文件获得符号表，然后将编译输入文件路径下的每个 jack 文件（同一个子文件夹中的文件共享一个符号表）。编译器会首先使用 SymbolTable 中的语法分析构建符号表，再将得到的符号表输入到完整的语法分析器获得生成的代码，如果在处理特定源文件时出现问题，编译器将放弃编译此文件，显示错误并继续编译文件夹中保留的其他文件。处理完所有文件后，编译器将在控制台上显示“Compilation Complete!”。

五、 使用方法

这里使用《计算机系统要素：从零开始构建现代计算机》中提供的以下一段源代码进行示例：

```
// This file is part of the materials accompanying the book
// "The Elements of Computing Systems" by Nisan and
Schocken,
// MIT Press. Book site: www.idc.ac.il/tecs
// File name: projects/10/ArrayTest/Main.jack

/** Computes the average of a sequence of integers. */
class Main {

    /**
     * Initializes RAM[8001]..RAM[8016] to -1, and
converts the value in
     * RAM[8000] to binary.
     */
    function void main() {
  var int result, value;

        do Main.fillMemory(8001, 16, -1); // sets
RAM[8001]..RAM[8016] to -1
        let value = Memory.peek(8000);      // reads a
value from RAM[8000]
  do Main.convert(value);                   // perform convertion

      return;
    }

    /** Converts the given decimal value to binary, and puts
     *    the resulting bits in RAM[8001]..RAM[8016]. */
    function void convert(int value) {
      var int mask, position;
      var boolean loop;

      let loop = true;

      while (loop) {
          let position = position + 1;
          let mask = Main.nextMask(mask);
            do Memory.poke(9000 + position, mask);

          if (~(position > 16)) {
```

```
                if (~((value & mask) = 0)) {
                    do Memory.poke(8000 + position, 1);
                }
                else {
                    do Memory.poke(8000 + position, 0);
                }
            }
            else {
                let loop = false;
            }
        }

        return;
    }

    /** Returns the next mask (the mask that should follow
the given mask). */
    function int nextMask(int mask) {
        if (mask = 0) {
            return 1;
        }
        else {
        return mask * 2;
        }
    }

    /** Fills 'length' consecutive memory locations with
'value',
      * starting at 'startAddress'. */
    function void fillMemory(int startAddress, int length, int
value) {
        while (length > 0) {
            do Memory.poke(startAddress, value);
            let length = length - 1;
            let startAddress = startAddress + 1;
        }

        return;
    }
}
```

在 test 目录下放置该测试用文件 Main.jack，在终端使用 Python 3 运行：

```
D:\JackCompiler>python myjc.py test

Compiling System library: syslib\Array.jack
Compilation success

Compiling System library: syslib\Keyboard.jack
Compilation success

Compiling System library: syslib\Math.jack
Compilation success

Compiling System library: syslib\Memory.jack
Compilation success

Compiling System library: syslib\Output.jack
Compilation success

Compiling System library: syslib\Screen.jack
Compilation success

Compiling System library: syslib\String.jack
Compilation success

Compiling System library: syslib\Sys.jack
Compilation success

Compiling test\Main.jack
Compilation success

Compilation Complete! Proceed 1 files in total.
```

在 test 文件夹生成了以下 Main.vm 代码：

```
function Main.main 2
push constant 8001
push constant 16
push constant 1
neg
call Main.fillMemory 3
pop temp 0
push constant 8000
call Memory.peek 1
pop local 1
push local 1
call Main.convert 1
pop temp 0
push constant 0
return
function Main.convert 3
push constant 0
not
pop local 2
label WHILE_EXP0
push local 2
```

```
not
if-goto WHILE_END0
push local 1
push constant 1
add
pop local 1
push local 0
call Main.nextMask 1
pop local 0
push constant 9000
push local 1
add
push local 0
call Memory.poke 2
pop temp 0
push local 1
push constant 16
gt
not
if-goto IF_TRUE0
goto IF_FALSE0
label IF_TRUE0
push argument 0
push local 0
and
push constant 0
eq
not
if-goto IF_TRUE1
goto IF_FALSE1
label IF_TRUE1
push constant 8000
push local 1
add
push constant 1
call Memory.poke 2
pop temp 0
goto IF_END1
label IF_FALSE1
push constant 8000
push local 1
add
push constant 0
call Memory.poke 2
```

```
pop temp 0
label IF_END1
goto IF_END0
label IF_FALSE0
push constant 0
pop local 2
label IF_END0
goto WHILE_EXP0
label WHILE_END0
push constant 0
return
function Main.nextMask 0
push argument 0
push constant 0
eq
if-goto IF_TRUE0
goto IF_FALSE0
label IF_TRUE0
push constant 1
return
goto IF_END0
label IF_FALSE0
push argument 0
push constant 2
call Math.multiply 2
return
label IF_END0
function Main.fillMemory 0
label WHILE_EXP0
push argument 1
push constant 0
gt
not
if-goto WHILE_END0
push argument 0
push argument 2
call Memory.poke 2
pop temp 0
push argument 1
push constant 1
sub
pop argument 1
push argument 0
push constant 1
```

```
add
pop argument 0
goto WHILE_EXP0
label WHILE_END0
push constant 0
return
```

此代码与原书中提供的 vm 示例代码没有差异，从一方面证明了生成代码功能的正确性，接下来可以将该 vm 代码放入 Jack 虚拟机中进行运行了。（请参考 https://www.nand2tetris.org/software 或者《计算机系统要素：从零开始构建现代计算机》书本）