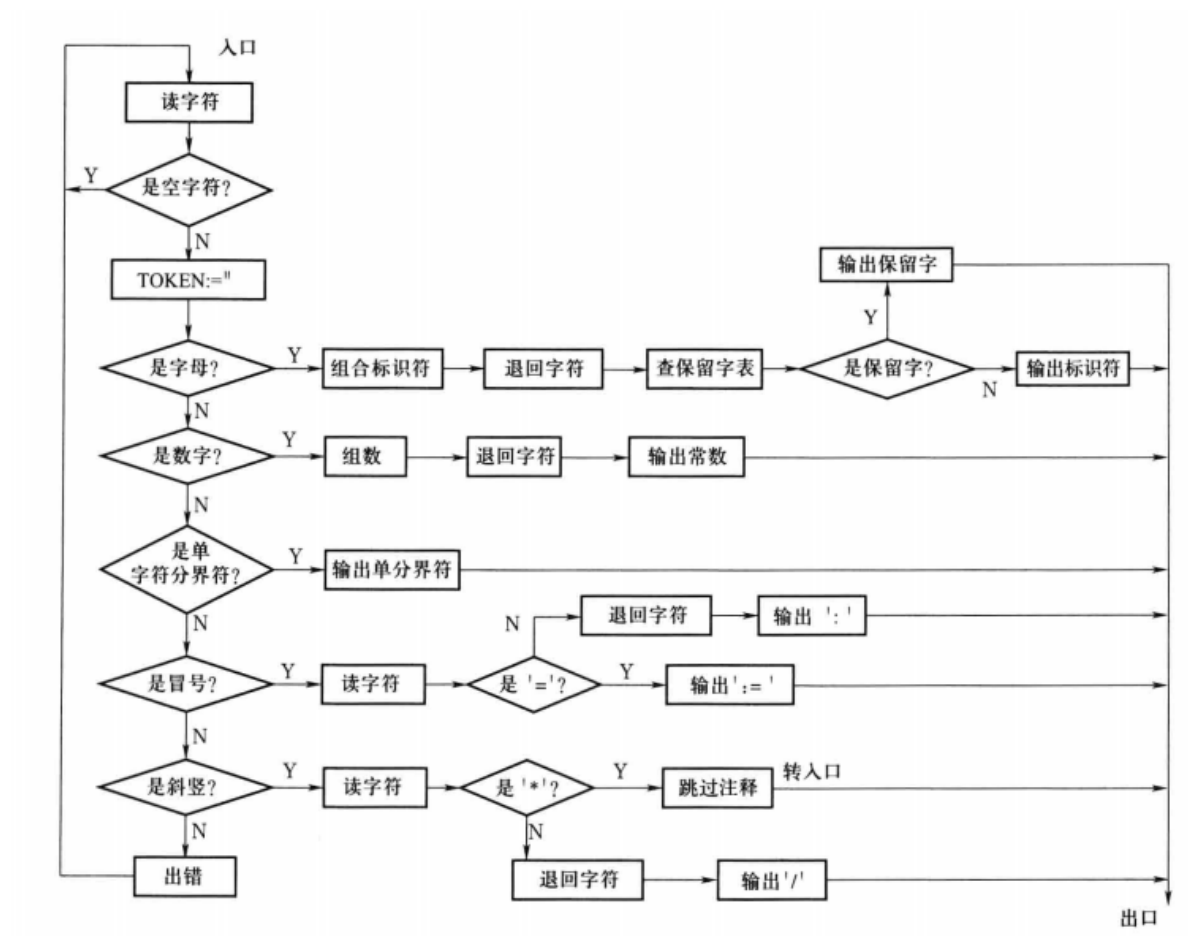


词法分析：

实现方式： 定义一个 `string` 类型的 `token` 变量，一个 `string` 类型的 `symbol` 变量，它们分别表示单词符和类型。在 `main` 函数中，是一个无限的 `while` 循环，用来获取输入的字符串并进行词法分析，最后输出 `token` 和对应类型。在 `getsym()` 中，每次读取一个输入的字符，并对字符的类型进行判断后与 `token` 进行拼接。

代码流程图：



输入和结果：

*test1.txt - 记事本

文件(E) 编辑(E) 格式(O) 查看(V)

```
int main(){
int c;
c = getint();
printf("%d",c);
return c;
}
```

D:\VS\Project\Project258\x64\Debug\Project258.exe

```
int main() {
INTTK int
MAINTK main
LPARENT (
RPARENT )
LBRACE {
int c;
INTTK int
IDENFR c
SEMISY ;
c = getint();
IDENFR c
ASSIGN =
GETINTTK getint
LPARENT (
RPARENT )
SEMISY ;
printf("%d",c);
PRINTFTK printf
LPARENT (
STRCON "%d"
COMMSY ,
IDENFR c
RPARENT )
SEMISY ;
return c;
RETURNTK return
IDENFR c
SEMISY ;
}
RBRACE }
```

```
int main(){
int a ,b = 32, c = 33, d;
if(c>b) a = 0;
else a = 1;
d = getint();
printf("%d",a); //打印c
return c;
}
```

```
int main() {
INTTK int
MAINTK main
LPARENT (
RPARENT )
LBRACE {
int a ,b = 32, c = 33, d;
INTTK int
IDENFR a
COMMSY ,
IDENFR b
ASSIGN =
INTCON 32
COMMSY ,
IDENFR c
ASSIGN =
INTCON 33
COMMSY ,
IDENFR d
SEMISY ;
if(c>b) a = 0;
IFTK if
LPARENT (
IDENFR c
GRE >
IDENFR b
RPARENT )
IDENFR a
ASSIGN =
INTCON 0
SEMISY ;
else a = 1;
ELSETK else
IDENFR a
ASSIGN =
INTCON 1
SEMISY ;
d = getint();
IDENFR d
ASSIGN =
GETINTTK getint
LPARENT (
RPARENT )
}
```

```

SEMISY ;
printf("%d",a); //打印c
PRINTFTK printf
LPARENT (
STRCON "%d"
COMMSY ,
IDENFR a
RPARENT )
SEMISY ;
return c;
RETURNTK return
IDENFR c
SEMISY ;
}
RBRACE }

```

源码:

```

1  #include <iostream>
2  #include <vector>
3  #include <string>
4  #include <unordered_map>
5  using namespace std;
6
7  char ch, chl;
8  bool lch;
9  string token;
10 string symbol;
11 int num;
12
13 unordered_map<string, string> mp = {
14     {"main", "MAINTK"},
15     {"const", "CONSTTK"},
16     {"int", "INTTK"},
17     {"break", "BREAKTK"},
18     {"continue", "CONTINUETK"},
19     {"if", "IFTK"},
20     {"else", "ELSETK"},
21     {"!", "NOT"},
22     {"&&", "ADN"},
23     {"||", "OR"},
24     {"while", "WHILETK"},
25     {"getint", "GETINTTK"},
26     {"printf", "PRINTFTK"},
27     {"return", "RETURNTK"},
28     {"+", "PLUS"},
29     {"-", "MINUS"},
30     {"void", "VOIDTK"},
31     {"*", "MULT"},
32     {"/", "DIV"},
33     {"%", "MOD"},
34     {"<", "LSS"},
35     {"<=", "LEQ"},

```

```
36     {">", "GRE"},
37     {">=", "GEQ"},
38     {"==", "EQL"},
39     {"!=", "NEQ"},
40     {"=", "ASSIGN"},
41     {";", "SEMICN"},
42     {"", ",", "COMMA"},
43     {"(", "LPARENT"},
44     {"")", "RPARENT"},
45     {"[", "LBRACK"},
46     {"]", "RBRACK"},
47     {"{", "LBRACE"},
48     {"}", "RBRACE"},
49 };
50
51 bool isSpace() // 判断是否为空格
52 {
53     return ch == ' ';
54 }
55
56 bool isNewline() // 判断是否为换行符
57 {
58     return ch == '\n';
59 }
60
61 bool isTab() // 判断是否为Tab
62 {
63     return ch == '\t';
64 }
65
66 bool isLetter() // 判断是否为字母
67 {
68     if ((ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z'))
69         return true;
70     else
71         return false;
72 }
73
74 bool isDigit() // 判断是否为数字
75 {
76     if (ch >= '0' && ch <= '9')
77         return true;
78     else
79         return false;
80 }
81
82 bool isColon() // 判断是否为冒号
83 {
84     return ch == ':';
85 }
86
87 bool isComma() // 判断是否为逗号
88 {
89     return ch == ',';
90 }
```

```
91
92 bool isSemi() // 判断是否为分号
93 {
94     return ch == ';';
95 }
96
97 bool isEqu() // 判断是否为等号
98 {
99     return ch == '=';
100 }
101
102 bool isPlus() // 判断是否为加号
103 {
104     return ch == '+';
105 }
106
107 bool isMinus() // 判断是否为减号
108 {
109     return ch == '-';
110 }
111
112 bool isDivi() // 判断是否为除号
113 {
114     return ch == '/';
115 }
116
117 bool isStar() // 判断是否为星号（乘*）
118 {
119     return ch == '*';
120 }
121
122 bool isDouble()
123 {
124     return ch == '"';
125 }
126
127 void getsym()
128 {
129     token = "";
130     symbol = "";
131
132     // 读入字符，通过lch判断上一次有没有多读的字符
133     if (!lch)
134         ch = getchar();
135     else
136     {
137         ch = chl;
138         lch = 0;
139     }
140
141     while (isSpace() || isNewline() || isTab())
142         ch = getchar();
143     if (isLetter())
144     {
145         while (isLetter() || isDigit())
```

```
146     {
147         token += ch;
148         ch = getchar(); // 会多读一个字符
149     }
150
151     // 处理多读的字符
152     chl = ch;
153     lch = 1;
154
155     if (mp.count(token))
156         symbol = mp[token];
157     else
158         symbol = "IDENFR";
159 }
160 else if (isDigit())
161 {
162     num = 0;
163     while (isDigit())
164     {
165         token += ch;
166         ch = getchar(); // 多读一个字符
167     }
168
169     // 处理多读的字符
170     chl = ch;
171     lch = 1;
172
173     num = stoi(token);
174     symbol = "INTCON";
175 }
176 else if (isColon())
177 {
178     token += ch;
179     symbol = "COLON";
180 }
181 else if (isPlus())
182 {
183     token += ch;
184     symbol = "PLUS";
185 }
186 else if (isMinus())
187 {
188     token += ch;
189     symbol = "MINUS";
190 }
191 else if (isStar())
192 {
193     token += ch;
194     symbol = "MULT";
195 }
196 else if (isComma())
197 {
198     token += ch;
199     symbol = "COMMA";
200 }
```

```
201     else if (isSemi())
202     {
203         token += ch;
204         symbol = "SEMICN";
205     }
206     else if (ch == '>')
207     {
208         char ch_ = ch;
209         token += ch;
210         ch = getchar();
211         if (ch == '=')
212         {
213             token += ch;
214             symbol = mp[token];
215         }
216         else
217         {
218             // 处理多读的字符
219             chl = ch;
220             lch = 1;
221             symbol = mp[token];
222         }
223     }
224     else if (ch == '<')
225     {
226         char ch_ = ch;
227         token += ch;
228         ch = getchar();
229         if (ch == '=')
230         {
231             token += ch;
232             symbol = mp[token];
233         }
234         else
235         {
236             // 处理多读的字符
237             chl = ch;
238             lch = 1;
239             symbol = mp[token];
240         }
241     }
242     else if (ch == '=')
243     {
244         char ch_ = ch;
245         token += ch;
246         ch = getchar();
247         if (ch == '=')
248         {
249             token += ch;
250             symbol = mp[token];
251         }
252         else
253         {
254             // 处理多读的字符
255             chl = ch;
```



```
256         lch = 1;
257         symbol = mp[token];
258     }
259 }
260 else if (ch == '&')
261 {
262     token += ch;
263     ch = getchar();
264     symbol = mp["&"];
265 }
266 else if (ch == '|')
267 {
268     token += ch;
269     ch = getchar();
270     symbol = mp["||"];
271 }
272 else if (ch == '(')
273 {
274     token += ch;
275     symbol = mp[token];
276 }
277 else if (ch == ')')
278 {
279     token += ch;
280     symbol = mp[token];
281 }
282 else if (ch == '[' || ch == '{')
283 {
284     token += ch;
285     symbol = mp[token];
286 }
287 else if (ch == ']' || ch == '}')
288 {
289     token += ch;
290     symbol = mp[token];
291 }
292 else if (isDivi())
293 {
294     char ch_ = ch;
295     ch = getchar();
296     if (isstar()) // 判断注释
297     {
298         while (ch != '\n')
299             ch = getchar();
300     }
301     else if (isDivi())
302     {
303         while (ch != '\n')
304             ch = getchar();
305     }
306     else
307     {
308         token += ch_;
309         symbol = "DIVI";
310     }
```

```

311     }
312     else if (isDouble())
313     {
314         do
315         {
316             token += ch;
317             ch = getchar();
318         } while (!isDouble());
319         token += ch;
320         symbol = "STRCON";
321     }
322 }
323
324 int main()
325 {
326     while (1)
327     {
328         getsym();
329         if (symbol == "INTCON")
330             cout << symbol << " " << num << endl;
331         else if (ch != '\n') // 处理到注释时，ch最终读到的是\n，此时symbol和
token都是空，所以要跳过
332             cout << symbol << " " << token << endl;
333     }
334     return 0;
335 }

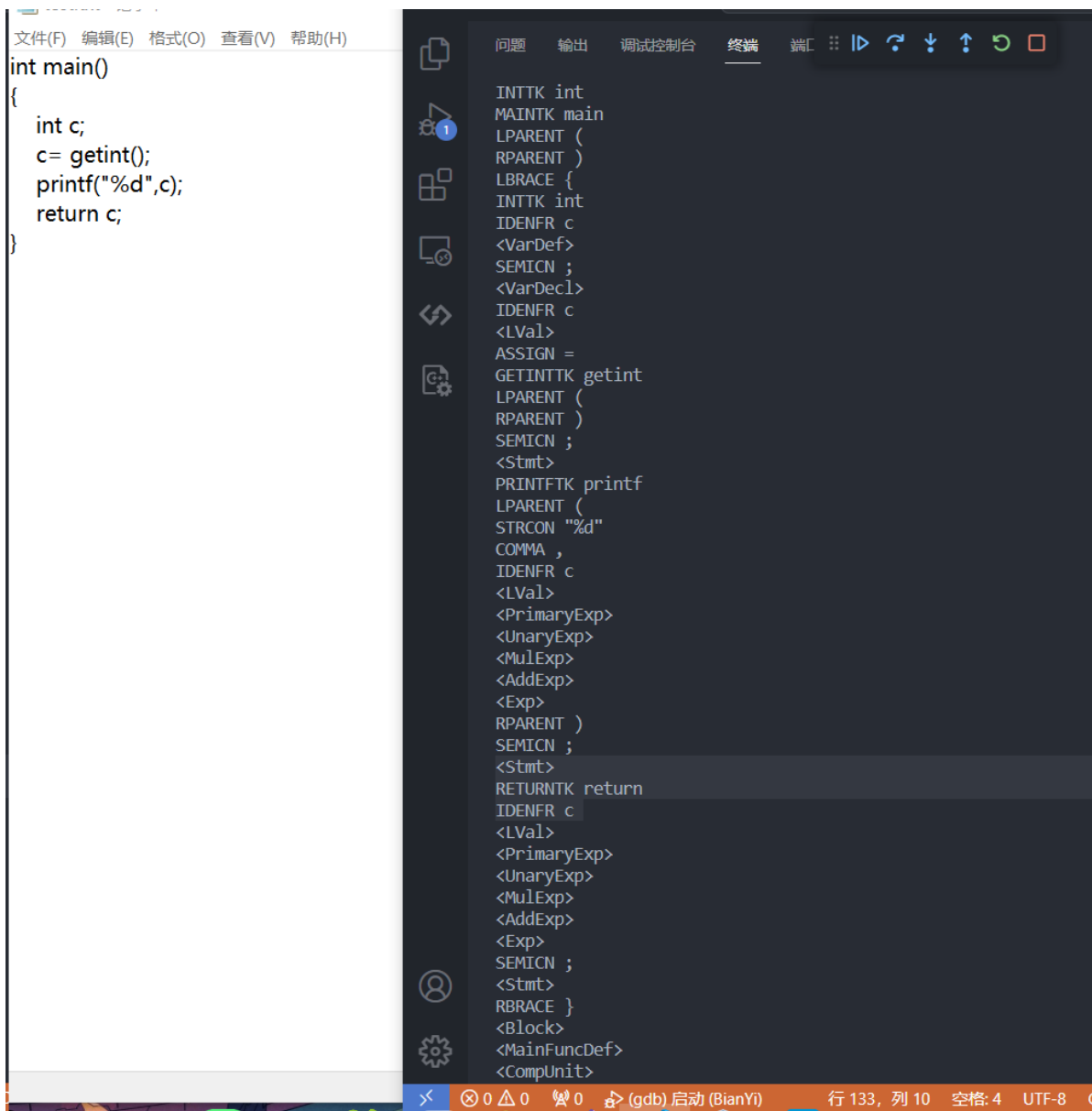
```

语法分析

实现方法：首先对整个输入程序进行词法分析，将得到的每个token等信息用链表连接。之后在语法分析时对链表从头到尾的顺序进行分析。

说明：语法分析中用到的词法分析代码和上面不同，因为一开始写词法分析的时候就每个token都getchar获取，但这种操作是不可逆的，不利于语法分析时的向前试探和预判，所以在语法分析程序中就不再使用上述代码，但大部分内容一样。

输入和结果：



```
1  #include <iostream>
2  #include <vector>
3  #include <string>
4  #include <fstream>
5  #include <sstream>
6  #include <cctype>
7  #include <algorithm>
8  #include <unordered_map>
9  #include "head.h"
10 using namespace std;
11
12 struct Node
13 {
14     string Tok;
15     string sym;
16     int tnum;
17     struct Node *next;
18     struct Node *pre;
19 };
20 struct Node *head = NULL;
```

```

21 struct Node *nowtok;          // 用于语法分析
22 char ch, chl, chne = '@'; // ch是现在要分析的字符, chl是多读到的字符, nech是下一个
    待分析的字符
23 bool lch;
24 string fileContent;
25 string token;
26 string symbol;
27 int num, idx;
28
29 unordered_map<string, string> mp = {
30     {"main", "MAINTK"},
31     {"const", "CONSTTK"},
32     {"int", "INTTK"},
33     {"break", "BREAKTK"},
34     {"continue", "CONTINUETK"},
35     {"if", "IFTK"},
36     {"else", "ELSETK"},
37     {"!", "NOT"},
38     {"&&", "ADN"},
39     {"||", "OR"},
40     {"while", "WHILETK"},
41     {"getint", "GETINTTK"},
42     {"printf", "PRINTFK"},
43     {"return", "RETURNK"},
44     {"+", "PLUS"},
45     {"-", "MINUS"},
46     {"void", "VOIDTK"},
47     {"*", "MULT"},
48     {"/", "DIV"},
49     {"%", "MOD"},
50     {"<", "LSS"},
51     {"<=", "LEQ"},
52     {">", "GRE"},
53     {">=", "GEQ"},
54     {"==", "EQL"},
55     {"!=", "NEQ"},
56     {"=", "ASSIGN"},
57     {";", "SEMICN"},
58     {"", "COMMA"},
59     {"(", "LPARENT"},
60     {")", "RPARENT"},
61     {"[", "LBRACK"},
62     {"]", "RBRACK"},
63     {"{", "LBRACE"},
64     {"}", "RBRACE"},
65 };
66
67 void go()
68 {
69     nowtok = nowtok->next;
70     if (nowtok->sym == "INTCON")
71         cout << nowtok->sym << " " << nowtok->tnum << endl;
72     else
73         cout << nowtok->sym << " " << nowtok->Tok << endl;
74 }

```

```

75
76 void appendNode(struct Node **head)
77 {
78     // 创建新节点
79     // struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
80     struct Node *newNode = new struct Node;
81     if (newNode == NULL)
82         return;
83     newNode->Tok = token;
84     newNode->sym = symbol;
85     newNode->tnum = num;
86     newNode->next = NULL;
87     newNode->pre = NULL;
88
89     // 如果链表为空，新节点即为头节点
90     if (*head == NULL)
91     {
92         *head = newNode;
93     }
94     else
95     {
96         // 否则，遍历链表找到末尾，将新节点连接到末尾
97         struct Node *current = *head;
98         while (current->next != NULL)
99         {
100             current = current->next;
101         }
102         current->next = newNode;
103         newNode->pre = current;
104     }
105 }
106
107 void freeList(struct Node *head)
108 {
109     struct Node *current = head;
110     struct Node *next;
111
112     while (current != NULL)
113     {
114         next = current->next;
115         delete (current);
116         current = next;
117     }
118 }
119
120 void compUnit()
121 {
122     nowtok = head;
123     if (nowtok->sym == "INTCON")
124         cout << nowtok->sym << " " << nowtok->tnum << endl;
125     else
126         cout << nowtok->sym << " " << nowtok->Tok << endl;
127
128     while (1)
129     {

```

```

130         if (nowtok->sym == "CONSTTK")
131         {
132             Decl();
133         }
134         else if (nowtok->sym == "VOIDTK") // 变量定义
135         {
136             FuncDef();
137         }
138         else // INTTK
139         {
140             go();
141             if (nowtok->sym == "MAINTK") // 主函数
142             {
143                 MainFuncDef();
144                 break;
145             }
146             else if (nowtok->sym == "IDENFR") // 变量定义
147             {
148                 Decl();
149             }
150         }
151     }
152     cout << "<CompUnit>" << endl;
153 }
154
155 void Decl()
156 {
157     if (nowtok->sym == "CONSTTK")
158     {
159         ConstDecl();
160     }
161     else
162     {
163         VarDecl();
164     }
165     // cout << "<Decl>" << endl;
166 }
167
168 void MainFuncDef()
169 {
170     go(); // 取(
171     go(); // 取)
172     Block();
173     cout << "<MainFuncDef>" << endl;
174 }
175
176 void ConstDecl()
177 {
178     BType();
179     ConstDef();
180     // 判断是否定义多变量
181     while (nowtok->next->sym == "COMMA")
182     {
183         ConstDef();
184         go(); // 最后一次读到分号;

```

```

185     }
186     cout << "<ConstDecl>" << endl;
187 }
188
189 void VarDecl()
190 {
191     BType();
192     varDef();
193     while (nowtok->next->sym == "COMMA")
194     {
195         varDef();
196     }
197     go(); // 取;
198     cout << "<VarDecl>" << endl;
199 }
200
201 void BType()
202 {
203     if (nowtok->sym != "INTTK")
204         go(); // 获取int
205     // cout << "<BType>" << endl;
206 }
207
208 void ConstDef()
209 {
210     go(); // 获取Ident
211     go(); // 获取等号, 假设不是数组
212     ConstInitVal();
213     cout << "<ConstDef>" << endl;
214 }
215 void Ident() {}
216 void ConstExp()
217 {
218     AddExp(); // CONST
219     cout << "<ConstExp>" << endl;
220 }
221 void ConstInitVal()
222 {
223     ConstExp();
224     cout << "<ConstInitVal>" << endl;
225 }
226 void VarDef()
227 {
228     go(); // 取变量名
229     if (nowtok->next->sym == "ASSIGN")
230         InitVal();
231     cout << "<VarDef>" << endl;
232 }
233 void InitVal()
234 {
235     Exp();
236     cout << "<InitVal>" << endl;
237 }
238 void Exp()
239 {

```

```

240     AddExp();
241     cout << "<Exp>" << endl;
242 }
243 void FuncDef()
244 {
245     // FuncDef → FuncType Ident '(' [FuncFParams] ')' Block
246     FuncType();
247     go(); // 获取函数名Ident
248     go(); // 获取左括号(
249
250     if (nowtok->next->sym != "RPARENT") // 有参数
251     {
252         FuncFParams();
253     }
254     else
255     {
256         go(); // 取)
257         Block();
258     }
259     cout << "<FuncDef>" << endl;
260 }
261
262 void FuncType()
263 {
264     // FuncType → 'void' | 'int'
265     go(); // 取函数类型
266     cout << "<FuncType>" << endl;
267 }
268
269 void FuncFParams()
270 {
271     FuncFParam();
272     cout << "<FuncFParams>" << endl;
273 }
274
275 void FuncFParam()
276 {
277     BType();
278     cout << "<FuncFParam>" << endl;
279 }
280 void Block()
281 {
282     go(); // 取左花括号{
283     do
284     {
285         BlockItem();
286     } while (nowtok->next->sym != "RBRACE");
287     go(); // 取右花括号}
288     cout << "<Block>" << endl;
289 }
290 void BlockItem()
291 {
292     if (nowtok->next->sym == "CONSTTK" || nowtok->next->sym == "INTTK")
293     {
294         if (nowtok->next->sym == "CONSTTK")

```



```

295         ConstDecl();
296     else
297         VarDecl();
298 }
299 else
300 {
301     Stmt();
302 }
303 // cout << "<BlockItem>" << endl;
304 }
305 void Stmt()
306 {
307     if (nowtok->next->sym == "IDENFR")
308     {
309         // Lval '=' Exp '; '
310         Lval();
311         go(); // 取=
312         if (nowtok->next->sym == "GETINTTK")
313         {
314             go(); // 取 getint
315             go(); // 取(
316             go(); // 取)
317             go(); // 取;
318         }
319         else
320         {
321             Exp();
322             go(); // 取;
323         }
324     }
325     else if (nowtok->next->sym == "IFTK")
326     {
327         go(); // 取 if
328         go(); // 取(
329         Cond();
330         go(); // 取)
331         if (nowtok->next->sym == "ELSETK") // 判断有没有else
332         {
333             go(); // 取 else
334             Stmt();
335         }
336         else
337             Stmt();
338     }
339     else if (nowtok->next->sym == "WHILETK")
340     {
341         go(); // 取 while
342         go(); // 取(
343         Cond();
344         go(); // 取)
345         Stmt();
346     }
347     else if (nowtok->next->sym == "BREAKTK")
348     {
349         go(); // 取 break

```

```

350         go(); // 取;
351     }
352     else if (nowtok->next->sym == "CONTINUETK")
353     {
354         go(); // 取 continue
355         go(); // 取;
356     }
357     else if (nowtok->next->sym == "RETURNTK")
358     {
359         //'return' [Exp] ';'
360         go(); // 取 return
361         if (nowtok->next->sym != "SEMICN")
362             Exp();
363         go(); // 取;
364     }
365     else if (nowtok->next->sym == "PRINTFTK")
366     {
367         go(); // 取 printf
368         go(); // 取(
369         go(); // 取一个字符串
370         while (nowtok->next->sym == "COMMA") // 有参数
371         {
372             go(); // 取 ,
373             Exp();
374         }
375         go(); // 取 )
376         go(); // 取 ;
377     }
378     else if (nowtok->next->sym == "LBRACE") // BLOCK情况
379     {
380         Block();
381     }
382     else //[Exp] ';'
383     {
384         if (nowtok->next->sym != "SEMICN")
385             Exp();
386         go(); // 取;
387     }
388     cout << "<Stmt>" << endl;
389 }
390 void LVal()
391 {
392     if (nowtok->sym != "IDENFR") // 看是否已经取到了变量名
393         go();
394     cout << "<LVal>" << endl;
395 }
396 void Cond()
397 {
398 }
399 void FormatString() {}
400 void AddExp()
401 {
402     MulExp();
403     cout << "<AddExp>" << endl;
404 }

```

```

405 void LOrExp() {}
406 void PrimaryExp()
407 {
408     if (nowtok->next->sym == "IDENFR")
409         LVal();
410     else
411         Number();
412     cout << "<PrimaryExp>" << endl;
413 }
414 void Number()
415 {
416     IntConst();
417     cout << "<Number>" << endl;
418 }
419 void IntConst()
420 {
421     go(); // 获取一个整数
422     cout << "<IntConst>" << endl;
423 }
424 void UnaryExp()
425 {
426     PrimaryExp();
427     cout << "<UnaryExp>" << endl;
428 }
429
430 void UnaryOp() {}
431 void FuncRParams() {}
432 void MulExp()
433 {
434     UnaryExp();
435     cout << "<MulExp>" << endl;
436 }
437 void RelExp() {}
438 void EqExp() {}
439 void LAndExp() {}
440 void identifier() {}
441 void identifier_nondigit() {}
442 void digit() {}
443 void integer_const() {}
444 void decimal_const() {}
445 void nonzero_digit() {}
446
447 bool isSpace() // 判断是否为空格
448 {
449     return ch == ' ';
450 }
451
452 bool isNewline() // 判断是否为换行符
453 {
454     return ch == '\n';
455 }
456
457 bool isTab() // 判断是否为Tab
458 {
459     return ch == '\t';

```

```
460 }
461
462 bool isLetter() // 判断是否为字母
463 {
464     if ((ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z'))
465         return true;
466     else
467         return false;
468 }
469
470 bool isDigit() // 判断是否为数字
471 {
472     if (ch >= '1' && ch <= '9')
473         return true;
474     else
475         return false;
476 }
477
478 bool isColon() // 判断是否为冒号
479 {
480     return ch == ':';
481 }
482
483 bool isComma() // 判断是否为逗号
484 {
485     return ch == ',';
486 }
487
488 bool isSemi() // 判断是否为分号
489 {
490     return ch == ';';
491 }
492
493 bool isEqu() // 判断是否为等号
494 {
495     return ch == '=';
496 }
497
498 bool isPlus() // 判断是否为加号
499 {
500     return ch == '+';
501 }
502
503 bool isMinus() // 判断是否为减号
504 {
505     return ch == '-';
506 }
507
508 bool isDivi() // 判断是否为除号
509 {
510     return ch == '/';
511 }
512
513 bool isStar() // 判断是否为星号（乘*）
514 {
```

```
515     return ch == '*';
516 }
517
518 bool isDouble()
519 {
520     return ch == '.';
521 }
522
523 void Lexical()
524 {
525
526     token = "";
527     symbol = "";
528     num = 0;
529
530     ch = fileContent[idx];
531     while (isspace() || isnewline() || istab())
532         ch = fileContent[++idx];
533     if (isLetter())
534     {
535         while (isLetter() || isDigit())
536         {
537             token += ch;
538             ch = fileContent[++idx]; // 会多读一个字符
539         }
540         idx--; // 减去多读的一个
541         if (mp.count(token))
542             symbol = mp[token];
543         else
544             symbol = "IDENFR";
545     }
546     else if (isDigit())
547     {
548         num = 0;
549         while (isDigit())
550         {
551             token += ch;
552             ch = fileContent[++idx]; // 多读一个字符
553         }
554         idx--; // 减去多读的一个
555         num = stoi(token);
556         symbol = "INTCON";
557     }
558     else if (isColon())
559     {
560         token += ch;
561         symbol = "COLON";
562     }
563     else if (isPlus())
564     {
565         token += ch;
566         symbol = "PLUS";
567     }
568     else if (isMinus())
569     {
```

```
570         token += ch;
571         symbol = "MINUS";
572     }
573     else if (isStar())
574     {
575         token += ch;
576         symbol = "MULT";
577     }
578     else if (isComma())
579     {
580         token += ch;
581         symbol = "COMMA";
582     }
583     else if (isSemi())
584     {
585         token += ch;
586         symbol = "SEMICN";
587     }
588     else if (ch == '>')
589     {
590         char ch_ = ch;
591         token += ch;
592
593         if (ch == '=')
594         {
595             token += ch;
596             symbol = mp[token];
597         }
598         else
599         {
600             // 处理多读的字符
601             chl = ch;
602             lch = 1;
603             symbol = mp[token];
604         }
605     }
606     else if (ch == '<')
607     {
608         char ch_ = ch;
609         token += ch;
610         ch = fileContent[idx + 1];
611         if (ch = fileContent[idx + 1] == '=')
612         {
613             token += ch;
614             symbol = mp[token];
615         }
616         else
617         {
618             symbol = mp[token];
619         }
620     }
621     else if (ch == '=')
622     {
623         char ch_ = ch;
624         token += ch;
```

```
625     ch = fileContent[idx + 1];
626     if (ch = fileContent[idx + 1] == '=')
627     {
628         token += ch;
629         symbol = mp[token];
630     }
631     else
632     {
633         symbol = mp[token];
634     }
635 }
636 else if (ch == '&')
637 {
638     token += ch;
639     idx++;
640     symbol = mp["&"];
641 }
642 else if (ch == '|')
643 {
644     token += ch;
645     idx++;
646     symbol = mp["||"];
647 }
648 else if (ch == '(')
649 {
650     token += ch;
651     symbol = mp[token];
652 }
653 else if (ch == ')')
654 {
655     token += ch;
656     symbol = mp[token];
657 }
658 else if (ch == '[' || ch == '{')
659 {
660     token += ch;
661     symbol = mp[token];
662 }
663 else if (ch == ']' || ch == '}')
664 {
665     token += ch;
666     symbol = mp[token];
667 }
668 else if (isDivi())
669 {
670     char ch_ = ch;
671     ch = fileContent[idx + 1];
672     if (isStar()) // 判断注释
673     {
674         while (ch != '\n')
675         {
676             ch = fileContent[++idx];
677         }
678         idx--;
679     }
```

```

680         else if (isDivi())
681         {
682             while (ch != '\n')
683             {
684                 ch = fileContent[++idx];
685             }
686             idx--;
687         }
688         else
689         {
690             token += ch_;
691             symbol = "DIV";
692         }
693     }
694     else if (isDouble())
695     {
696         do
697         {
698             token += ch;
699             ch = fileContent[++idx];
700         } while (!isDouble());
701         token += ch;
702         symbol = "STRCON";
703     }
704     // if (symbol == "INTCON")
705     //     cout << symbol << " " << num << endl;
706     // else if (ch != '\n') //处理到注释时, ch最终读到的是\n, 此时symbol和token都
是空, 所以要跳过
707     //     cout << symbol << " " << token << endl;
708 }
709
710 int main()
711 {
712     std::ifstream inputFile("test.txt");
713
714     if (inputFile.is_open())
715     {
716         string line;
717
718         // 逐行读取文件内容
719         while (getline(inputFile, line))
720         {
721             // 去除每一行的空格和制表符
722             // line.erase(remove_if(line.begin(), line.end(), ::isspace),
line.end());
723             fileContent += line;
724         }
725
726         inputFile.close();
727     }
728     // 词法分析
729     for (; idx < fileContent.size(); idx++)
730     {
731         Lexical();
732         appendNode(&head); // 将token添加到链表末尾

```



```
733     }  
734     CompUnit();  
735     freeList(head);  
736     return 0;  
737 }  
738
```