

一、扩展L24文法:实现了数组和结构体，变量使用需要先声明

注：指针和带参数嵌套过程也在此文法中，但鉴于能力有限，没有实现对应的编译器代码。

```
1  <program> = "main" "{" [变量说明部分][array_decl][record_decl][pointer_decl]
   [proc_decl]<stmt_list> "}"
2
3  <stmt_list> = {<stmt> ";" }
4
5  <stmt> = <assign_stmt> | <if_stmt> | <while_stmt> | <scan_stmt> |
   <print_stmt> | <proc_call>
6
7  <assign_stmt> = <ident> "=" <expr> | <array_access> "=" <expr> |
   <record_access> "=" <expr> | "&" <ident> "=" <expr> | <ident> "=" "&" <ident>
8
9  <if_stmt> = "if" "(" <bool_expr> ")" "then" "{" <stmt_list> "}" "end"
10           | "if" "(" <bool_expr> ")" "then" "{" <stmt_list> "}" "else" "{"
   <stmt_list> "}" "end"
11
12 <while_stmt> = "while" "(" <bool_expr> ")" "{" <stmt_list> "}"
13 <scan_stmt> = "scan" "(" <ident> { "," <ident> } ")"
14 <print_stmt> = "print" "(" <expr> { "," <expr> } ")"
15 <proc_call> = CALL <标识符> (<ident> ', ' <ident>)
16 <bool_expr> = <expr> ("==" | "!=" | "<" | "<=" | ">" | ">=") <expr>
17 <expr> = ["+" | "-"] <term> { ("+" | "-") <term> }
18 <term> = <factor> { ("*" | "/" ) <factor> }
19 <factor> = <ident> | <number> | "(" <expr> ")" | <array_access> |
   <record_access> | "&" <ident>
20 <array_access> = <ident> "[" <expr> "]" "[" <expr> "]" "[" <expr> "]"
21 <record_access> = <ident> "." <ident>
22
23 /*声明部分*/
24 <变量说明部分> = "VAR" <ident> { "," <ident> } ";"
25 <array_decl> = "array" <ident> "[" <number> "]" "[" <number> "]" "["
   <number> "]" ";"
26 <record_decl> = "record" <ident> "{" { <ident> ";" } "}" ";"
27 <pointer_decl> = "pointer" <ident> ";"
28 <proc_decl> = "procedure" <ident> "(" <ident> { "," <ident> } ")" ":"
   <ident> //最后ident为返回变量
29
30 <ident> = <字母> { <字母> | <数字> }
31 <number> = <数字> { <数字> }
32
```

文法具体定义

程序结构

```
1 <program> ::= "main" "{" <decl_list> <stmt_list> "}"
```

声明列表

```
1 <decl_list> ::= { <var_decl> | <array_decl> | <record_decl> | <pointer_decl> |  
  <proc_decl> }
```

变量声明

```
1 <var_decl> ::= "var" <ident> { "," <ident> } ";"
```

数组声明

```
1 <array_decl> ::= "array" <ident> "[" <number> "]" "[" <number> "]" "["  
  <number> "]" { "," <ident> } ";"
```

数组访问

```
1 <array_access> = <ident> "[" <expr> "]" "[" <expr> "]" "[" <expr> "]"
```

记录声明

```
1 <record_decl> ::= "record" <ident> "{" { <ident>";" } "}" ";"
```

记录访问

```
1 <record_access> = <ident> "." <ident>
```

指针声明

```
1 <pointer_decl> ::= "pointer" <ident> ";"
```

过程声明

```
1 <proc_decl> = "procedure" <ident> "(" <ident> { "," <ident> } ")" ":"<ident>//  
  最后<ident>为返回变量
```

过程调用

```
1 <proc_call> = CALL <标识符>(<ident>','<ident>)
```

语句列表

```
1 <stmt_list> ::= { <stmt> ;}
```

语句

```
1 <stmt> ::= <assign_stmt> | <if_stmt> | <while_stmt> | <scan_stmt> |  
  <print_stmt> | <proc_call>
```

赋值语句

```
1 <assign_stmt> ::= <ident> "=" <expr>  
2               | <array_access> "=" <expr>  
3               | <record_access> "=" <expr>  
4               | "&" <ident> "=" <expr>  
5               | <ident> "=" "&" <ident>
```

数组访问

```
1 <array_access> ::= <ident> "[" <expr> "]" "[" <expr> "]" "[" <expr> "]"
```

记录访问

```
1 <record_access> ::= <ident> "." <ident>
```

if语句

```
1 <if_stmt> = "if" "(" <bool_expr> ")" "then" "{" <stmt_list> "}" "end"  
2           | "if" "(" <bool_expr> ")" "then" "{" <stmt_list> "}" "else" "{"  
           <stmt_list> "}" "end"
```

while语句

```
1 <while_stmt> = "while" "(" <bool_expr> ")" "{" <stmt_list> "}"
```

scan语句，等价于PL0中的read

```
1 <scan_stmt> = "scan" "(" <ident> {"," <ident>} ")"
```

print语句，等价于PL0中的write

```
1 <print_stmt> = "print" "(" <expr> {"," <expr>} ")"
```

其余部分

```
1 <bool_expr> = <expr> ("==" | "!=" | "<" | "<=" | ">" | ">=") <expr>  
2 <expr> = ["+" | "-"] <term> {"+" | "-"} <term>  
3 <term> = <factor> {"*" | "/" } <factor>  
4 <factor> = <ident> | <number> | "(" <expr> ")" | <array_access> |  
  <record_access> | "$" <ident>  
5 <ident> = <字母> {<字母> | <数字>}  
6 <number> = <数字> {<数字>}
```

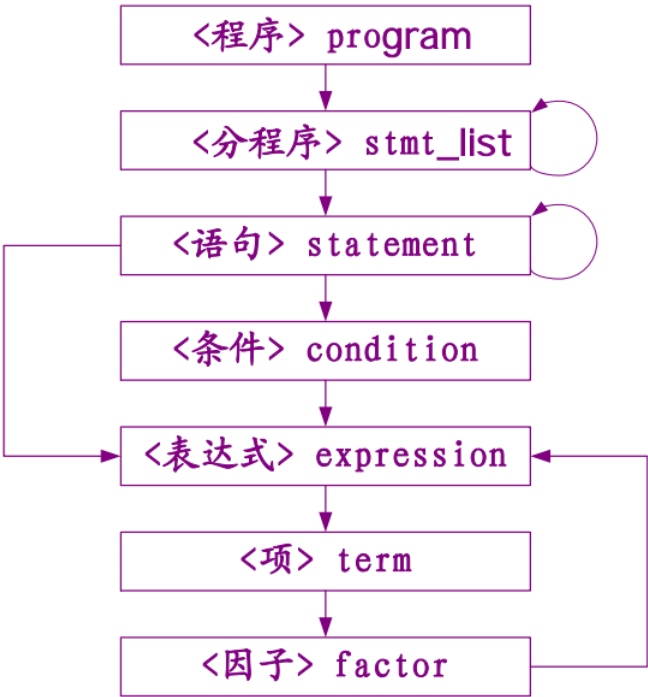
保留字

else	end
if	main
print	scan
then	while
array	record
pointer	var
end	call

终结符

+、-、*、/、(、)、[、]、,、=、,、.、;、:、#、&

◇ 主要语法单位相应子程序之间的调用关系



二、代码结构

仿照PLO编译器的写法，将L24程序翻译为类P-code代码，并在虚拟机上运行。

1.编译器的核心包括以下主要模块：

- **词法分析**：从源代码中读取字符，识别并返回单词（标识符、关键字、符号等）。
- **语法分析**：基于L24文法，解析词法分析得到的单词，构建语法树，并生成中间代码（虚拟机代码）。
- **语义分析**：检查程序的语义正确性，如类型检查、变量声明和使用的一致性。
- **中间代码生成**：根据语法树生成虚拟机代码（P-code）。
- **虚拟机**：解释和执行虚拟机代码。

2.完整的L24编译器代码包括词法分析、语法分析、语义分析和中间代码生成等模块。以下是主要函数和结构的说明：

- `main`：程序入口，处理用户输入，初始化编译器，调用编译函数，生成虚拟机代码。
- `init`：初始化保留字、符号和指令集。
- `gen`：生成目标代码。
- `getsym`：词法分析，获取一个单词符号。
- `program`：语法分析，解析程序结构。
- `statement`：解析语句列表，循环调用statement。
- `statement`：解析并生成语句的虚拟机代码。
- `expression`：解析表达式。
- `factor`：解析因子，处理变量、常量、数组、记录和指针等。
- `interpret`：虚拟机，解释执行生成的虚拟机代码。

3.完整函数声明如下：完整代码见附件

```
1 // 错误处理函数，根据错误码n输出错误信息
2 void error(int n);
3
4 // 词法分析函数，从输入源代码中获取一个符号
5 void getsym();
6
7 // 从输入中获取一个字符
8 void getch();
9
10 // 初始化编译器，设置保留字、符号和指令集等
11 void init();
12
13 // 生成一条虚拟机指令x，参数为y和z
14 void gen(enum fct x, int y, int z);
15
16 // 测试符号集合s1和s2中的符号，若当前符号不在s1中，则报错n，并在s2中查找补救
17 void test(bool* s1, bool* s2, int n);
```

```

18
19 // 判断元素e是否在集合s中
20 int inset(int e, bool* s);
21
22 // 计算s1和s2的并集, 结果存入sr, 返回集合的大小n
23 int addset(bool* sr, bool* s1, bool* s2, int n);
24
25 // 计算s1是否是s2的子集, 结果存入sr, 返回集合的大小n
26 int subset(bool* sr, bool* s1, bool* s2, int n);
27
28 // 计算s1和s2的交集, 结果存入sr, 返回集合的大小n
29 int mulset(bool* sr, bool* s1, bool* s2, int n);
30
31 // 解析程序, lev为当前层次, tx为符号表索引, fsys为后继符号集合
32 void program(int lev, int tx, bool* fsys);
33
34 // 解析语句列表, lev为当前层次, ptx为符号表索引, pdx为数据分配指针, fsys为后继符号集合
35 void stmt_list(int lev, int* ptx, int* pdx, bool* fsys);
36
37 // 解释执行虚拟机代码
38 void interpret();
39
40 // 解析因子, fsys为后继符号集合, ptx为符号表索引, lev为当前层次
41 void factor(bool* fsys, int* ptx, int lev);
42
43 // 解析项, fsys为后继符号集合, ptx为符号表索引, lev为当前层次
44 void term(bool* fsys, int* ptx, int lev);
45
46 // 解析条件表达式, fsys为后继符号集合, ptx为符号表索引, lev为当前层次
47 void condition(bool* fsys, int* ptx, int lev);
48
49 // 解析表达式, fsys为后继符号集合, ptx为符号表索引, lev为当前层次
50 void expression(bool* fsys, int* ptx, int lev);
51
52 // 解析语句, fsys为后继符号集合, ptx为符号表索引, lev为当前层次, pdx为数据分配指针
53 void statement(bool* fsys, int* ptx, int lev, int* pdx);
54
55 // 列出从cx0开始的代码
56 void listcode(int cx0);
57
58 // 列出所有生成的代码
59 void listall();
60
61 // 变量声明, ptx为符号表索引, lev为当前层次, pdx为数据分配指针
62 void vardeclaration(int* ptx, int lev, int* pdx);
63
64 // 常量声明, ptx为符号表索引, lev为当前层次, pdx为数据分配指针
65 void constdeclaration(int* ptx, int lev, int* pdx);
66
67 // 隐式常量声明, ptx为符号表索引, lev为当前层次, pdx为数据分配指针
68 void implicitconstdeclaration(int* ptx, int lev, int* pdx);
69
70 // 查找标识符idt在符号表中的位置, 返回索引tx
71 int position(char* idt, int tx);
72

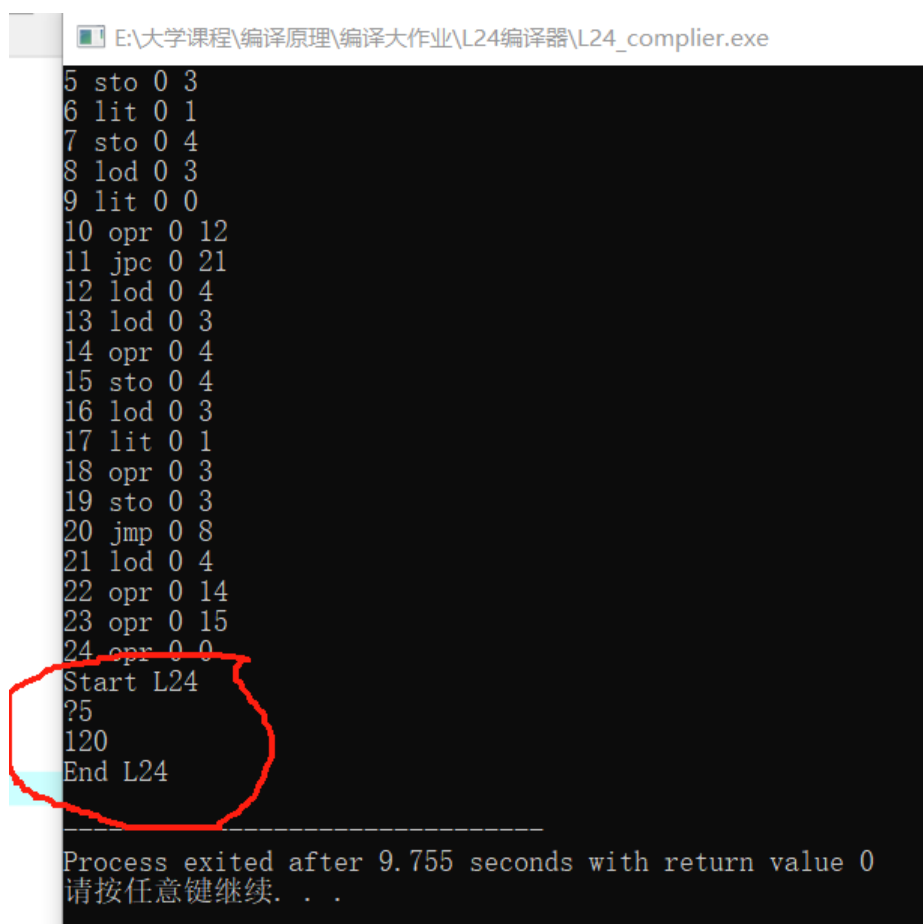
```

```
73 // 将对象k（变量、常量、数组等）加入符号表，ptx为符号表索引，lev为当前层次，pdx为数据分配
    指针
74 void enter(enum object k, int* ptx, int lev, int* pdx);
75
76 // 返回l层基地址，s为栈数组，b为当前基地址
77 int base(int l, int* s, int b);
78
79 // 数组声明，ptx为符号表索引，lev为当前层次，pdx为数据分配指针
80 void arraydeclaration(int* ptx, int lev, int* pdx);
81
82 // 记录声明，ptx为符号表索引，lev为当前层次，pdx为数据分配指针
83 void recorddeclaration(int* ptx, int lev, int* pdx);
84
85 // 指针声明，ptx为符号表索引，lev为当前层次，pdx为数据分配指针
86 void pointerdeclaration(int* ptx, int lev, int* pdx);
87
88 // 过程声明，ptx为符号表索引，lev为当前层次，pdx为数据分配指针
89 void procdeclaration(int* ptx, int lev, int* pdx);
90
```

三、测试样例

1.阶乘：输入一个整数n，返回n! 的值，如5! =120。

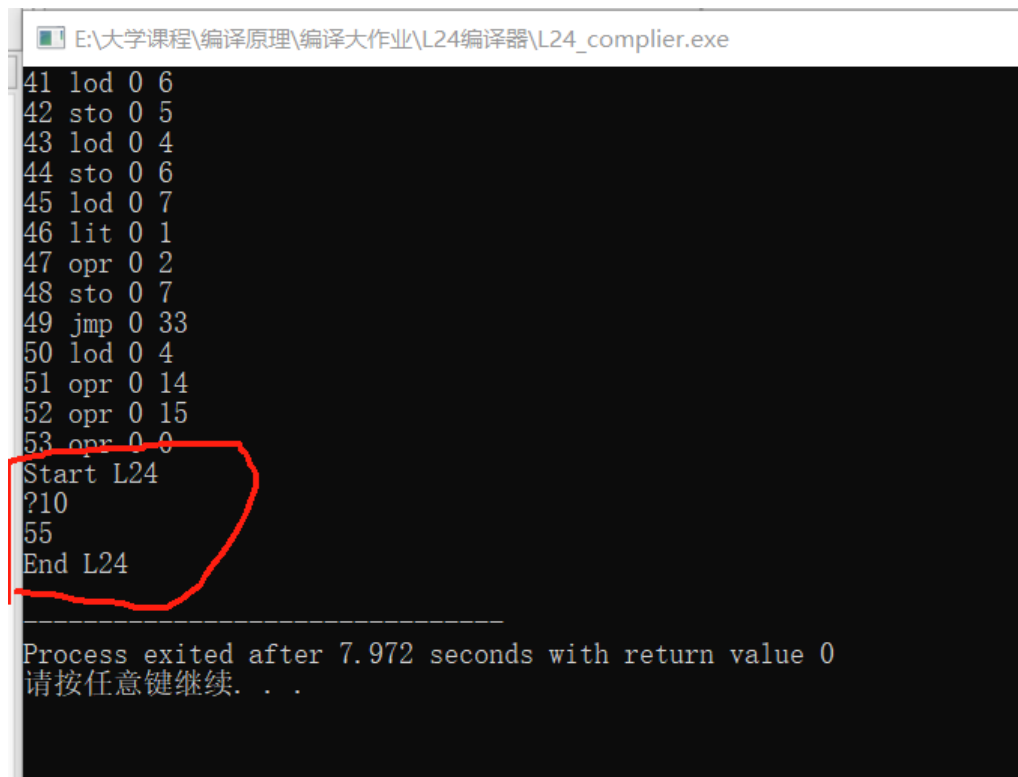
```
1  main {
2      var n,fact;
3      n=5;
4      scan(n);
5      fact = 1;
6      while (n > 0) {
7          fact = fact * n;
8          n = n - 1;
9      };
10     print(fact);
11 }
```



```
E:\大学课程\编译原理\编译大作业\L24编译器\L24_compiler.exe
5 sto 0 3
6 lit 0 1
7 sto 0 4
8 lod 0 3
9 lit 0 0
10 opr 0 12
11 jpc 0 21
12 lod 0 4
13 lod 0 3
14 opr 0 4
15 sto 0 4
16 lod 0 3
17 lit 0 1
18 opr 0 3
19 sto 0 3
20 jmp 0 8
21 lod 0 4
22 opr 0 14
23 opr 0 15
24 opr 0 0
Start L24
5
120
End L24
-----
Process exited after 9.755 seconds with return value 0
请按任意键继续. . .
```


2.斐波拉契：输入一个整数n，返回斐波拉契数列中的第n项值，如f(10)=55。

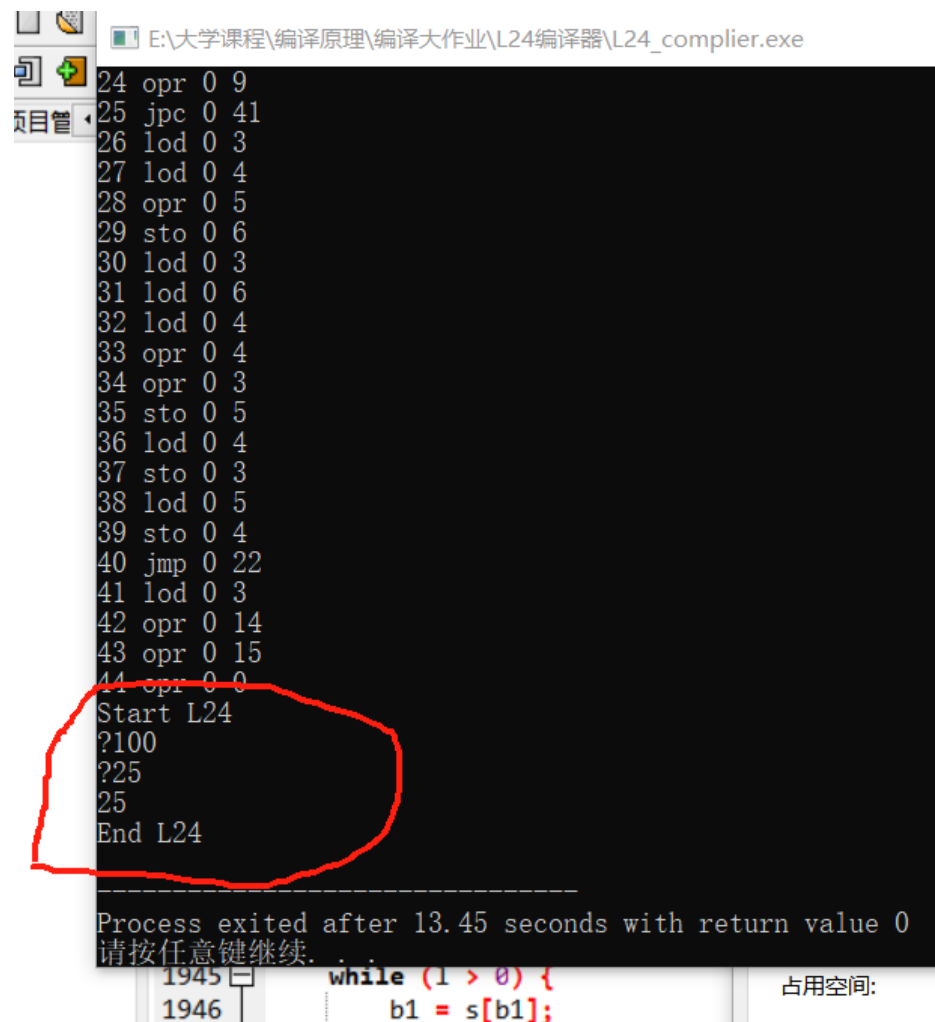
```
1  main {
2      var n,fib,a,b,i;
3      n=5;
4      fib=0;
5      scan(n);
6      if (n == 0) then {
7          fib = 0;
8      } end;
9      if (n == 1) then {
10         fib = 1;
11     } end;
12     if (n == 2) then {
13         fib = 1;
14     } else {
15         a = 1;
16         b = 1;
17         i = 3;
18         while (i <= n) {
19             fib = a + b;
20             a = b;
21             b = fib;
22             i = i + 1;
23         };
24     }end;
25     print(fib);
26 }
```



```
E:\大学课程\编译原理\编译大作业\L24编译器\L24_compiler.exe
41 lod 0 6
42 sto 0 5
43 lod 0 4
44 sto 0 6
45 lod 0 7
46 lit 0 1
47 opr 0 2
48 sto 0 7
49 jmp 0 33
50 lod 0 4
51 opr 0 14
52 opr 0 15
53 opr 0 0
Start L24
?10
55
End L24
-----
Process exited after 7.972 seconds with return value 0
请按任意键继续. . .
```

3.GCD: 输入一个整数m,n, 求他们的最大公约数。

```
1  main{
2      var m,n,r,q;
3      m=0;
4      n=0;
5      scan(m);
6      scan(n);
7
8      if (m < n) then{
9          r=m;
10         m=n;
11         n=r;
12     }end;
13     r=1;
14     while(r!=0){
15         q=m/n;
16         r=m-q*n;
17         m=n;
18         n=r;
19     };
20     print(m);
21
22 }
```



The screenshot shows a debugger window titled "E:\大学课程\编译原理\编译大作业\L24编译器\L24_compiler.exe". The assembly code is displayed in a list with addresses and instructions. A red circle highlights the section from address 1945 to 1946, which contains the instructions "Start L24", "?100", "?25", "25", and "End L24". Below the assembly code, a message box states "Process exited after 13.45 seconds with return value 0" and "请按任意键继续.". At the bottom, a code editor shows the C code snippet:

```
1945 while (1 > 0) {
1946     b1 = s[b1];
```

 and the text "占用空间:".

4.三维数组的简单使用

```
1  main {
2      array a[2][2][2];
3
4      a[1][1][1] = 1;
5      a[1][1][2] = 2;
6      a[1][2][1] = 3;
7      a[1][2][2] = 4;
8      a[2][1][1] = 5;
9      a[2][1][2] = 6;
10     a[2][2][1] = 7;
11     a[2][2][1] = 8;
12
13     print(a[1][1][1]);
14     print(a[2][2][1]);
15     print(a[2][1][1]);
16     print(a[1][1][2]+a[2][1][1]);
17 }
18
```

选择 E:\大学课程\编译原理\编译大作业\L24编译器\L24_compiler.exe

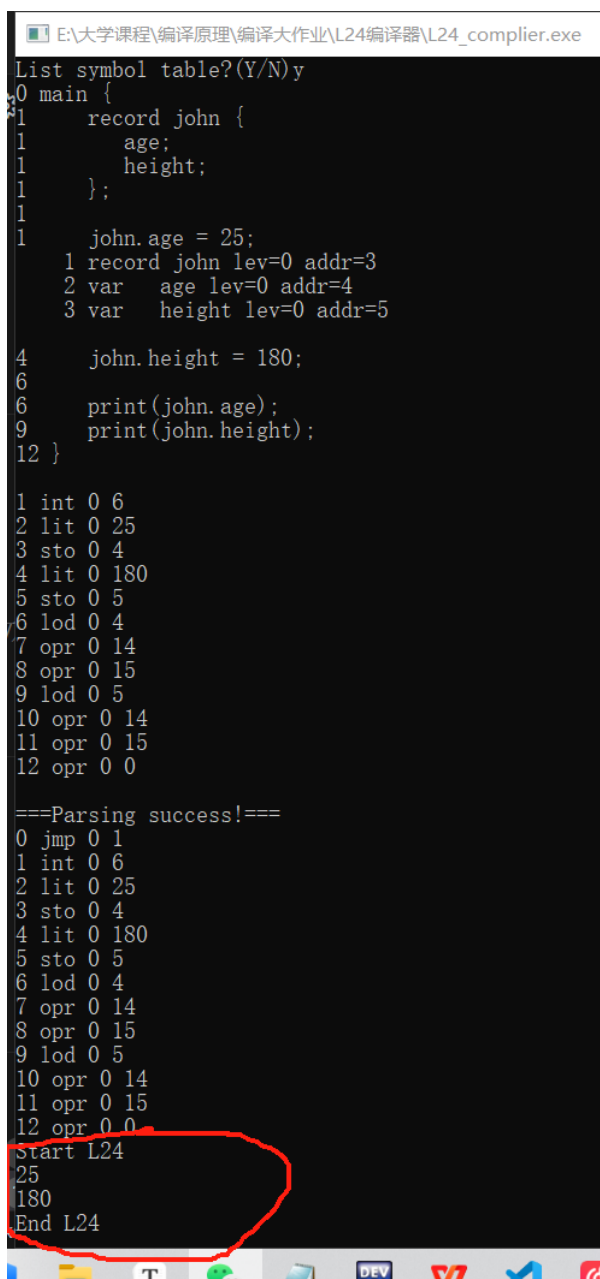
```
56 lit 0 2
57 lit 0 1
58 lit 0 1
59 opr 0 2
60 lod 0 14
61 opr 0 14
62 opr 0 15
63 lit 0 1
64 lit 0 1
65 lit 0 2
66 opr 0 2
67 lod 0 11
68 lit 0 2
69 lit 0 1
70 lit 0 1
71 opr 0 2
72 lod 0 14
73 opr 0 2
74 opr 0 14
75 opr 0 15
76 opr 0 0
Start L24
1
8
5
7
End L24
```

5.结构体的简单使用

注意我的结构体定义类似Python的类，一个结构体具有的属性直接可以赋值和使用。

因为都是整型变量，所以没有声明变量类型。

```
1  main {
2      record john {
3          age;
4          height;
5      };
6
7      john.age = 25;
8      john.height = 180;
9
10     print(john.age);
11     print(john.height);
12 }
13
```



```
E:\大学课程\编译原理\编译大作业\L24编译器\L24_compiler.exe
List symbol table?(Y/N)y
0 main {
1   record john {
1   age;
1   height;
1   };
1
1   john.age = 25;
1 record john lev=0 addr=3
2 var age lev=0 addr=4
3 var height lev=0 addr=5
4
4   john.height = 180;
6
6   print(john.age);
9   print(john.height);
12 }

1 int 0 6
2 lit 0 25
3 sto 0 4
4 lit 0 180
5 sto 0 5
6 lod 0 4
7 opr 0 14
8 opr 0 15
9 lod 0 5
10 opr 0 14
11 opr 0 15
12 opr 0 0

===Parsing success!===
0 jmp 0 1
1 int 0 6
2 lit 0 25
3 sto 0 4
4 lit 0 180
5 sto 0 5
6 lod 0 4
7 opr 0 14
8 opr 0 15
9 lod 0 5
10 opr 0 14
11 opr 0 15
12 opr 0 0
Start L24
25
180
End L24
```

四、代码细节：数组和结构体

两者处理采用类似的方法：记录数组或结构体符号名以及首地址base，用另外的变量记录offset，赋值和引用时，取地址为base+offset处的变量即可。

(1).数组

1.数组声明

```
1 //识别三维数组<ident>[number][number][number],如a[3][3][3]
2 /*注意数组声明后需要在栈中分配空间，而在program()中是根据pdx的大小来确定目标代码
   (int,0,A) 中的A大小为多少，所以有 (*pdx) += dim1 * dim2 * dim3;*/
3 void arraydeclaration(int* ptx, int lev, int* pdx) {
4     if (sym == ident) {
5         enter(array, ptx, lev, pdx);
6         getsym();
7         if (sym == lbrack) {
8             getsym();
9             int dim1 = num;
10            getsym();
11            if (sym == rbrack) {
12                getsym();
13                if (sym == lbrack) {
14                    getsym();
15                    int dim2 = num;
16                    getsym();
17                    if (sym == rbrack) {
18                        getsym();
19                        if (sym == lbrack) {
20                            getsym();
21                            int dim3 = num;
22                            getsym();
23                            if (sym == rbrack) {
24                                table[*ptx].dimensions[0] = dim1;
25                                table[*ptx].dimensions[1] = dim2;
26                                table[*ptx].dimensions[2] = dim3;
27                                (*pdx) += dim1 * dim2 * dim3;
28                                getsym();
29                            }
30                        }
31                    }
32                }
33            }
34        }
35    } else {
36        error(4); // 数组声明格式错误
37    }
38 }
```

2.数组赋值,在statement()中实现

```
1  if (table[i].kind == array) { /* 数组赋值 */
2      getsym();
3      if (sym == lbrack) {
4          getsym();
5          memcpy(nxtlev, fsys, sizeof(bool) * symnum);
6          nxtlev[rbrack] = true;
7          expression(nxtlev, ptx, lev); // 计算第一个下标
8          j=num*table[i].dimensions[1] * table[i].dimensions[2];
9          if (sym == rbrack) {
10             getsym();
11             if (sym == lbrack) {
12                 getsym();
13                 memcpy(nxtlev, fsys, sizeof(bool) * symnum);
14                 nxtlev[rbrack] = true;
15                 expression(nxtlev, ptx, lev); // 计算第二个下标
16                 j+=num*table[i].dimensions[2];
17                 if (sym == rbrack) {
18                     getsym();
19                     if (sym == lbrack) {
20                         getsym();
21                         memcpy(nxtlev, fsys, sizeof(bool) *
symnum);
22
23                         nxtlev[rbrack] = true;
24                         expression(nxtlev, ptx, lev); // 计算第三
个下标
25
26                         j+=num;
27                         if (sym == rbrack) {
28                             getsym();
29                             if (sym == becomes) {
30                                 getsym();
31                                 memcpy(nxtlev, fsys, sizeof(bool)
* symnum);
32
33                                 nxtlev[semicolon] = true;
34                                 expression(nxtlev, ptx, lev); /*
处理赋值符号右侧表达式 */
35
36                                 gen(sto, lev - table[i].level,
table[i].adr+j); /* 生成指令, 将结果存储到数组元素中 */
37                             } else {
38                                 error(13); /* 没有检测到赋值符号 */
39                             }
40                         } else {
41                             error(22); // 缺少右括号
42                         }
43                     } else {
44                         error(21); // 缺少左括号
45                     }
46                 } else {
47                     error(22); // 缺少右括号
48                 }
49             } else {
50                 error(21); // 缺少左括号
51             }
52         }
53     }
54 }
```

```

49         error(22); // 缺少右括号
50     } } }

```

3.数组引用,在factor()中实现

```

1  if (table[i].kind == array) { // 数组类型引用
2      j=1;
3      getsym();
4      if (sym == lbrack) {
5          getsym();
6          memcpy(nxtlev, fsys, sizeof(bool) * symnum);
7          nxtlev[rbrack] = true;
8          expression(nxtlev, ptx, lev); // 计算第一个下标
9          j=num*table[i].dimensions[1] *
table[i].dimensions[2];
10         if (sym == rbrack) {
11             getsym();
12             if (sym == lbrack) {
13                 getsym();
14                 memcpy(nxtlev, fsys, sizeof(bool) * symnum);
15                 nxtlev[rbrack] = true;
16                 expression(nxtlev, ptx, lev); // 计算第二个下标
17                 j+=num*table[i].dimensions[2];
18                 if (sym == rbrack) {
19                     getsym();
20                     if (sym == lbrack) {
21                         getsym();
22                         memcpy(nxtlev, fsys, sizeof(bool) *
symnum);
23
24                         nxtlev[rbrack] = true;
25                         expression(nxtlev, ptx, lev); // 计算
第三个下标
26
27                         j+=num;
28                         gen(opr, 0, 2); // 生成加法指令
29                         if (sym == rbrack) {
30                             gen(lod, lev - table[i].level,
table[i].adr+j); // 加载数组元素
31
32                             } else {
33                                 error(22); // 缺少右括号
34                             }
35                         } else {
36                             error(21); // 缺少左括号
37                         }
38                     }
39                 } else {
40                     error(22); // 缺少右括号
41                 }
42             } else {
43                 error(22); // 缺少右括号
44             }
45         }
46     }
47 }

```

(2).结构体 (记录)

1.记录声明

```
1 void recorddeclaration(int* ptx, int lev, int* pdx) {
2     int i,j;
3     if (sym == ident) {
4
5         enter(record, ptx, lev, pdx);
6         char field_name2[a1];
7         strcpy(field_name2, id); //记录名
8         i=*ptx;
9         table[(*ptx)].field_count=0;
10        strcpy(table[(*ptx)].field_name, id);
11        getsym();
12        if (sym == lbrace) {
13            getsym();
14            if (sym==ident)
15            {
16                while(sym==ident) {
17                    enter(variable, ptx, lev, pdx);
18                    j=*ptx; //enter
19                    table[i].field_count++;
20                    table[j].offset = j-i;
21                    strcpy(table[j].field_name, field_name2);
22
23                    //(*pdx)++;
24                    getsym();
25                    if (sym == semicolon) {
26                        getsym();
27                    } else {
28                        error(4);
29                    }
30                }
31
32                if (sym == rbrace) {
33                    getsym();
34                } else {
35                    error(4);
36                }
37            } else {error(4);}
38
39        } else {
40            error(4);
41        }
42    }
43    else {
44        error(4);
45    }
46 }
47 }
```


2.记录赋值, statement()中实现

```
1  if(table[i].kind==record){//记录类型, 过于繁琐.....
2      getsym();
3      if (sym == period) {
4          getsym();
5          if (sym == ident) {
6              for (j = 0; j <= table[i].field_count; j++) {
7                  if (strcmp(table[i+j].name, id) == 0) {
8                      break;
9                  }
10             }
11             if (j <= table[i].field_count) {
12                 getsym();
13                 if (sym == becomes) {
14                     getsym();
15                     memcpy(nxtlev, fsys, sizeof(bool) *
symnum);
16                     expression(nxtlev, ptx, lev); /* 处理赋值
符号右侧表达式 */
17                     gen(sto, lev - table[i].level,
table[i+j].adr); /* 生成指令, 将结果存储到记录的字段中 */
18                     } else {
19                         error(13); /* 没有检测到赋值符号 */
20                     }
21                     } else {
22                         error(27); /* 未找到记录的字段 */
23                     }
24                     } else {
25                         error(28); /* 缺少字段名 */
26                     }
27                     } else {
28                         error(29); /* 缺少点号 */
29                     }
30             }
```

3.记录引用, factor()中实现

```
1  if(table[i].kind == record){//记录
2      getsym();
3      if (sym == period) {
4          getsym();
5          if (sym == ident) {
6              for (j = 0; j <= table[i].field_count; j++) {
7                  if (strcmp(table[i+j].name, id) == 0) {
8                      break;
9                  }
10             }
11             if (j <= table[i].field_count) {
12                 // 加载记录字段的值
13                 gen(lod, lev - table[i].level,
14                    table[i+j].adr);
15                 //getsym();
16             } else {
17                 error(27); // 未找到记录的字段
18             }
19             } else {
20                 error(28); // 缺少字段名
21             }
22             } else {
23                 error(29); // 缺少点号
24             }
25         }
```

Question?

过程的参数如何定义? ? 如何传递参数? ? ? id??