

2021秋 编译原理与技术 课程实践

刘泽 PB19051176 方若兮 PB19061370 刘羲飞 PB19051052

指导教师：郑启龙 副教授

一、Yacc实现的数值比较程序

1.1 处理原文法

文法G1：

```
1 E -> E '>' E
2
3     | E '<' E
4
5     | number
```

是二义文法，消除二义性，同时为了支持浮点数和额外比较操作，增加了非终结符INUM、DNUM和OP，得G2：

```
1 E -> E OP T | T
2
3 T -> INUM | DNUM
4
5 OP -> LT | GT | EQ | NE | GE | LE
```

1.2设计思路

为非终结符E增添综合属性num_i和num_d，用于保存整数值和浮点数值；为非终结符增添综合属性type，用于确定该终结符是整数类型还是浮点类型，从而便于进行相应比较；增添综合属性value，保存每次比较后的结果。

拓展比较类型，由原来的只能进行 < 和 > 的比较，变为支持<,>,<=,>=这六种比较，为此引入了非终结符OP，并给其增加综合属性operator，并采用开关语句switch-case完成，简化代码书写。

1.3 源程序

1.3.1词法分析

```
1 //compare.l
2 %{
3 #include <stdio.h>
4 #include "y.tab.h"
5 void yyerror(char *);
6 %}
7 DIGIT [1-9]+[0-9]*|0
8 INUM  [\-]?{DIGIT}
9 DNUM  [\-]?{DIGIT}+\.[0-9]+
10 %%
11
12 {DNUM}      sscanf(yytext, "%lf", &yyval.dnum); return DNUM;
13 {INUM}      sscanf(yytext, "%d", &yyval.inum); return INUM;
14 "<"        return LT;
15 ">"        return GT;
16 "\n"       return CR;
17 "="        return EQ;
18 "!="       return NE;
19 ">="       return GE;
20 "<="       return LE;
21 [ \t]+     /*ignore whitespace*/;
22 .          return 0;
23
24 %%
```

1.3.2语法分析

```
1 //compare.y
2 %{
3 #include <stdio.h>
4 #include <string.h>
5 int yylex(void);
6 void yyerror(char *);
7 %}
8
9 %union{
10     int inum;
11     double dnum;
12     char operator;
13     struct{
14         int num_i;
```

```

15         double num_d;
16         char type;
17         int value;
18     }node;
19 }
20 %token LT GT CR EQ NE GE LE
21 %token <inum> INUM
22 %token <dnum> DNUM
23 %type <node> E
24 %type <node> T
25 %type <operator> OP
26
27 %%
28     line_list : line
29               | line_list line
30               ;
31
32     line : E CR {if($1.value) printf("True\n"); else printf("False\n"); }
33
34     E : T { $$num_i = $1.num_i;  $$num_d = $1.num_d;  $$type = $1.type;}
35       | E OP T
36       {
37         $$num_i = $3.num_i;  $$num_d = $3.num_d;  $$type = $3.type;
38         switch($2 + 256)
39         {
40             case LT:
41                 if($1.type == 'I' &&  $3.type == 'I')
42                     $$value = $1.value && ($1.num_i < $3.num_i)?1:0;
43                 else if($1.type == 'I' &&  $3.type == 'D')
44                     $$value = $1.value && ($1.num_i < $3.num_d)?1:0;
45                 else if($1.type == 'D' &&  $3.type == 'I')
46                     $$value = $1.value && ($1.num_d < $3.num_i)?1:0;
47                 else
48                     $$value = $1.value && ($1.num_d < $3.num_d)?1:0;
49                 break;
50             case GT:
51                 if($1.type == 'I' &&  $3.type == 'I')
52                     $$value = $3.value && ($1.num_i > $3.num_i)?1:0;
53                 else if($1.type == 'I' &&  $3.type == 'D')
54                     $$value = $3.value && ($1.num_i > $3.num_d)?1:0;
55                 else if($1.type == 'D' &&  $3.type == 'I')
56                     $$value = $3.value && ($1.num_d > $3.num_i)?1:0;
57                 else
58                     $$value = $3.value && ($1.num_d > $3.num_d)?1:0;
59                 break;
60             case EQ:
61                 if($1.type == 'I' &&  $3.type == 'I')
62                     $$value = $3.value && ($1.num_i == $3.num_i)?1:0;
63                 else if($1.type == 'I' &&  $3.type == 'D')
64                     $$value = $3.value && ($1.num_i == $3.num_d)?1:0;
65                 else if($1.type == 'D' &&  $3.type == 'I')
66                     $$value = $3.value && ($1.num_d == $3.num_i)?1:0;
67                 else
68                     $$value = $3.value && ($1.num_d == $3.num_d)?1:0;
69                 break;
70             case NE:
71                 if($1.type == 'I' &&  $3.type == 'I')
72                     $$value = $3.value && ($1.num_i != $3.num_i)?1:0;
73                 else if($1.type == 'I' &&  $3.type == 'D')
74                     $$value = $3.value && ($1.num_i != $3.num_d)?1:0;
75                 else if($1.type == 'D' &&  $3.type == 'I')
76                     $$value = $3.value && ($1.num_d != $3.num_i)?1:0;
77                 else
78                     $$value = $3.value && ($1.num_d != $3.num_d)?1:0;
79                 break;
80             case GE:
81                 if($1.type == 'I' &&  $3.type == 'I')
82                     $$value = $3.value && ($1.num_i >= $3.num_i)?1:0;
83                 else if($1.type == 'I' &&  $3.type == 'D')
84                     $$value = $3.value && ($1.num_i >= $3.num_d)?1:0;
85                 else if($1.type == 'D' &&  $3.type == 'I')
86                     $$value = $3.value && ($1.num_d >= $3.num_i)?1:0;
87                 else
88                     $$value = $3.value && ($1.num_d >= $3.num_d)?1:0;
89                 break;
90             case LE:
91                 if($1.type == 'I' &&  $3.type == 'I')
92                     $$value = $3.value && ($1.num_i <= $3.num_i)?1:0;
93                 else if($1.type == 'I' &&  $3.type == 'D')
94                     $$value = $3.value && ($1.num_i <= $3.num_d)?1:0;
95                 else if($1.type == 'D' &&  $3.type == 'I')
96                     $$value = $3.value && ($1.num_d <= $3.num_i)?1:0;
97                 else

```

```

98         $$$.value = $3.value && ($1.num_d <= $3.num_d)?1:0;
99         break;
100     }
101 }
102 ;
103
104 T : INUM { $$$.num_i = $1;  $$$.value = 1;  $$$.type = 'I'; }
105   | DNUM { $$$.num_d = $1;  $$$.value = 1;  $$$.type = 'D'; }
106   ;
107
108 OP : LT { $$ = LT - 256; }
109     | GT { $$ = GT - 256; }
110     | EQ { $$ = EQ - 256; }
111     | NE { $$ = NE - 256; }
112     | GE { $$ = GE - 256; }
113     | LE { $$ = LE - 256; }
114     ;
115 %%
116 void yyerror(char *str){
117     fprintf(stderr, "error:%s", str);
118 }
119
120 int yywrap(){
121     return 1;
122 }
123
124 int main()
125 {
126     yyparse();
127 }
128
129
```

1.4 运行结果

1.4.1 所给样例测试

```
1<2<3
True
1<5>3
True
7>5<3
False
```

1.4.2 额外拓展

```
2. 9>3. 2
False
-1<0
True
2>=2
True
3<=1
False
1=1. 0
True
2!=-1
True
```

二、PI/0数组扩展

2.1 数组声明的实现

实现数组声明，例如：

```

1  var v1, v2, A[2][3][2];
2  ...
3  A[1][2][1] = 1;
4  ...
5  v1 = A[1][2][1];

```

其中数组声明的相关产生式：

```
1  声明：
2
3  declaration    -> vardeclaration | constdeclaration | ...
4  vardeclaration -> 'var' variable
5  variable       -> identifier idsucc ',' variable
6  idsucc         -> epsilon | '[' number ']' idsucc
7
8  作为右值：
9
10 factor         -> identifier id_succ | ...
11 id_succ        -> epsilon | array_succ | ...
12 array_succ     -> '[' expression ']' array_succ
13
14 作为左值：
15
16 statement      -> l_value '=' expression
17 l_value        -> identifier id_succ | ...
18 id_succ        -> epsilon | array_succ | ...
19 array_succ     -> '[' expression ']' array_succ
```

数组元素的寻址原理：

设该数组声明为 $a[m-1][m-2] \dots [m-n]$ ，调用的数组元素为 $a[e-1][e-2] \dots [e-n]$ ，

那么计算地址的公式： $address = (\dots(((e_1) * m_2 + e_2) * m_3 + e_3) \dots) * m_n + e_n$.

设计思路：

1. 符号表设计

在符号表里，数组的相关信息按如下方式存储：



为此，我们设计了两个相关的数据结构dimensionHead(存储数组维数信息)和dimension(存储各维宽度)。在头文件中添加上述数据结构。

```
1  typedef struct
2  {
3      char name[MAXIDLEN + 1];
4      int kind;
5      int dimnum;
6  }dimensionHead;
7
8  typedef struct
9  {
10     char name[MAXIDLEN + 1];
11     int kind;
12     int width;
13 }dimension;
```

2. 相关指令的实现

为了便于数组实现，我们设计了四条指令LEA, LODA, STOA和OUT，并在interpret()中添加实现(头文件中也要添加定义)。

LEA：取地址指令，从符号表中读取变量地址或是数组首地址。

```
1  case LEA:
2      stack[++top] = base(stack, b, i.l) + i.a;
3      break;
```

LODA：由于数组里一个数据的地址是算出来后放到栈顶的，与以前的从符号表中获取地址不一样。因此添加了这样一个读取栈顶的地址并取值的指令。

```
1  case LODA:
2      stack[top] = stack[stack[top]];
3      break;
```

STOA：读取次栈顶中的地址并将栈顶值存入该地址。

```
1 |         case STOA:
2 |             stack[stack[top - 1]] = stack[top];
3 |             top = top - 2;
4 |             break;
```

OUT：输出数组元素。

```
1 |         case OUT:
2 |             if (i.a == 0)
3 |                 printf("%d ", stack[top--]);
4 |             else
5 |                 printf("\n");
6 |             break;
```

3. 寻址操作流程

因为涉及了新sydtype，所以在头文件中添加。

```
1 |     SYM_PRINT,
2 |     SYM_LSBRACKET,
3 |     SYM_RSBRACKET,
4 |     SYM_LBRACE,
5 |     SYM_RBRACE
```

数组元素寻址按以下步骤生成相应的指令序列：

- 第一步，取数组首地址；
- 第二步，将一个0压栈（这个是偏移地址 $offset$ ），令 $i = 1$ ；
- 第三步，计算 E_i ；
- 第四步，如果 $i = n$ ，进入第七步；否则，添加加法指令，即将 E_i 与 $offset$ 相加；
- 第五步，取 M_{i+1} ；
- 第六步，将 M_{i+1} 与 $offset$ 相乘， $i = i + 1$ ，进入第三步；
- 第七步，将首地址与 $offset$ 相加，最终在栈顶得到该数组元素的地址。

数组作为右值时，处理完以上步骤再加上LODAI以从该地址将数组变量的值取到栈顶。作为左值时，处理完等式右值后，再利用STOA将栈顶的值存入次栈顶的数组地址所指向的地址处。

4. 具体实现

添加完相应的数据结构后，下面开始具体实现：

①数组声明是和普通变量一起声明的，所以当文法分析读到var时，不管后面是什么，一起进入vardeclaration()进行分析，在函数enter()中才给予不同的分析。因此添加enter_array()函数将数组导入符号表。

```
1 | void enter_array()
2 | {
3 |     mask *mk;
4 |     dimensionHead *dhead;
5 |     dimension *dim;
6 |     array_basetx = tx;
7 |     int firstdim = 0, basetx = tx, flag = 0;
8 |
9 |     int arraysize = 1;
10 |    mk = (mask *)&table[tx];
11 |    mk->level = level;
12 |    mk->address = dx;
13 |
14 |    tx++;
15 |    strcpy(table[tx].name, id);
16 |    table[tx].kind = ID_ARRAY;
17 |    dhead = (dimensionHead *)&table[tx];
18 |    dhead->dimnum = 0;
19 |    do
20 |    {
21 |        dhead->dimnum++;
22 |        getsym();
23 |        if (sym == SYM_NUMBER)
24 |        {
25 |            if (dhead->dimnum == 1)
26 |                firstdim = 1; //第一维未缺省
27 |            tx++;
28 |            strcpy(table[tx].name, id);
29 |            table[tx].kind = ID_ARRAY;
30 |            dim = (dimension *)&table[tx];
31 |            dim->width = num;
32 |            arraysize = arraysize * num;
33 |            getsym();
34 |        }
35 |        else
36 |        {
37 |            if (dhead->dimnum != 1)
```

```

38         error(28);
39     else if (sym == SYM_RSBRACKET)
40     {
41         tx++;
42         strcpy(table[tx].name, id);
43         table[tx].kind = ID_ARRAY;
44         dim = (dimension *)&table[tx];
45         dim->width = 0;
46     }
47 }
48
49 if (sym == SYM_RSBRACKET)
50     getsym();
51 else
52     error(29);
53 } while (sym == SYM_LSBRACKET);
54
55 if (sym == SYM_EQU)
56 {
57     flag = 1;
58     initsize = 0;
59     firstdimwidth = 0;
60     for (int i = 1; i <= dhead->dimnum; i++)
61         initindex[i] = 0;
62     getsym();
63     initializer(0, basetx + 1);
64     initgen(basetx);
65     int subsize = 1;
66
67     for (int i = 3; i <= dhead->dimnum + 1; i++)
68     {
69         dim = (dimension *)&table[i + basetx];
70         subsize = subsize * dim->width;
71     }
72     dim = (dimension *)&table[basetx + 2];
73     dim->width = initsize / subsize;
74     getsym();
75 }
76
77 if (flag == 0)
78     dx = dx + arraysize;
79 }

```

②数组变量是右值时，数组分析时在获取到 ID_IDENTIFIER 之后进行的进一步分析。因此，在 factor() 该分支内的 switch 语句中添加 ID_ARRAY 选项，其内容是用一系列的指令将该数组变量的地址放到栈顶，再利用 LODA 以从该地址将数组变量的值取到栈顶。

```

1         case ID_ARRAY:
2             mk = (mask *)&table[i];
3             dimensionHead *dhead;
4             dimension *dim;
5             gen(LEA, level - mk->level, mk->address);
6             dhead = (dimensionHead *)&table[++i];
7             i++;
8             int depth = 1;
9             getsym();
10            if (sym != SYM_LSBRACKET)
11                error(30);
12            getsym();
13            set1 = createset(SYM_LSBRACKET, SYM_RSBRACKET, SYM_NULL);
14            set = uniteset(set1, fsys);
15            expression(set);
16
17            while (sym == SYM_RSBRACKET && depth < dhead->dimnum)
18            {
19                getsym();
20                if (sym != SYM_LSBRACKET)
21                    error(30);
22                getsym();
23                depth++;
24                dim = (dimension *)&table[++i];
25                gen(LIT, 0, dim->width);
26                gen(OPR, 0, OPR_MUL);
27                expression(set);
28                gen(OPR, 0, OPR_ADD);
29            }
30            gen(OPR, 0, OPR_ADD);
31            gen(LODA, 0, 0);
32            destroyset(set1);
33            destroyset(set);
34            break;

```

③数组变量是左值时，在 `statement()` 中处理。

```
1      else if (table[i].kind == ID_ARRAY)
2      {
3          mk = (mask *)&table[i];
4          dimensionHead *dhead;
5          dimension *dim;
6          gen(LEA, level - mk->level, mk->address);
7          dhead = (dimensionHead *)&table[++i];
8          i++;
9          int depth = 1;
10         getsym(); // read '['
11         if (sym != SYM_LSBACKET)
12             error(30);
13         getsym(); // read the first symbol of the expression of the first dimension
14         set1 = createset(SYM_LSBACKET, SYM_RSBRACKET, SYM_NULL);
15         set = uniteset(set1, fsys);
16         expression(set);
17
18         while (sym == SYM_RSBRACKET && depth < dhead->dimnum)
19         {
20             getsym();
21             if (sym != SYM_LSBACKET)
22                 error(30);
23             getsym();
24             depth++;
25             dim = (dimension *)&table[++i];
26             gen(LIT, 0, dim->width);
27             gen(OPR, 0, OPR_MUL);
28             expression(set);
29             gen(OPR, 0, OPR_ADD);
30         }
31         gen(OPR, 0, OPR_ADD);
32         destroyset(set1);
33         destroyset(set);
34     }
```

2.2 print()的实现

1. 需要实现的功能

实现简单的格式化输出，如：

```
1  print( b[i][j] );
2  print( i );
```

2. 思路及方法

首先给出格式化输出的产生式：

```
1  statement -> print ( L )
2
3      L -> expression , L
4
5          | expression
6
```

同时头文件中添加 `SYM_PRINT`。在词法分析器识别到 `print` 关键字时，转入 `statement()` 中分析。我们在 `statement()` 中添加相应的实现。

```
1      else if (sym == SYM_PRINT)
2      { // print number
3          getsym();
4          if (sym != SYM_LPAREN)
5              error(26);
6          else
7          {
8              getsym();
9              set = uniteset(createset(SYM_COMMA, SYM_RPAREN, SYM_NULL), fsys);
10             symset p = set;
11             expression(set);
12             gen(OUT, 0, 0);
13             while (sym == SYM_COMMA)
14             {
15                 getsym();
16                 expression(set);
17                 gen(OUT, 0, 0);
18             }
19             destroyset(set);
20             gen(OUT, 0, 1);
21             if (sym == SYM_RPAREN)
22             {
```

```

23         getsym();
24     }
25     else
26     {
27         error(22); // Missing ')'.
28     }
29 }
30 }

```

2.3 数组初始化的实现

1. 需要实现的功能：数组变量在声明时的初始化,考虑一维缺省。风格类似C语言，比如：

```

1  b[][2] = { 0,{1},{2},3,{4},5,{6},{7},{8},9, };
2  var a[][2] = {{1, 2}, {3, 4}},
3  b[3] = {1, 2, 3};

```

2. 实现思路和方法

首先给出数组初始化相关产生式。

```

1  initializer      -> assignment expression
2
3                      | '{' initializer_list '}'
4
5                      | '{' initializer_list ',' '}'
6
7  initializer_list  -> initializer
8
9                      | initializer_list ',' initializer

```

照此产生式，我们添加函数 `initializer()`。作用是处理数组初始化语句。

```

1  int initializer(int depth, int basetx)
2  {
3      dimensionHead *dhead = (dimensionHead *)&table[basetx];
4      dimension *dim;
5      if (sym == SYM_LBRACE)
6      {
7          initializer_list(depth + 1, basetx);
8          if (sym == SYM_COMMA)
9          {
10             getsym();
11             if (sym != SYM_RBRACE)
12                 error(31);
13             }
14             else if (sym != SYM_RBRACE)
15                 error(31);
16             else
17             {
18                 adjust_index(basetx);
19                 depth = caculate_depth(basetx);
20                 return depth;
21             }
22         }
23         else if (sym == SYM_NUMBER)
24         {
25             if (depth > dhead->dimnum + 1)
26                 error(35);
27             else
28                 depth = dhead->dimnum;
29
30             dim = (dimension *)&table[basetx + depth]; // the index may need to be adjusted
31             if (dim->width <= initindex[depth] && depth > 1)
32             {
33                 error(36);
34             }
35             initable[++initsize] = num;
36             initindex[depth]++;
37
38             adjust_index(basetx);
39             depth = caculate_depth(basetx);
40             return depth;
41         }
42     }

```

其中，`initializer_list()` 函数用于处理初始化数组。

```

1  int initializer_list(int depth, int basetx)
2  {
3      dimensionHead *dhead = (dimensionHead *)&table[basetx];

```



```

4     dimension *dim;
5
6     int current_depth = depth;
7     int count = 0;
8     do
9     {
10        getsym();
11        if (sym == SYM_RBACE)
12            break;
13        else if (sym == SYM_NUMBER)
14        {
15            if (current_depth == dhead->dimnum +1)
16            {
17                count++;
18            }
19            if (count > 1)
20            {
21                error(36);
22            }
23            else
24                depth = initializer(depth, basetx);
25        }
26        else if (sym == SYM_LBRACE)
27        {
28            dim = (dimension *)&table[basetx + dhead->dimnum];
29            if (initindex[dhead->dimnum] == dim->width)
30            {
31                adjust_index(basetx);
32                depth = caculate_depth(basetx);
33            }
34            depth = initializer(depth, basetx);
35        }
36        getsym();
37    } while (sym == SYM_COMMA);
38
39    enter_zero(current_depth, basetx);
40 }

```

添加 `caculate_depth()` 计算数组深度。

```

1 int caculate_depth(int basetx)
2 {
3     dimensionHead *dhead = (dimensionHead *)&table[basetx];
4     dimension *dim;
5
6     if (dhead->dimnum == 1)
7     {
8         firstdimwith = initssize;
9         return 1;
10    }
11    else
12    {
13        int i;
14        for (i = dhead->dimnum; i > 1; i--)
15        {
16            if (initindex[i] != 0)
17                return i;
18        }
19        if (i == 1)
20        {
21            firstdimwith++;
22            return 1;
23        }
24    }
25 }

```

添加 `adjust_index()`。

```

1 int adjust_index(int basetx)
2 {
3     dimensionHead *dhead = (dimensionHead *)&table[basetx];
4     dimension *dim;
5
6     int i;
7     for (i = dhead->dimnum; i > 1; i--)
8     {
9         dim = (dimension *)&table[basetx + i];
10        if (initindex[i] == dim->width)
11        {
12            initindex[i - 1]++;
13            initindex[i] = 0;
14        }
15    }
16 }

```

```
15     }
16 }
```

添加 `enter_zero()` 补零。

```
1 void enter_zero(int curdepth, int basetx)
2 {
3     dimensionHead *dhead = (dimensionHead *)&table[basetx];
4     dimension *dim;
5     if (curdepth == dhead->dimnum + 1)
6         return;
7     if (curdepth == 1 && dhead->dimnum == 1)
8         return;
9     int begin = initsize + 1;
10    int subsize = 1;
11    int i = (curdepth == 1) ? 2 : curdepth;
12    for (; i <= dhead->dimnum; i++)
13    {
14        dim = (dimension *)&table[i + basetx];
15        subsize = subsize * dim->width;
16    }
17
18    if (initsize % subsize == 0)
19        return;
20
21    int end = (initsize + subsize) / subsize * subsize;
22    for (i = begin; i <= end; i++)
23        initable[i] = 0;
24    initsize = end;
25
26    for (i = curdepth; i < dhead->dimnum; i++)
27    {
28        dim = (dimension *)&table[i + basetx];
29        initindex[i] = dim->width - 1;
30    }
31
32    dim = (dimension *)&table[dhead->dimnum + basetx];
33    initindex[dhead->dimnum] = dim->width;
34 }
```

添加 `initgen()` 生成初始化机器码。

```
1 void initgen(int basetx)
2 {
3     mask *array = (mask *)&table[basetx];
4
5     int initaddr = array->address;
6     for (int i = 1; i <= initsize; i++)
7     {
8         ini_gen(LIT, 0, i + initaddr);
9         ini_gen(LIT, 0, initable[i]);
10        ini_gen(STOA, 0, 0);
11    }
12    dx = dx + initsize;
13 }
```

2.4运行结果

2.4.1数值声明、元素打印与正常初始化

```

Please input source file name: example0.txt
0 var I,J,a[] = { 0,{1},{2},3,{4},5,{6},{7},{8},9, },
1      b[][2] = { 0,{1},{2},3,{4},5,{6},{7},{8},9, },
1      c[][2][2] = { 0,{1},{2},3,{4},5,{6},{7},{8},9, },
1      d[][2][1][1] = { 0,{1},{2},3,{4},5,{6},{7},{8},9, },
1      e[][1][3][1][3] = { 0,{1},{2},3,{4},5,{6},{7},{8},9, };
1 begin
284 I := 0;
286 while I <10 do
290 begin
290   print(a[I]);
296   I := I + 1;
300 end;
301 I := 0;
303 while I <7 do
307 begin
307   J := 0;
309 while J <2 do
313 begin
313   print(b[I][J]);
323   J := J + 1;
327 end;
328   I := I + 1;
332 end;
333 end.

Begin executing PL/0 program.
0
1
2
3
4
5
6
7
8
9
0
1
2
0
3
4
5
6
7
0
8
0
9
0
End executing PL/0 program.

```

2.4.2初始化报错

```

Please input source file name: example2.txt
0 var f[][8][5] = { {{0},{1}},{2},{3},{4},{5},{6},7,{{8}},9,},
1      g[][8][5] = { {{0},{1}},{2},{3},{4},5,{5},{6},7,{{8}},9,};
                                ^
Error 27: too many number in the array
                                ^

Please input source file name: example3.txt
0 var f[][8][5] = { {{0},{1}},{2},{3},{4}},{{5}},6,7,{{8}},9,};
                                ^
Error 28: too much '{','}' is not permitted

```

三.团队分工

刘泽：完成yacc文件的设计和代码编写，完成pl0数组声明、数组元素打印和数组初始化的主体功能，参与部分报告撰写。

方若兮：参与完成pl0数组初始化，完成答辩展示PPT。

刘羲飞：设计pl0数组的相关结构，参与完成pl0数组声明，完成实验报告撰写。

