2021秋 编译原理与技术 课程实践

刘泽 PB19051176 方若兮 PB19061370 刘義飞 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：

```
E -> E ‘>’ E

    | E '<' E

    | 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：

```
E -> E OP T | T

T -> INUM | DNUM

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

```
//compare.l
%{
#include <stdio.h>
#include "y.tab.h"
void yyerror(char *);
%}
DIGIT [1-9]+[0-9]*|0
INUM  [\-]?{DIGIT}
DNUM  [\-]?{DIGIT}+\.[0-9]+
%%

{DNUM}      sscanf(yytext, "%lf", &yylval.dnum);  return DNUM;
{INUM}      sscanf(yytext, "%d", &yylval.inum); return INUM;
"<"         return LT;
">"         return GT;
\n          return CR;
"="         return EQ;
"!="        return NE;
">="        return GE;
"<="        return LE;
[ \t]+      /*ignore whitespace*/;
.           return 0;

%%
```

1.3.2语法分析

```
//compare.y
%{
#include <stdio.h>
#include <string.h>
int yylex(void);
void yyerror(char *);
%}

%union{
       int inum;
       double dnum;
       char operator;
       struct{
                int num_i;
```

```
                double num_d;
                char type;
                int  value;
      }node;
}
%token LT GT CR EQ NE GE LE
%token <inum> INUM
%token <dnum> DNUM
%type <node> E
%type <node> T
%type <operator> OP

%%
    line_list : line
              | line_list line
              ;

    line : E CR {if($1.value) printf("True\n"); else printf("False\n"); }

    E : T { $$.num_i = $1.num_i;  $$.num_d = $1.num_d; $$.type = $1.type;}
        | E OP T
            {
                $$.num_i = $3.num_i;  $$.num_d = $3.num_d;  $$.type = $3.type;
                switch($2 + 256)
                {
                    case LT:
                        if($1.type == 'I' &&  $3.type == 'I')
                            $$.value = $1.value && ($1.num_i < $3.num_i)?1:0;
                        else if($1.type == 'I' &&  $3.type == 'D')
                            $$.value = $1.value && ($1.num_i < $3.num_d)?1:0;
                        else if($1.type == 'D' &&  $3.type == 'I')
                            $$.value = $1.value && ($1.num_d < $3.num_i)?1:0;
                        else
                            $$.value = $1.value && ($1.num_d < $3.num_d)?1:0;
                        break;
                    case GT:
                        if($1.type == 'I' &&  $3.type == 'I')
                            $$.value = $3.value && ($1.num_i > $3.num_i)?1:0;
                        else if($1.type == 'I' &&  $3.type == 'D')
                            $$.value = $3.value && ($1.num_i > $3.num_d)?1:0;
                        else if($1.type == 'D' &&  $3.type == 'I')
                            $$.value = $3.value && ($1.num_d > $3.num_i)?1:0;
                        else
                            $$.value = $3.value && ($1.num_d > $3.num_d)?1:0;
                        break;
                    case EQ:
                        if($1.type == 'I' &&  $3.type == 'I')
                            $$.value = $3.value && ($1.num_i == $3.num_i)?1:0;
                        else if($1.type == 'I' &&  $3.type == 'D')
                            $$.value = $3.value && ($1.num_i == $3.num_d)?1:0;
                        else if($1.type == 'D' &&  $3.type == 'I')
                            $$.value = $3.value && ($1.num_d == $3.num_i)?1:0;
                        else
                            $$.value = $3.value && ($1.num_d == $3.num_d)?1:0;
                        break;
                    case NE:
                        if($1.type == 'I' &&  $3.type == 'I')
                            $$.value = $3.value && ($1.num_i != $3.num_i)?1:0;
                        else if($1.type == 'I' &&  $3.type == 'D')
                            $$.value = $3.value && ($1.num_i != $3.num_d)?1:0;
                        else if($1.type == 'D' &&  $3.type == 'I')
                            $$.value = $3.value && ($1.num_d != $3.num_i)?1:0;
                        else
                            $$.value = $3.value && ($1.num_d != $3.num_d)?1:0;
                        break;
                    case GE:
                        if($1.type == 'I' &&  $3.type == 'I')
                            $$.value = $3.value && ($1.num_i >= $3.num_i)?1:0;
                        else if($1.type == 'I' &&  $3.type == 'D')
                            $$.value = $3.value && ($1.num_i >= $3.num_d)?1:0;
                        else if($1.type == 'D' &&  $3.type == 'I')
                            $$.value = $3.value && ($1.num_d >= $3.num_i)?1:0;
                        else
                            $$.value = $3.value && ($1.num_d >= $3.num_d)?1:0;
                        break;
                    case LE:
                        if($1.type == 'I' &&  $3.type == 'I')
                            $$.value = $3.value && ($1.num_i <= $3.num_i)?1:0;
                        else if($1.type == 'I' &&  $3.type == 'D')
                            $$.value = $3.value && ($1.num_i <= $3.num_d)?1:0;
                        else if($1.type == 'D' &&  $3.type == 'I')
                            $$.value = $3.value && ($1.num_d <= $3.num_i)?1:0;
                        else
```

```
                            $$.value = $3.value && ($1.num_d <= $3.num_d)?1:0;
                        break;
                }
            }
        ;

    T  : INUM { $$.num_i = $1;  $$.value = 1; $$.type = 'I'; }
       | DNUM { $$.num_d = $1;  $$.value = 1; $$.type = 'D'; }
       ;

    OP : LT {$$ = LT - 256;}
       | GT {$$ = GT - 256;}
       | EQ {$$ = EQ - 256;}
       | NE {$$ = NE - 256;}
       | GE {$$ = GE - 256;}
       | LE {$$ = LE - 256;}
       ;
%%
void    yyerror(char *str){
    fprintf(stderr, "error:%s", str);
}

int yywrap(){
    return 1;
}

int main()
{
    yyparse();
}
```

1.4 运行结果

1.4.1所给样例测试

1.4.2额外拓展

二、PI/0数组扩展

2.1 数组声明的实现

实现数组声明，例如：

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

其中数组声明的相关产生式：

```
声明：

declaration     -> vardeclaration | constdeclaration | ...
vardeclaration  -> 'var' variable
variable        -> identifier idsucc ',' variable
idsucc          -> epsilon | '[' number ']' idsucc

作为右值：

factor          -> identifier id_succ | ...
id_succ         -> epsilon | array_succ | ...
array_succ      -> '[' expression ']' array_succ

作为左值：

statement       -> l_value '=' expression
l_value         -> identifier id_succ | ...
id_succ         -> epsilon | array_succ | ...
array_succ      -> '[' expression ']' array_succ
```

数组元素的寻址原理：

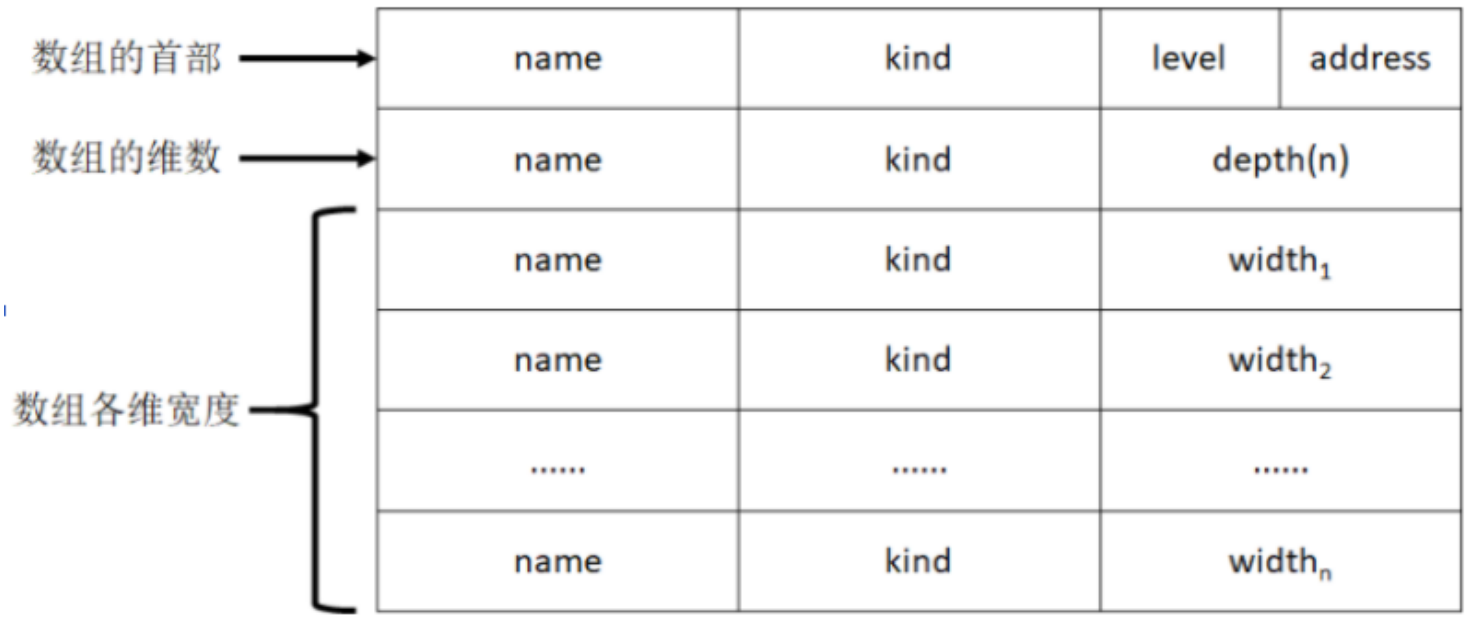
设该数组声明为 a [m - 1] [m - 2]...[m - n]，调用的数组元素为 a [e - 1] [e - 2]...[e - n]，

那么计算地址的公式：address = (...(((e $_1$) * m $_2$ + e $_2$) * m $_3$ + e $_3$)...) * 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。

```
typedef struct
{
    char name[MAXIDLEN +1];
    int kind;
    int dimnum;
}dimensionHead;

typedef struct
{
    char name[MAXIDLEN +1];
    int kind;
    int width;
}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。

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

```
        case LODA:
            stack[top] = stack[stack[top]];
            break;
```

STOA：读取次栈顶中的地址并将栈顶值存入该地址。

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

OUT：输出数组元素。

```
        case OUT:
            if (i.a == 0)
                printf("%d ", stack[top--]);
            else
                printf("\n");
            break;
```

3. 寻址操作流程

因为涉及了新symtype，所以在头文件中添加。

```
    SYM_PRINT,
    SYM_LSBRACKET,
    SYM_RSBRACKET,
    SYM_LBRACE,
    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LODA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。

```
void enter_array()
{
    mask *mk;
    dimensionHead *dhead;
    dimension *dim;
    array_basetx = tx;
    int firstdim = 0, basetx = tx, flag = 0;

    int arraysize = 1;
    mk = (mask *)&table[tx];
    mk->level = level;
    mk->address = dx;

    tx++;
    strcpy(table[tx].name, id);
    table[tx].kind = ID_ARRAY;
    dhead = (dimensionHead *)&table[tx];
    dhead->dimnum = 0;
    do
    {
        dhead->dimnum++;
        getsym();
        if (sym == SYM_NUMBER)
        {
            if (dhead->dimnum == 1)
                firstdim = 1; //第一维未缺省
            tx++;
            strcpy(table[tx].name, id);
            table[tx].kind = ID_ARRAY;
            dim = (dimension *)&table[tx];
            dim->width = num;
            arraysize = arraysize * num;
            getsym();
        }
        else
        {
            if (dhead->dimnum != 1)
```

```
                error(28);
            else if (sym == SYM_RSBRACKET)
            {
                tx++;
                strcpy(table[tx].name, id);
                table[tx].kind = ID_ARRAY;
                dim = (dimension *)&table[tx];
                dim->width = 0;
            }
        }

        if (sym == SYM_RSBRACKET)
            getsym();
        else
            error(29);
    } while (sym == SYM_LSBRACKET);

    if (sym == SYM_EQU)
    {
        flag = 1;
        initsize = 0;
        firstdimwith = 0;
        for (int i = 1; i <= dhead->dimnum; i++)
            initindex[i] = 0;
        getsym();
        initializer(0, basetx + 1);
        initgen(basetx);
        int subsize = 1;

        for (int i = 3; i <= dhead->dimnum + 1; i++)
        {
            dim = (dimension *)&table[i + basetx];
            subsize = subsize * dim->width;
        }
        dim = (dimension *)&table[basetx + 2];
        dim->width = initsize / subsize;
        getsym();
    }

    if (flag == 0)
        dx = dx + arraysize;
}
```

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。

```
                case ID_ARRAY:
                    mk = (mask *)&table[i];
                    dimensionHead *dhead;
                    dimension *dim;
                    gen(LEA, level - mk->level, mk->address);
                    dhead = (dimensionHead *)&table[++i];
                    i++;
                    int depth = 1;
                    getsym();
                    if (sym != SYM_LSBRACKET)
                        error(30);
                    getsym();
                    set1 = createset(SYM_LSBRACKET, SYM_RSBRACKET, SYM_NULL);
                    set = uniteset(set1, fsys);
                    expression(set);

                    while (sym == SYM_RSBRACKET && depth < dhead->dimnum)
                    {
                        getsym();
                        if (sym != SYM_LSBRACKET)
                            error(30);
                        getsym();
                        depth++;
                        dim = (dimension *)&table[++i];
                        gen(LIT, 0, dim->width);
                        gen(OPR, 0, OPR_MUL);
                        expression(set);
                        gen(OPR, 0, OPR_ADD);
                    }
                    gen(OPR, 0, OPR_ADD);
                    gen(LODA, 0, 0);
                    destroyset(set1);
                    destroyset(set);
                    break;
```

③数组变量是左值时，在`statement()`中处理。

```
        else if (table[i].kind == ID_ARRAY)
        {
            mk = (mask *)&table[i];
            dimensionHead *dhead;
            dimension *dim;
            gen(LEA, level - mk->level, mk->address);
            dhead = (dimensionHead *)&table[++i];
            i++;
            int depth = 1;
            getsym(); // read '['
            if (sym != SYM_LSBRACKET)
                error(30);
            getsym(); // read the first symbol of the expression of the first dimension
            set1 = createset(SYM_LSBRACKET, SYM_RSBRACKET, SYM_NULL);
            set = uniteset(set1, fsys);
            expression(set);

            while (sym == SYM_RSBRACKET && depth < dhead->dimnum)
            {
                getsym();
                if (sym != SYM_LSBRACKET)
                    error(30);
                getsym();
                depth++;
                dim = (dimension *)&table[++i];
                gen(LIT, 0, dim->width);
                gen(OPR, 0, OPR_MUL);
                expression(set);
                gen(OPR, 0, OPR_ADD);
            }
            gen(OPR, 0, OPR_ADD);
            destroyset(set1);
            destroyset(set);
         }
```

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：

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

2. 思路及方法

   首先给出格式化输出的产生式：

   ```
   statement -> print ( L )

           L -> expression , L

                | expression

   ```

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。

```
    else if (sym == SYM_PRINT)
    { // print number
        getsym();
        if (sym != SYM_LPAREN)
            error(26);
        else
        {
            getsym();
            set = uniteset(createset(SYM_COMMA, SYM_RPAREN, SYM_NULL), fsys);
            symset p = set;
            expression(set);
            gen(OUT, 0, 0);
            while (sym == SYM_COMMA)
            {
                getsym();
                expression(set);
                gen(OUT, 0, 0);
            }
            destroyset(set);
            gen(OUT, 0, 1);
            if (sym == SYM_RPAREN)
            {
```

```
                getsym();
            }
            else
            {
                error(22); // Missing ')'.
            }
        }
    }
```

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：

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

```
initializer         -> assignment expression

                     | '{' initializer_list '}'

                     | '{' initializer_list ',' '}'

initialzer_list    -> initialzer

                     | initializer_list ',' initialzer
```

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

```
int initializer(int depth, int basetx)
{
    dimensionHead *dhead = (dimensionHead *)&table[basetx];
    dimension *dim;
    if (sym == SYM_LBRACE)
    {
        initializer_list(depth + 1, basetx);
        if (sym == SYM_COMMA)
        {
            getsym();
            if (sym != SYM_RBRACE)
                error(31);
        }
        else if (sym != SYM_RBRACE)
            error(31);
        else
        {
            adjust_index(basetx);
            depth = caculate_depth(basetx);
            return depth;
        }
    }
    else if (sym == SYM_NUMBER)
    {
        if (depth > dhead->dimnum + 1)
            error(35);
        else
            depth = dhead->dimnum;

        dim = (dimension *)&table[basetx + depth]; // the index may need to be adjusted
        if (dim->width <= initindex[depth] && depth > 1)
        {
            error(36);
        }
        initable[++initsize] = num;
        initindex[depth]++;

        adjust_index(basetx);
        depth = caculate_depth(basetx);
        return depth;
    }
}
```

其中，`initializer_list()`函数用于处理初始化数组。

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

```
    dimension *dim;

    int current_depth = depth;
    int count = 0;
    do
    {
        getsym();
        if (sym == SYM_RBRACE)
            break;
        else if (sym == SYM_NUMBER)
        {
            if (current_depth == dhead->dimnum +1)
            {
                count++;
            }
            if (count > 1)
            {
                error(36);
            }
            else
                depth = initializer(depth, basetx);
        }
        else if (sym == SYM_LBRACE)
        {
            dim = (dimension *)&table[basetx + dhead->dimnum];
            if (initindex[dhead->dimnum] == dim->width)
            {
                adjust_index(basetx);
                depth = caculate_depth(basetx);
            }
            depth = initializer(depth, basetx);
        }
        getsym();
    } while (sym == SYM_COMMA);

    enter_zero(current_depth, basetx);
}
```

添加`caculate_depth()`计算数组深度。

```
int caculate_depth(int basetx)
{
    dimensionHead *dhead = (dimensionHead *)&table[basetx];
    dimension *dim;

    if (dhead->dimnum == 1)
    {
        firstdimwith = initsize;
        return 1;
    }
    else
    {
        int i;
        for (i = dhead->dimnum; i > 1; i--)
        {
            if (initindex[i] != 0)
                return i;
        }
        if (i == 1)
        {
            firstdimwith++;
            return 1;
        }
    }
}
```

添加`adjust_index()`。

```
int adjust_index(int basetx)
{
    dimensionHead *dhead = (dimensionHead *)&table[basetx];
    dimension *dim;

    int i;
    for (i = dhead->dimnum; i > 1; i--)
    {
        dim = (dimension *)&table[basetx + i];
        if (initindex[i] == dim->width)
        {
            initindex[i - 1]++;
            initindex[i] = 0;
        }
```

```
    }
}
```

添加`enter_zero()`补零。

```
void enter_zero(int curdepth, int basetx)
{
    dimensionHead *dhead = (dimensionHead *)&table[basetx];
    dimension *dim;
    if (curdepth == dhead->dimnum + 1)
        return;
    if (curdepth == 1 && dhead->dimnum == 1)
        return;
    int begin = initsize + 1;
    int subsize = 1;
    int i = (curdepth == 1) ? 2 : curdepth;
    for (; i <= dhead->dimnum; i++)
    {
        dim = (dimension *)&table[i + basetx];
        subsize = subsize * dim->width;
    }

    if (initsize % subsize == 0)
        return;

    int end = (initsize + subsize) / subsize * subsize;
    for (i = begin; i <= end; i++)
        initable[i] = 0;
    initsize = end;

    for (i = curdepth; i < dhead->dimnum; i++)
    {
        dim = (dimension *)&table[i + basetx];
        initindex[i] = dim->width - 1;
    }

    dim = (dimension *)&table[dhead->dimnum + basetx];
    initindex[dhead->dimnum] = dim->width;
}
```

添加`initgen()`生成初始化机器码。

```
void initgen(int basetx)
{
    mask *array = (mask *)&table[basetx];

    int initaddr = array->address;
    for (int i = 1; i <= initsize; i++)
    {
        ini_gen(LIT, 0, i + initaddr);
        ini_gen(LIT, 0, initable[i]);
        ini_gen(STOA, 0, 0);
    }
    dx = dx + initsize;
}
```

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。

刘義飞：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。