语法分析

周闯

320170922001

2018计算机二班

实验五

对算术表达式的递归下降分析

基本文法:

```
<Expr> → <Term> <Expr1>
<Expr1> → <AddOp> <Term> <Expr1> | empty
<Term> → <Factor> <Term1>
<Term1> → <MulOp> <Factor> <Term1> | empty
<Factor> → id |number | ( <Expr> )
<AddOp> → + | -
<MulOp> → * | /
```

给定的文法是无二义性的，且没有左递归。因此，可以对其进行直接分析；

First集&&Follow集

根据给定的文法，对每个非终结符构造First集和Follow集，结果如下所示。并且可以看出对于文法的任一非终结符，其规则右部的多个选择所推导的First集与非终结符的Follow集（当存在A→ empty）的情况下不相交。

	First	Follow
Expr	id，number，(	)，;
Expr1	+，-	)，;
Term	id，number，(	+，，)，;
Term1	* ，/	+，，)，;
Factor	id，number，(	* ，/，+，，)，;
AddOp	+ ，-	id，number，(
MulOp	* ,/	id，number，(

关键代码分析

接下来一节只展示重要代码，具体细节可以在源码中注释中查看

构建树节点

```
struct node
{
    char c;       //operate
    int pl,pr;    //child's point
    node(char tchar='0',int pleft=-1,int pright=-1)
        {
            c=tchar;pl=pleft;pr=pright;
        }
}nd[maxlen];
```

所有树节点存储再nd数组中，结构体中的c表示节点的内容；pleft和pright代表着指向的下一个方向。

递归下降方法例子分析：

以Expr1（）函数为例进行分析

```
int Expr1(int p)
{
    if(AddOp(str[f1])){     //match('+','-')
        char ch=str[f1];
        ++f1;
        int p1=Term();
        if(p1==-1)
            return p1;   //error
        int p2=++f2;
        nd[p2]=node(ch,p,p1);   //create node
        return Expr1(p2);
    }
    else if(str[f1]==')'||str[f1]==';'){
        return p;              //match Follow set
    }
    else{     //error
        return -1;
    }
}
```

因为前面已经证实first集和follow集没有交集；故通过将AddOp（+， -）进行match，消去，将指针指向下一个元素，并调用Term（）嵌套进行进一步的分析。同时通过node（）建立树中的节点。如果过程中出现错误，即可以通过返回参数进行报错。

打印语法树：

```
void Print_BFS(int p){
     const int axlen=24;
    queue<int> q0; //节点位置
    queue<int> q1;  //行数
    q0.push(p);
    q1.push(1);
    queue<int> q2;
    q2.push(int(axlen/2));
    char axes[axlen][axlen];
    for (int i=0;i<axlen;i++)
        for(int j=0;j<axlen;j++)
            axes[i][j]=' ';
    while(!q0.empty()){         //队列为空时结束遍历
        int pp=q0.front();
```

```
        int row=q1.front();
        q0.pop();q1.pop();
        int col=q2.front();
        q2.pop();
        while(axes[row][col]!=' ')
            ++col;
            axes[row][col]=nd[pp].c;
        if(nd[pp].pl!=-1){      //如果该节点不是叶子节点，则将其左右孩子添加进队列
            q0.push(nd[pp].pl);q1.push(row+1);
            q0.push(nd[pp].pr);q1.push(row+1);
            q2.push(col-2);
            q2.push(col+2);
        }
    }
    for (int i=0;i<axlen;i++)
        {
            for(int j=0;j<axlen;j++)
            cout<<axes[i][j];
            cout<<"\n";
            }
}
```

通过三个队列来帮助实现语法树的打印；q0存储节点在nd数组的位置；q1存储该节点所在的行数；q2存储该节点所在的列数。

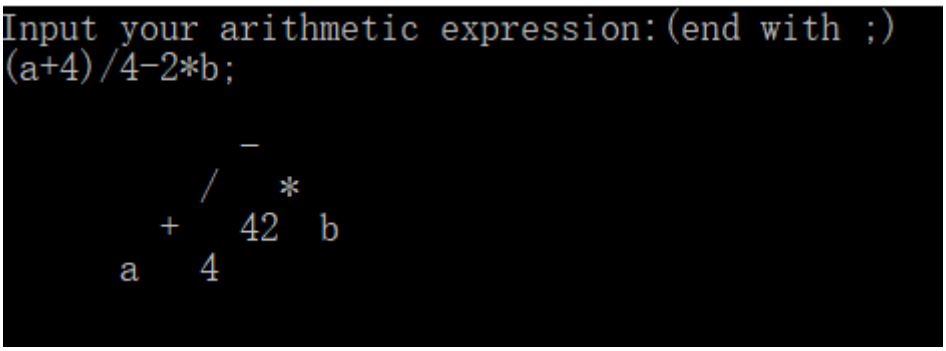
同时通过一个二维数组axes构建一个坐标系，axes数组所有默认的内容为一个空格字符；在层序遍历(广度优先遍历)语法树的同时，更新遍历节点所应该存在于axes坐标系中的值（因为题目要求所有变量以及数字都为一个字符，故可以这么进行操作）。

第一行的首个元素将被放置在坐标系第一行的中间位置，对其所有子节点，行数加一；对左节点列数减二，对右节点的列数加二；由于使用程序便利，所以同一行的节点从左至右的打印，如果某一个节点位置被前一个节点占据，则自动向右移动一格，保证内容完整的打印。

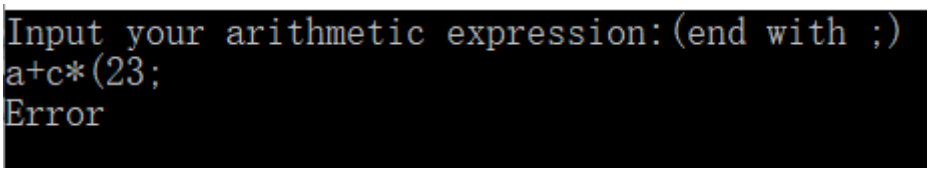
最后通过两个for循环将axes坐标系打印出来。

实验结果

输入一个正确的算术表达式，以;结尾。



错误示例：



实验六&实验七

对多条执行语句的递归下降分析

对完整程序的递归下降分析

因为实验六与实验七内容十分相似，故将其两个结合在一起进行分析。

文法分析

参考于参考资料编译原理2021.pdf中附录A 中的LittleC文法定义规则

经过分析LittleC的文法大致方向正确，但是存在一定的缺陷；但是可以将其修改后进行运用；下面列举其中的缺陷，并给出解决方法：

1.DECLS 和 STMTS 文法定义存在左递归

解决方法：将其改成：

DECLS -> DECLS1
DECLS1 -> DECL DECLS1 | empty

STMTS -> STMTS1
STMTS1 -> STMT STMTS1 | empty

2.NAMES文法定义存在左递归

解决方法：将其改成：

NAMES -> NAME NAMES1
NAMES1 -> , NAME NAMES1 | empty

3.

STMT -> if (BOOL) STMT
STMT -> if (BOOL) STMT else STMT

该文法定义存在二义性。

解决方法：

在STMT（）函数中展望一步，对其进行提前处理，以保证C语言的规则：else优先于离得最近的if进行结合。

4.因为题目只要求实现运算关系表达式，所以将BOOL与ROP等价。

5.EXPR以及TERM存在左递归

解决方法：

将其修改成实验五中的文法。

最终修改后的文法定义如下：

```
PROG→int main ( ) BLOCK
BLOCK→{ DECLS STMTS }

DECLS -> DECLS1
DECLS1 -> DECL DECLS1 | empty

DECL→TYPE NAMES ;
TYPE→int
```

```
NAMES -> NAME NAMES1
NAMES1 -> , NAME NAMES1 | empty

NAME→id

STMTS -> STMTS1
STMTS1 -> STMT STMTS1 | empty

STMT→id = EXPR ;
STMT→if ( BOOL ) STMT

STMT→if ( BOOL ) STMT else STMT
STMT→while ( BOOL ) STMT
STMT→BLOCK

STMT→ return int ;

BOOL→EXPR ROP EXPR
ROP→ > | >= | < | <= | == | !=

EXPR→TERM EXPR1

EXPR1→ADDOP TERM EXPR1| empty

TERM→FACTOR TERM1

TERM1→MULOP FACTOR TERM1 | empty

FACTOR→id | int |(EXPR)

ADDOP→+ | -

MULOP→* | /
```

First集&&Follow集

通过上面的修改之后的文法定义可以得到其的First集和Follow集：

列1	列2	列3
	First	Follow
PROG	int	{
BLOCK	{	id，if，while，return,{，}，#
DECLS	int，empty	id，if，while，return,{
DECLS1	int，empty	id，if，while，return,{
DECL	int	int，id，if，while，return,{,}
NAMES	id	;
NAMES1	,，empty	;
NAME	id	,，;
TYPE	int	id
STMTS	id，if，while，{，empty	}
STMTS1	id，if，while，{，empty	}
STMT	id，if，while，{	id，if，while，return,{，}
BOOL	id，int,(	)
ROP	>, <, ==, !=, >=, <=	id，int,(
EXPR	id，number，(	),;,>,<,==,!=,>=,<=
EXPR1	+, -	),;,>,<,==,!=,>=,<=
TERM	id, int,(	+, -,),;,>,<,==,!=,>=,<=
TERM1	*, /	+, -,),;,>,<,==,!=,>=,<=
FACTOR	id, int,(	*, /, +, -,),;,>,<,==,!=,>=,<=
ADDOP	-,+	id，int,(
MULOP	* /	id, int，(

关键代码分析

接下来一节只展示重要代码，具体细节可以在源码中注释中查看

element数组：

```
struct element
{
    int type=-1;
    string value;
}ele[100];
int e=0;//element的指针
int cnt=0;//element句法遍历
```

存储词法处理后的元素，包括其值以及类型。

```
string keyword[23]={"int","if","else","while","main","return","=","+","-","*",
//0-9
                    "/",">",">=","<","<=","==","!=","{","}",";"
 //10-19
                    ",","(",")"};
//20-22
```

type代表的类型为处在keyword数组中的下标；type=23时表示为id类型。

添加element元素并对其分类：

```
void AddEle(string c,int f)
{

    if(f==1)//字符串
    {
        int i;
        for(i=0;i<=5;i++)
        {
            if(c==keyword[i])
                {
                    ele[e].value=c;
                    ele[e].type=i;
                    e++;
                    break;
                }
        }
        if(i==6)
        {
            ele[e].value=c;
            ele[e].type=23;
            e++;
        }
    }
    else if(f==2)//数字
    {
        ele[e].value=c;
        ele[e].type=0;
        e++;
    }
    else if(f==3)//标点
    {
        int i;
        for(i=6;i<=22;i++)
        {
            if(c==keyword[i])
                {
                    ele[e].value=c;
                    ele[e].type=i;
                    e++;
                    break;
                }
        }
        if(i==23)
            error=1;
```

```
    }
    else
        error=1;
}
```

c：传入的字符串；f：传入的类型（1：关键字或者id；2：数字；标点以及符号：3）

通过该函数将词法处理的元素分类。

词法处理

```
void cifa(char* str)
{
    char str1[10];
    int i=0;
    int len=strlen(str);
   while(i<len)
   {
    if (str[i]==' '||str[i]=='\n')
        i++;
    else
    {
        if(IsChar(str[i]))
        {
            int j=0;
            str1[j]=str[i];
            j++;i++;
            while(IsId(str[i]))
            {
                str1[j]=str[i];
                j++;i++;
            }
            str1[j]='\0';
            AddEle(str1,1);

            //cout<<ele[e-1].value<<"   "<<ele[e-1].type<<"\n";
        }

        if(IsNum(str[i]))
        {
            int j=0;
            str1[j]=str[i];
            j++;i++;
            while(IsNum(str[i]))
            {
                str1[j]=str[i];
                j++;i++;
            }
            str1[j]='\0';
            AddEle(str1,2);

            //cout<<ele[e-1].value<<"   "<<ele[e-1].type<<"\n";
        }

        if(IsPun(str[i]))
        {
```

```
            int j=0;
            str1[j]=str[i];
            j++;i++;
            while((str[i-1]<='>'&&str[i-1]>='<')&&(str[i]<='>'&&str[i]>='<'))
            {
                str1[j]=str[i];
                j++;i++;
            }
            str1[j]='\0';
            AddEle(str1,3);

        }
    }
   }


}
```

通过cifa()和AddEle将词法部分功能完成。

语法处理

语法处理使用递归下降的方法，接下来将演式几个关键函数以供参考：

DECLS&DECLS1&DECL

```
void DECLS(){
    int x=ele[cnt].type;            //x记录当前单词的类别
        DECLS1();
}

void DECLS1(){
    int x=ele[cnt].type;            //x记录当前单词的类别
    if(x==0){                     //如果是 first-int
        DECL();
        if(error==1) return;       //如果出现错误，直接return
        DECLS1();
    }
    else if(x==23||x==1||x==3||x==17||x==18||x==5){ //Follow

        return;
    }
    else{      //Error
        error=1;
    }
}

void DECL(){
    int x=ele[cnt].type;            //x记录当前单词的类别

        TYPE();

        if(error==1) return;       //如果出现错误，直接return
        NAMES();

        if(error==1) return;       //如果出现错误，直接return
        if(ele[cnt].type==19){      //match ;
            cnt++;
```

```
        }
        else{                       //如果不存在';'，令error=1
            error=1;
        }


}
```

error为全局变量，初值为0；出现错误时修改为1，并直接return

其中，因为存在DESCLS1->empty，故需要考虑其follow集。

STMT

```
void STMT(){
    int x=ele[cnt].type;            //x记录当前单词的类别
    if(x==23){                      //类别(id)
        ++cnt;
        if(ele[cnt].type!=6){
            error=1;
            return;
        }
        ++cnt;                    //match('=')
        EXPR();
        if(error==1) return;       //如果出现错误，直接return
        if(ele[cnt].type==19){     //match(';')
            ++cnt;
        }
        else{
            error=1;
        }
    }
    else if(x==1){                //类别('if')
        ++cnt;
        if(ele[cnt].type!=21){     //match(
            error=1;
            return;
        }
        ++cnt;
        BOOL();
        if(error==1) return;       //如果出现错误，直接return
        if(ele[cnt].type!=22){     //match(')')
            error=1;
            return;
        }
        ++cnt;
        STMT();
        if(error==1) return;       //如果出现错误，直接return
        if(ele[cnt].type==2)      //展望一步是否为else
        {
            ++cnt;
            STMT();

        }
    }
    else if(x==3){                //类别('while')
        ++cnt;
        if(ele[cnt].type!=21){     //match('(')
```

```
            error=1;
            return;
        }
        ++cnt;
        BOOL();
        if(error==1) return;       //如果出现错误，直接return
        if(ele[cnt].type!=22){     //match(')')
            error=1;
            return;
        }
        ++cnt;
        STMT();
    }
    else if(x==17){              //类别('{')
        BLOCK();
    }
    else if(x==5)
    {

        ++cnt;
        if(ele[cnt].type!=0){     //int
            error=1;
            return;

        }
        ++cnt;
        if(ele[cnt].type!=19){     //;
            error=1;
            return;
        }
        ++cnt;
    }
    else{       //Error
        error=1;
    }
}
```

STMT文法有多个输出，所以需要考虑到不同的分支；这里可以采用提取公因式的方法。

但是值得注意到的是，在前文已经提到了：

（1）STMT -> if (BOOL) STMT
（2）STMT -> if (BOOL) STMT else STMT

该文法定义存在二义性的问题。

于是在if (BOOL) STMT后会对下一步的element进行分析，如果为else则进行（2）式的分析，否则则为（1）式；虽然文法定义较难解决二义性的问题，通过在STMT（）中展望一步更容易解决。

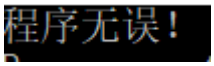
实验结果

对于输入：

```
int main ()
{
int a,b;
a=1;
```

```
b=0;
while(a<10)
{
if(a<=5)
{
a=a+2;
b=b+1;
}
else
a=a+1;

b=b+a;
}
if(b>a)
b=a;
return 0;
}
```



对于输入：

```
int main ()
{
int a,b;
a=1;
b=0;
while(a<10)
{
if(a<=5
{
a=a+2;
b=b+1;
}
else
a=a+1;

b=b+a;
}
if(b>a)
b=a;
return 0
}
```