编译原理 语法分析程序设计与实现

班级:2017211314 学号:2017213508 学生:蒋雪枫

1. **实验目标：**

编写语法分析程序，实现对算术表达式的语法分析，要求所分析的算术表达式由如下的文法产生。

**E->E+T| E-T |T**

**T->T\*F| T/F |F**

**F->id | (E) | num**

可以选择递归调用程序实现自顶向下分析、LL（1）语法分析程序、LR语法分析程序或者使用YACC自动生成语法分析程序，调用LEX自动生成的词法分析程序。

这里学生选择LL（1）展开设计：先构造为给定的文法自动构造预测分析表，然后编程实现教材的算法4.1，构造LL(1)预测分析程序，

1. **实现思路：**

（1）根据上述文法，构造消除左递归且无左公因子的文法，求解First集和Follow集

（2）构造该文法的LL(1)分析表；

（3）构造LL分析程序。

1. **具体设计：**

本次设计采用C++语言编程：

1. 从标准输入流读取一个字符串，保存需要分析的语法串
2. 用string数组来保存LL(1)分析表
3. 编程时，用M表示E’，N表示T‘，e表示ε，n表示num，打印的时候正常打印
4. 正确分析则控制程序继续推进，否则报错，提示ERROR信息
5. LL(1)预测分析程序都是按照栈顶符号X和当前串内的输入符号a来决定进行何种 过程，对于指针对对于任何（X，a），总控程序每次都执行下述三种可能的动作之一：

(1)若X = a =‘$’，则宣布分析成功，停止分析过程。

(2)若X = a!=‘$’，则把X从STACK栈顶弹出，让a指向下一个输入符号。

①如果是终结符合，则栈不加入新符号

②如果是非终结符合，则把表达式右边逆序入栈

1. 若M[A，a]中存放着“出错标志”，则调用出错诊断程序ERROR。
2. 计算可知，其first集和follow集如下：

FIRST(E) = { id, num, ( } FOLLOW (E) = { $ , )}

FIRST(E’) = { +, -,ε} FOLLOW (E’) = { $ , )}

FIRST(T) = { id, num, ( } FOLLOW (T) = { $ , + , - , )}

FIRST(T’) = { \*,/,ε( } FOLLOW (E’) = { $ , + , - , )}

FIRST(F) ={ id, num, ( } FOLLOW (F) = { $ , + , - , \* , / , ) }

1. **代码说明：**
2. 全局变量说明：

stack<char> Stack; //符号栈

char Terminals[]={'+','-','\*','/','(',')','i','n','$'}; //终结符号,i表示identifier,n表示数字num

char UnTerminals[]={'E','M','T','N','F'}; //非终结符号集合

string Map[5][9]={ //手动构造LL(1)语法预测分析表

"","","","","TM","","TM","TM","",

"+TM","-TM","","","","e","","","e",

"","","","","FN","","FN","FN","",

"e","e","\*FN","/FN","","e","","","e",

"","","","","(E)","","i","n",""

};

char input[30];// 保存输入的文法串

int len;//输入串长度

Int ip; //输入串已经翻译的指针

2.全局函数说明：

void getInput(void) //获取待分析输入表达式，并计算其长度，添加末尾$符号

bool checkID(char ch) //判断字符是否为构成id的字母

bool checkNum(char ch) //判断字符是否为构成num的数字

void outputCurSTack(void) //打印当前栈中的符号

void outputCurBuffer(int ip) //打印当前输入缓冲区中的符号串

int getTerminalSeq(char ch) //返回终结符在终结符表中的下标

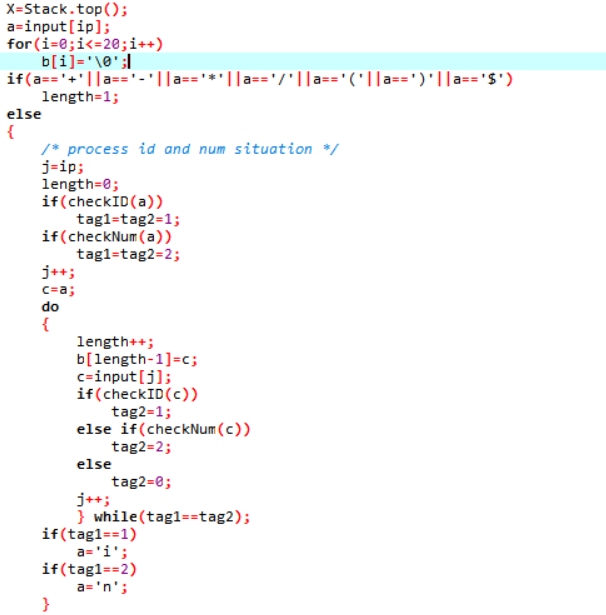
int getUnterminalSeq(char ch) //返回非终结符在非终结符表中的下标

bool checkTerminal(char ch) //判断字符是否为终结符

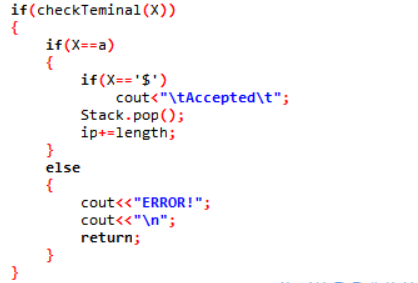
bool checkUnterminal(char ch) //判断字符是否为非终结符

void analyzer(void) //使用LL(1)语法预测分析表分析输入的表达式并输出分析结果

3.重点代码说明：



以上用于处理输入语法串的限界，比如189是整体作为一个num，abc是整体作为一个identifier。

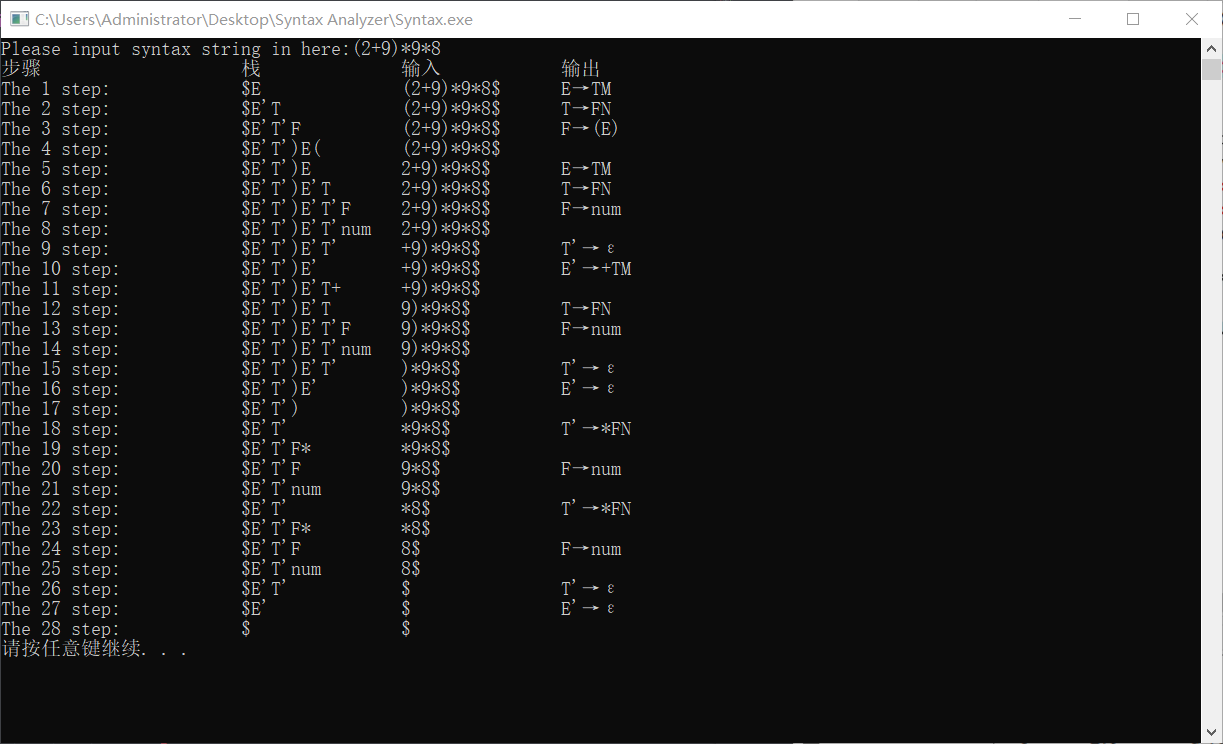


如果X栈顶是终结符，则判断是否与栈顶相等，不相等则报错。



查表逻辑

1. **运行结果说明：**



1. **代码附录：**

#include<iostream>

#include<iomanip>

#include<stack>

#include<string>

using namespace std;

stack<char> Stack;

*// 终结符号集,i表示identifier,n表示数字num*

char Terminals[]={'+','-','\*','/','(',')','i','n','$'};

char UnTerminals[]={'E','M','T','N','F'};

string Map[5][9]={*//手动构造LL(1)语法预测分析表*

     "","","","","TM","","TM","TM","",

     "+TM","-TM","","","","e","","","e",

     "","","","","FN","","FN","FN","",

     "e","e","\*FN","/FN","","e","","","e",

     "","","","","(E)","","i","n",""

     };

*// 保存输入的文法串*

char input[30];

int len;

void getInput(void)

{

    printf("Please input syntax string in here:");

    scanf("%s",input);

*// solve its len*

    for(len=0;input[len]!='\0';len++);

*// dollar符号添加到分析串最后*

    input[len]='$';

    input[len+1]='\0';

    len++;

*// 初始化符号栈*

    Stack.push('$');

    Stack.push('E');

}

bool checkID(char *ch*)

{

    if(isalpha(ch))

        return true;

    return false;

}

bool checkNum(char *ch*)

{

    if(isdigit(ch))

        return true;

    return false;

}

void outputCurStack(void)

{

    char curr[30];

    int i=0,j=0,number=Stack.size();

    cout<<"\t";

    for(i=0;i<number;i++)

    {

        curr[i]=Stack.top();

        Stack.pop();

    }

    for(i=number-1;i>=0;i--)

    {

        switch(curr[i])

        {

            case 'M':

                cout<<"E'";

                break;

            case 'N':

                cout<<"T'";

                break;

            case 'i':

                cout<<"id";

                j++;

                break;

            case 'n':

                cout<<"num";

                j+=2;

                break;

            default:

                cout<<curr[i];

                break;

        }

        Stack.push(curr[i]);

    }

    for(i=0;i<8-j-number;i++)

        cout<<" ";

}

void outputCurBuffer(int *ip*)

{

    int i,j;

    cout<<"\t";

    for(i=ip;input[i]!='$';i++)

        cout<<input[i];

    cout<<"$";

    for(j=0;j<8-i+ip;j++)

        cout<<" ";

}

int getTerminalSeq(char *ch*)*//返回终结符在终结符表中的下标*

{

    int i;

    for(i=0;i<9;i++)

       if(ch==Terminals[i])

          return i;

    return -1;

}

int getUnterminalSeq(char *ch*)*//返回非终结符在非终结符表中的下标*

{

    int i;

    for(i=0;i<5;i++)

       if(ch==UnTerminals[i])

          return i;

    return -1;

}

bool checkTeminal(char *ch*)*//判断字符是否为终结符*

{

    if(ch=='+'||ch=='-'||ch=='\*'||ch=='/'||ch=='('||ch==')'||ch=='i'||ch=='n'||ch=='$')

       return true;

    else

       return false;

}

bool checkUnterminal(char *ch*)*//判断字符是否为非终结符*

{

    if (ch=='E'||ch=='M'||ch=='T'||ch=='N'||ch=='F')

       return true;

    else

       return false;

}

void analyzer(void)

{

    int i,j,n,m;

    int step=0,ip=0,length=0;

    int tag1=0,tag2=0;

    char X,a,c,b[20],st[20];

    cout<<"步骤\t"<<"\t\t栈\t\t"<<"输入\t"<<"\t输出\t\n";

    do

    {

        step++;

        printf("The %d step:\t",step);

        X=Stack.top();

        a=input[ip];

        for(i=0;i<=20;i++)

            b[i]='\0';

        if(a=='+'||a=='-'||a=='\*'||a=='/'||a=='('||a==')'||a=='$')

            length=1;

        else

        {

*/\* process id and num situation \*/*

            j=ip;

            length=0;

            if(checkID(a))

                tag1=tag2=1;

            if(checkNum(a))

                tag1=tag2=2;

            j++;

            c=a;

            do

            {

                length++;

                b[length-1]=c;

                c=input[j];

                if(checkID(c))

                    tag2=1;

                else if(checkNum(c))

                    tag2=2;

                else

                    tag2=0;

                j++;

                } while(tag1==tag2);

            if(tag1==1)

                a='i';

            if(tag1==2)

                a='n';

            }

        outputCurStack();

        outputCurBuffer(ip);

        if(checkTeminal(X))

        {

            if(X==a)

            {

                if(X=='$')

                    cout<"\tAccepted\t";

                Stack.pop();

                ip+=length;

            }

            else

            {

                cout<<"ERROR!";

                cout<<"\n";

                return;

            }

        }

        else*//栈顶符号是非终结符*

        {

            n=getUnterminalSeq(X);

            m=getTerminalSeq(a);

            if(Map[n][m]!="")

            {

                Stack.pop();*//从栈顶弹出X*

                if(Map[n][m]!="e")*//如果产生式不是ε，将产生式逆序压入栈中*

                {

                   for(i=0;Map[n][m][i]!='\0';i++)

                      st[i]=Map[n][m][i];

                   for(i=i-1;i>=0;i--)

                      Stack.push(st[i]);

                   }

                switch(X)

                {

                   case 'M':

                      cout<<"\tE'→";

                      break;

                   case 'N':

                      cout<<"\tT'→";

                      break;

                   default:

                      cout<<"\t"<<X<<"→";

                      break;

                      }

                if(Map[n][m]=="e")

                   cout<<"ε\t";

                else if(Map[n][m]=="i")

                   cout<<"id";

                else if(Map[n][m]=="n")

                   cout<<"num";

                else

                   cout<<Map[n][m]<<"\t";

                }

            else

            {

                cout<<"\t错误!\t\n";

                return;

                }

            }

        cout<<"\n";

    } while(X!='$');

}

int main(void)

{

    getInput();

    analyzer();

    system("pause");

    return 0;

}