Project3_表达式类型的实现_实验报告

一. 实验目的

假设算术表达式 Expression 内可以含有变量(a~z)、常量(0~9)和二元运算符 (+,-,*,/,^(乘幂))。实现以下操作：

(1) ReadExpr(E)——以字符序列的形式输入语法正确的前缀表达式并构成表 达式 E；

(2) WritrExpr(E)——用带括弧的中缀表示式输出表达式 E；

(3) Assign(V, c)——实现对变量 V 的赋值(V = c)，变量的初值为 0；

(4) Value(E)——对算术表达式 E 求值；

(5) CompoundExpr(P, E1, E2)——构成一个新的复合表达式(E1)P(E2)。

(6) 增加求偏导数运算 Diff(E, V)——求表达式 E 对变量 V 的导数；

(7) 增加求偏导数运算 Diff(E, V)——求表达式 E 对变量 V 的导数；

(8) 增加常数合并操作 MergeConst(E)——合并表达式 E 中所有常数运算。例如， 对表达式 E = (2+3-a)*(b+3*4) 进行合并常数的操作后，求得 E = (5-a)*(b+12)；

二.实验环境

编程语言和开发工具

编程语言：C++

开发工具：Visual Studio Code

三.分析与设计

3.1 需求分析:将输入的前缀表达式存入二叉树中，并可以进行上述实验要求当中的一系列操作

3.2 设计思路及细节:

1. 大概：倒序遍历前缀表达式，遇到运算符则将后两位数字赋之成为其叶子结点，双重 for 循环以建成正确的二叉树，中序输出只需要注意各个符号的优先级并给出括号即可，求值与赋值仅需要将相应的子节点进行赋值，新增一个根节点便可实现复合表达式，三角函数等初等函数的实现将符号结点的左子结点放空，右子节点赋值即可，合并只需查找相应的数字结点并合并即可。

   排错：相关错误会有报错。

2. 不足：求偏导函数过于冗长繁杂。

3. 优点：大部分要求的功能以实现。

四.代码

1.结点的结构

将变量和数据以及运算符分三个值存放，方便管理

```
class Node{ //结点类
public:
    Node* l;     //左孩子
    Node* r;     //右孩子
    double value;    //结点为数值
    string op;  //结点为操作符
    string var;
    Node(){ //构造函数
        l = NULL;
        r = NULL;
        value=0;
        var=" ";
        op="#";
    }
};
```

2.输入前缀表达式的函数

```
string p,co;
Node *tree=NULL;
Node *temptree=NULL;
vector<string> pv,pvtemp;
Node *diffTree=new Node();
string px;
cout<<"#######################################################################"<<endl;
cout<<"#######                    请输入将要读取的前缀表达式                 #######"<<endl;
cout<<"#######                     每输入一个单位打一次回车                  #######"<<endl;
cout<<"#######                    最后输入一次回车来结束操作                 #######"<<endl;
cout<<"#######################################################################"<<endl;
getline(cin,p);
while(p!=""){
    pv.push_back(p);
    getline(cin,p);
}
findtype(pv,pv.size());
create(pv,tree,pv.size());
cout<<"#######                        建树成功                           #######"<<endl;
cout<<"#######                    该表达式带括号的中缀形式为             #######"<<endl;
inOrd(tree);
cout<<endl;
```

2.建树的操作

```
Node* create(vector<string> put,Node *&tree,int len){//建树函数
    Node *treenode[100];
    for(int i=0;i<len;i++){
        treenode[i] =new Node;
    }
    int *iook=new int [100];
    for(int i=0;i<len;i++){      //判断是否为操作符或者操作数
        if(type[i]==1){
            iook[i]=1;
        }else if(type[i]==2){
            iook[i]=3;
        }else{
            iook[i]=0;
        }
    }
    if(len==1) treenode[0]->value=num[0];
```

```
for(int i=len-1;i>=0;i--){
    if(iook[i]==0){ //如果为操作符，创建节点
        treenode[i]->op=ops[i];
        int nsnum=2;    //判断是否有左右结点，2为左子树赋值
        if((ops[i]=="sin")||(ops[i]=="cos")||(ops[i]=="t
            nsnum=1;
        }
        for(int j=i+1;j<len;j++){
            if(iook[j]==3){
                if(nsnum==2){
                    treenode[i]->l=treenode[j];
                    treenode[j]->var=vari[j];
                }
                if(nsnum==1){
                    treenode[i]->r=treenode[j];
                    treenode[j]->var=vari[j];
                }
                iook[j]=0;
                nsnum--;
            }
            if(iook[j]==1){
                if(nsnum==2){
                    treenode[i]->l=treenode[j];
                    treenode[j]->value=num[j];
                }
                if(nsnum==1){
                    treenode[i]->r=treenode[j];
                    treenode[j]->value=num[j];
                }
                iook[j]=0;
                nsnum--;
            }
            if(iook[j]==2){
                if(nsnum==2) treenode[i]->l=treenode[j];
                if(nsnum==1) treenode[i]->r=treenode[j];
                iook[j]=0;
                nsnum--;
            }
            if(nsnum==0) j=len;
        }
        iook[i]=2;//将操作后的结点设置为当前节点
    }
}
tree =treenode[0];
return tree;
```

3. 变量赋值函数（将节点中相应的变量值替换即可）

```
void Assign(Node *& p,string pv,double x){//赋值函数
    if(p!=NULL){
        Assign(p->l,pv,x);
        if(p->var==pv){
            p->value=x;
            isvar--;
        }
        Assign(p->r,pv,x);
    }
}
```

4.求值的函数

搭载三角函数等初等函数的运算

```
double count(Node* p){//求值
    if(p==NULL){return 0;}
    if((p->l==NULL)&&(p->r==NULL)){return p->value;}
    double temp1=count(p->l);
    double temp2=count(p->r);
    if (p->op == "+")//做相应运算
        p->value = temp1 + temp2;//加
    else if (p->op == "-")
        p->value = temp1 - temp2;//减
    else if (p->op == "*")
        p->value = temp1 * temp2;//乘
    else if (p->op == "/")
        p->value = temp1 / temp2;//除
    else if (p->op == "^")
        p->value = pow(temp1,temp2);//幂
    else if (p->op == "sin")
        p->value = sin(temp2);
    else if (p->op == "cos")
        p->value = cos(temp2);
    else if (p->op == "tan")
        p->value = tan(temp2);
    else if (p->op == "arcsin")
        p->value = asin(temp2);
else if (p->op == "arcsin")
    p->value = asin(temp2);
else if (p->op == "arccos")
    p->value = acos(temp2);
else if (p->op == "arctan")
    p->value = atan(temp2);
else if (p->op == "ln")
    p->value = log(temp2);
else if (p->op == "log")
    p->value = log10(temp2);
else cout << "错误请重新输入表达式"<<endl;
return p->value;
```

5.形成复合表达式的函数

```
Node *CompoundExpr(string link,Node *&p1,Node *&p2){//构成复合表达式
    Node *temp=new Node();
    temp->op=link;
    temp->l=p1;
    temp->r=p2;
    return temp;
}
```

6.求偏导

因能力有限，在偏导方面将所有可能情况枚举出来进行运算，导致偏导方面的代码量极其庞大，便不一一展示在报告中

```
void Diff(Node *&p, string x){//求偏导函数
    if(p!=NULL){
        if(p->op=="+"){ //加法的偏导
        if(p->op=="-"){ //减法的偏导
        if(p->op=="*"){ //乘法的偏导
        if(p->op=="/"){ //除法的偏导
        if(p->op=="^"){ //幂的偏导
        if(p->op=="sin"){//sin的偏导
        if(p->op=="cos"){//cos的偏导
        if(p->op=="tan"){//tan的偏导
        if(p->op=="arcsin"){//arcsin的偏导
        if(p->op=="arccos"){//arccos的偏导
        if(p->op=="arctan"){//arctan的偏导
    }
}
```

```
if(p->op=="+"){ //加法的偏导
    int temp=0;
    if(p->l->var==x){cout<<"1"; temp=1;}
    if((p->l->op!="#")&&IsExist(p->l,x)){Diff(p->l,x);temp=1;}
    if(p->r->var==x){if(temp==1){cout<<"+";}cout<<"1";}
    if((p->r->op!="#")&&IsExist(p->r,x)){if(temp==1){cout<<"+";}Diff(p->r,x);}
}
if(p->op=="-"){ //减法的偏导
    int temp=0;
    if(p->l->var==x){cout<<"1"; temp=1;}
    if((p->l->op!="#")&&IsExist(p->l,x)){Diff(p->l,x);temp=1;}
    if(p->r->var==x){if(temp==1){cout<<"-";}cout<<"1";}
    if((p->r->op!="#")&&IsExist(p->r,x)){if(temp==1){cout<<"-";}Diff(p->r,x);}
}
```

```
if(p->op=="*"){ //乘法的偏导
    if(p->l->var==x){
        if(p->r->var==x){//x*x
            cout<<"2*"<<x;
        }else if(p->r->op!="#"&&IsExist(p->r,x)){//x*f(x)
            inOrd(p->r);
            cout<<"+";
            if(p->r->op!="#"){
                cout<<"(";
            }
            Diff(p->r,x);
            if(p->r->op!="#"){
            cout<<")";
            }
            cout<<"*x";
        }else{//x*a
            inOrd(p->r);
        }
    }else if(p->l->op!="#"&&IsExist(p->l,x)){
        if(p->r->var==x){//f(x)*x
            inOrd(p->l);
            cout<<"+";
        if(p->l->op!="#"){
            cout<<"(";
        }
        Diff(p->l,x);
        if(p->l->op!="#"){
            cout<<")";
        }
        cout<<"*x";
        }else if(p->r->op!="#"&&IsExist(p->r,x)){//f(x)*f(x)
            cout<<"(";
            Diff(p->l,x);
            cout<<"*";
            inOrd(p->r);
            cout<<")+(";
            Diff(p->r,x);
            cout<<"*";
            inOrd(p->l);
            cout<<")";
        }else{//f(x)*a
            Diff(p->l,x);
            cout<<"*";
            if(p->r->op!="#"){
                cout<<"(";
            }
            inOrd(p->r);
            if(p->r->op!="#"){
                cout<<")";
            }
        }
```

```
if(p->l->var==x){
    if(p->r->var==x){//x/x
        cout<<0;
    }else if(p->r->op!="#"&&IsExist(p->r,x)){//x/f(x)
        cout<<"(";
        inOrd(p->r);
        cout<<"-"<<x<<"*";
        if(p->r->op!="#"){
            cout<<"(";
        }
        Diff(p->r,x);
        if(p->r->op!="#"){
            cout<<")";
        }
        cout<<")/";
        cout<<"(";
        if(p->r->op!="#"){
            cout<<"(";
        }
        inOrd(p->r);
        if(p->r->op!="#"){
            cout<<")";
        }
        cout<<"^2)";
```

```
}else if(p->r->op!="#"&&IsExist(p->r,x)){//f(x)/f(x)
    cout<<"((";
    Diff(p->l,x);
    cout<<")*";
    if(p->r->op!="#"){
        cout<<"(";
    }
    inOrd(p->r);
    if(p->r->op!="#"){
        cout<<")";
    }
    cout<<"-";
    if(p->l->op!="#"){
        cout<<"(";
    }
    inOrd(p->l);
    if(p->l->op!="#"){
        cout<<")";
    }
    cout<<"*(";
    Diff(p->r,x);
    cout<<"))/(";
    inOrd(p->r);
    cout<<"^2)";
}else{//f(x)/a
```

```
if(p->op=="^"){ //幂的偏导
    if(p->l->var==x){
        if(p->r->var==x){//x^x

        }else if(p->r->op!="#"&&IsExist(p->r,x)){//x^f(x)

        }else{//x^a
            if(p->r->op!="#"){
                cout<<"(";
            }
            inOrd(p->r);
            if(p->r->op!="#"){
                cout<<")";
            }
            cout<<"*"<<x<<"^(";
            inOrd(p->r);
            cout<<"-1)";
        }
    }else if(p->l->op!="#"&&IsExist(p->l,x)){
        if(p->r->var==x){//f(x)^x

        }else if(p->r->op!="#"&&IsExist(p->r,x)){//f(x)^f(x)

        }else{//f(x)^a
            cout<<"(";
            Diff(p->l,x);
```

```
        if(p->l->op!="#"){
            cout<<"(";
        }
        inOrd(p->l);
        if(p->l->op!="#"){
            cout<<")";
        }
        cout<<"^"<<x<<"*ln";
        if(p->l->op!="#"){
            cout<<"(";
        }
        inOrd(p->l);
        if(p->l->op!="#"){
            cout<<")";
        }
    }else if(p->r->op!="#"&&IsExist(p->r,x)){//a^f(x)
        cout<<"(";
        Diff(p->r,x);
        cout<<")*";
        if(p->l->op!="#"){
            cout<<"(";
        }
        inOrd(p->l);
        if(p->l->op!="#"){
            cout<<")";
        }
```

```
if(p->op=="sin"){//sin的偏导
    if(p->r->var==x){
        cout<<"cos";
        inOrd(p->r);
    }
    if(p->r->op!="#"&&IsExist(p->r,x)){
        cout<<"cos(";
        Diff(p->r,x);
        cout<<")";
    }
}
```

```
    if(p->op=="arcsin"){//arcsin的偏导
        if(p->r->var==x){
            cout<<"1/(1-"<<x<<")^(1/2)";
        }
        if(p->r->op!="#"&&IsExist(p->r,x)){
            cout<<"1/(1-(";
            inOrd(p->r);
            cout<<"))^(1/2)";
            cout<<"*(";
            Diff(p->r,x);
            cout<<")";
        }
    }
```

7.合并同类项

```
MergeConst(p->l);
if(p->op!="#"){
    if((p->l->value!=0)&&(p->r->value!=0)){
        if (p->op == "+")
            p->value =p->l->value+p->r->value;//加
        else if (p->op == "-")
            p->value = p->l->value-p->r->value;//减
        else if (p->op == "*")
            p->value = p->l->value*p->r->value;//乘
        else if (p->op == "/")
            p->value = p->l->value/p->r->value;//除
        else if (p->op == "^")
            p->value = pow(p->l->value,p->r->value);/
        p->op="#";
        p->l=NULL;
        p->r=NULL;
```

与求值有些许相似

五．实验结果

1.使用

开始先输入一个表达式再显示菜单

```
########################################################################################
########                    请输入将要读取的前缀表达式                          ########
########                      每输入一个单位打一次回车                          ########
########                    最后输入一次回车来结束操作                          ########
########################################################################################
-
sin
*
x
2
-
31
6

########                          建树成功                                      ########
########                    该表达式带括号的中缀形式为                          ########
(sin(x*2)-(31-6))
########################################################################################
########                  请依照菜单输入数字来确定操作：                        ########
########################################################################################
########################################################################################
####      1.Assign(V,c)             对变量 V 的赋值                                 ####
####      2.Value(E)                对表达式求值                                    ####
####      3.CompoundExpr(P,E1,E2)   构成新复合表达式(E1)P(E2)                       ####
####      4.Diff(E,V)               求表达式E对变量V的导数                          ####
####      5.MergeConst(E)           合并表达式E中所有常数运算                       ####
####      6.RereadExpr(E)           重新读取新的表达式                              ####
####      7.EXIT                    退出程序                                      ####
########################################################################################
```

2.测试案例

赋值

```
########              请输入将要赋值的变量名和对应值：                        ########
变量:
x
赋值为:
2
########                        赋值成功                                      ########
########################################################################################
####      1.Assign(V,c)             对变量 V 的赋值                                 ####
####      2.Value(E)                对表达式求值                                    ####
####      3.CompoundExpr(P,E1,E2)   构成新复合表达式(E1)P(E2)                       ####
####      4.Diff(E,V)               求表达式E对变量V的导数                          ####
####      5.MergeConst(E)           合并表达式E中所有常数运算                       ####
####      6.RereadExpr(E)           重新读取新的表达式                              ####
####      7.EXIT                    退出程序                                      ####
########################################################################################
```

求值

```
#######                    表达式的值为:                    #######
-25.7568
##########################################################################
####      1.Assign(V,c)              对变量 V 的赋值                    ####
####      2.Value(E)                 对表达式求值                       ####
####      3.CompoundExpr(P,E1,E2)    构成新复合表达式(E1)P(E2)          ####
####      4.Diff(E,V)                求表达式E对变量V的导数             ####
####      5.MergeConst(E)            合并表达式E中所有常数运算          ####
####      6.RereadExpr(E)            重新读取新的表达式                 ####
####      7.EXIT                     退出程序                         ####
##########################################################################
```

合并常数同类

```
((2+3)-a)*(b+3*4)
##########################################################################
#######                  请依照菜单输入数字来确定操作:              #######
##########################################################################
##########################################################################
####      1.Assign(V,c)              对变量 V 的赋值                    ####
####      2.Value(E)                 对表达式求值                       ####
####      3.CompoundExpr(P,E1,E2)    构成新复合表达式(E1)P(E2)          ####
####      4.Diff(E,V)                求表达式E对变量V的导数             ####
####      5.MergeConst(E)            合并表达式E中所有常数运算          ####
####      6.RereadExpr(E)            重新读取新的表达式                 ####
####      7.EXIT                     退出程序                         ####
##########################################################################
```

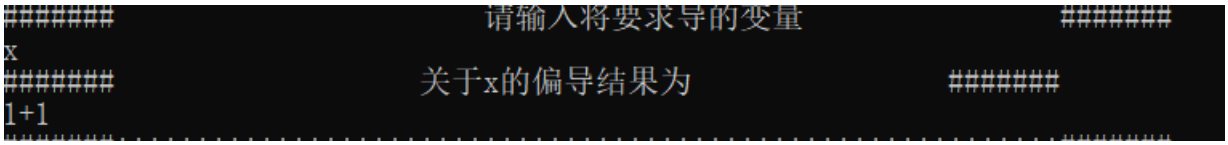
```
#######                  合并后的表达式为:                  #######
(5-a)*(b+12)
##########################################################################
####      1.Assign(V,c)              对变量 V 的赋值                    ####
####      2.Value(E)                 对表达式求值                       ####
####      3.CompoundExpr(P,E1,E2)    构成新复合表达式(E1)P(E2)          ####
####      4.Diff(E,V)                求表达式E对变量V的导数             ####
####      5.MergeConst(E)            合并表达式E中所有常数运算          ####
####      6.RereadExpr(E)            重新读取新的表达式                 ####
####      7.EXIT                     退出程序                         ####
##########################################################################
```

复合表达式

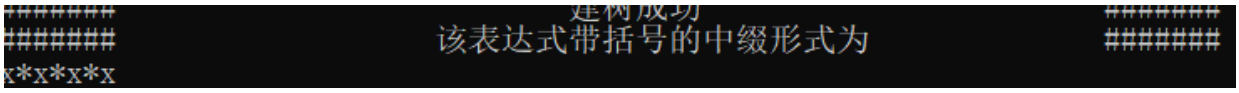
```
#######              合并后的各项操作均为合并后的表达式              #######
#######                  请输入将要合并的前缀表达式                  #######
*
-
+
2
3
a
+
b
*
3
4

#######                        建树成功                        #######
#######                  该表达式带括号的中缀形式为                #######
((2+3)-a)*(b+3*4)
#######                  请输入将要用于合并的连接符                #######
+
((5-a)*(b+12)+((2+3)-a)*(b+3*4))
```

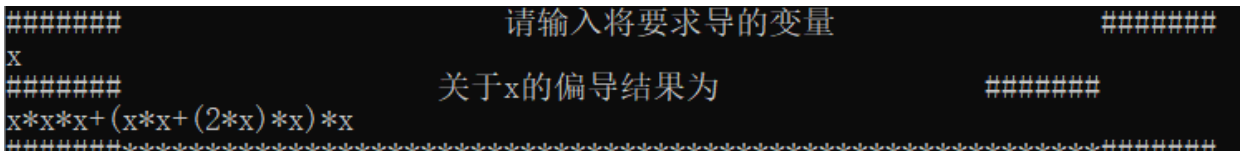
求偏导

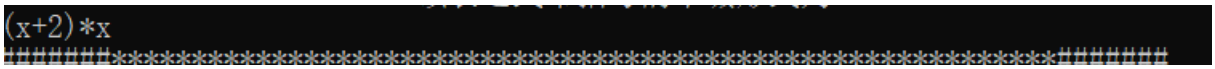
加法与减法一致

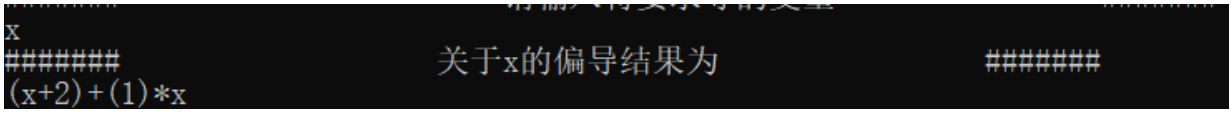


乘法







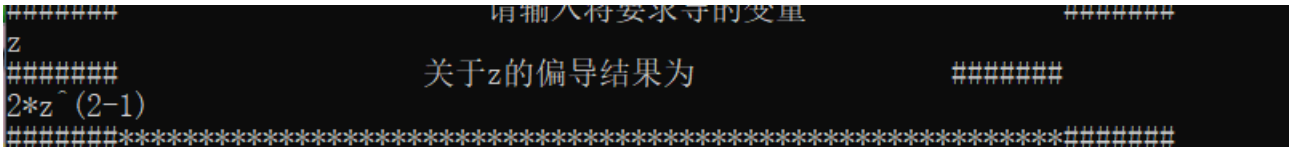


除法

1/x

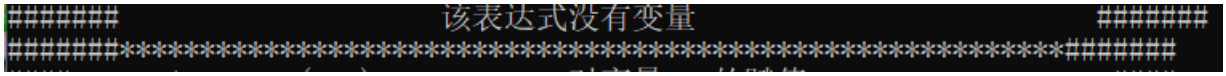
2^x

Z^2



3.排错

没有变量选择赋值



还有变量选择求值