一元稀疏多项式计算器

——实验报告

学号：PB21050988 姓名：杜朋澈

问题描述

设计一个一元稀疏多项式计算器。

基本要求

- 输入并建立多项式；
- 输出多项式（系数n为浮点序列，指数e为整数序列）；
- 多项式a和b相加，建立多项式a+b；
- 多项式a和b相减，建立多项式a-b；

拓展需求

1. 计算多项式在x处的值；
2. 求多项式a的导函数a'；
3. 多项式a和b相乘，建立乘积多项式；
4. 多项式的输出形式为类数学表达式；
5. 计算器的仿真ui界面；

*本程序包含的扩展功能为：1，2，4，5

实验设计

环境准备

- 定义一个链式存储结构，其数据域包含一个浮点型的系数域(coefficient)，一个整型的指数域(exponent)。指针域包含指向下个节点的指针。

- **多项式链式设计**：
  头节点系数域为空，指数域存放多项式项数；
  按指数非升序存放（按非降序输入）；
  每个节点数据域存放多项式各项的系数与指数；
  将0定义为空多项式，项数为1，系数与指数为0；

代码实现如下：

```
typedef struct LinkedlistNode{
    double coef;
    int expn;
    struct LinkedlistNode *next;
}LNode,*Linkedlist;
```

[模块-1]输入并建立多项式

- **函数功能设计**：接收多项式项数参数 elemnum 以及从终端录入的节点相关数据（coef，expn），将其存放在地址为head的内存空间中。

代码及必要注释如下：

```
  void CreatePolyn(Linkedlist &head,int elemnum){
    head=(Linkedlist)malloc(sizeof(LNode));
    head->expn=elemnum;//头节点指数域存放多项式项数
    head->next=NULL;
    for(int i=0;i<elemnum;i++){
        Linkedlist p=(Linkedlist)malloc(sizeof(LNode));
        cin>>p->coef;
        cin>>p->expn;
        p->next=head->next;
        head->next=p;
    }//头插法创建链表
  }
```

[模块-2]以类数学表达式模式输出多项式

- **函数功能设计**：接受多项式头节点地址，对多项式格式进行判断，以类数学表达式方式进行输出。
- **类数学表达式特征**：
  多项式首项前方无符号；
  系数为1的非0次项在输出时略去系数；
  指数为0的项在输出时仅输出系数数值；
  指数为1的项在输出时无需输出指数符号；

代码及必要注释如下：

```
void PrintPolyn(Linkedlist head){
    Linkedlist top=head;//用于存放链表中第一个系数不为0的项的地址，即多项式首项
    for(Linkedlist p=head;p->next!=NULL;p=p->next){
        if(p==head)cout<<p->expn<<"-items Polynomial: ";//指示多项式项数
        if(head->next->next==NULL&&head->next->coef==0)cout<<0;
        //若多项式为空多项式，直接输出0
        else if(p==top){
            if(p->next->coef==0)top=top->next;
            else if(p->next->coef==1&&p->next->expn==1)cout<<"x";
            else if(p->next->expn==1)cout<<p->next->coef<<"x";
            else if(p->next->expn==0)cout<<p->next->coef;
            else if(p->next->coef==1)cout<<"x^"<<p->next->expn;
            else cout<<p->next->coef<<"x^"<<p->next->expn;
        }//p为多项式首项时的输出格式判断处理
        else{
            if(p->next->coef>0){
                if(p->next->coef==1&&p->next->expn==1)cout<<"+x";
                else if(p->next->expn==1)cout<<"+"<<p->next->coef<<"x";
                else if(p->next->expn==0)cout<<"+"<<p->next->coef;
                else if(p->next->coef==1)cout<<"+x^"<<p->next->expn;
                else cout<<"+"<<p->next->coef<<"x^"<<p->next->expn;
            }//多项式中项p->next的系数为正时的输出格式判断处理
            else if(p->next->coef<0){
                if(p->next->coef==-1&&p->next->expn==1)cout<<"-x";
                else if(p->next->expn==1)cout<<p->next->coef<<"x";
                else if(p->next->expn==0)cout<<p->next->coef;
                else if(p->next->coef==-1)cout<<"-x^"<<p->next->expn;
                else cout<<p->next->coef<<"x^"<<p->next->expn;
            }//多项式中项p->next的系数为负时的输出格式判断处理
        }
    }
        cout<<endl;
}
```

*空多项式的定义由前文给出

[辅助模块-1]多项式重定义

- **函数功能设计**:对节点发生变化的多项式的各项信息进行重新定义:

代码及必要注释如下:

```
void RedefPolyn(Linkedlist &head){
    int count=0;
    for(Linkedlist p=head->next;p!=NULL;p=p->next)count++;
    head->expn=count;
    //多项式节点数量变化后，重定义头节点项数
    if(head->next==NULL){
        head->expn=1;
        head->next=(Linkedlist)malloc(sizeof(LNode));
        head->next->expn=head->next->coef=0;
        head->next->next=NULL;
    }//若链表为空表，重定义多项式为空多项式
}
```

[模块-3]多项式a b相加（a+b）

- **函数功能设计**：有序输入多项式a与多项式b的头指针，摘取b中与a指数相同的节点按多项式运算方法插入到a中合适位置，最终将b与a合并，并销毁释放相加后系数为0的节点。

代码及必要注释如下：

```
void AddPolyn(Linkedlist &La,Linkedlist &Lb){
    Linkedlist temp,temp1,temp2;//
    Linkedlist Pa,Pb;//遍历多项式a b的光标
    for(Pa=La,Pb=Lb;Pa->next!=NULL&&Pb->next!=NULL;){
        if(Pa->next->expn>Pb->next->expn)Pa=Pa->next;
        //若Pa节点指数大于Pb节点指数，保存Pa节点数据作为最终结果，Pa光标直接后移
        else if(Pa->next->expn<Pb->next->expn){
            temp=Pb->next;
            Pb->next=temp->next;
            temp->next=Pa->next;
            Pa->next=temp;
            Pa=Pa->next;
        }/*若Pa节点指数小于Pb节点指数，保存Pb节点数据作为最终结果，
        摘取Pb节点连接至Pa前，将Pb前后节点重连，等效于Pb光标后移*/
        else if(Pa->next->expn==Pb->next->expn){
            float texp=Pb->next->coef+Pa->next->coef;
        //若Pa节点指数等于Pb节点指数，暂存两节点系数之和进行判断
            if(texp==0){
                temp1=Pa->next->next;
                free(Pa->next);
                Pa->next=temp1;
                temp2=Pb->next->next;
                free(Pb->next);
                Pb->next=temp2;
            }//若系数之和为0，释放Pa与Pb的空间，将Pa，Pb的前后节点重连
            else{
                Pa->next->coef=texp;
                temp2=Pb->next->next;
                free(Pb->next);
                Pb->next=temp2;
            }//若系数和不为0，重定义Pa节点系数，释放Pb节点空间，将Pb前后节点相连
        }
    }
    if(Pa->next==NULL){
        Pa->next=Pb->next;
        free(Lb);
        Lb=NULL;
    }//若La优先遍历完成，将Lb剩余节点直接连接至La尾部，销毁释放Lb
    else{
        free(Lb);
        Lb=NULL;
    }//否则直接销毁Lb
    RedefPolyn(La);
    //[多项式重定义模块]
}
```

[辅助模块-2]多项式取负

- 为实现多项式减法而引入
- **函数功能设计**：遍历多项式，对各节点系数取相反数。

代码实现如下：

```
void NegPolyn(Linkedlist &head){
    for(Linkedlist p=head->next;p!=NULL;p=p->next)
        p->coef=-(p->coef);
}
```

[辅助模块-3]多项式销毁

- **函数功能设计**：遍历链表，释放全部节点空间，最后销毁头节点。

代码实现如下：

```
void DestroyPolyn(Linkedlist &head){
    Linkedlist temp;
    for(;head->next!=NULL;free(temp)){
        temp=head->next;
        head->next=temp->next;
    }
    free(head);
    head=NULL;
}
```

[拓展模块-1]多项式求导

- **函数功能设计**：按数学求导运算规则重定义多项式。
- **求导运算法则**：
  对于指数不为0的项，求导后系数与指数相乘，指数减一；
  对于指数为0的项（即常数项）求导后记为0；

代码及必要注释如下：

```
void DiffPolyn(Linkedlist &head){
    Linkedlist temp;
    for(Linkedlist p=head;p!=NULL&&p->next!=NULL;p=p->next){
        if(p->next->expn==0){
            temp=p->next->next;
            free(p->next);
            p->next=temp;
        }//指数为0则直接销毁，前后节点重连
        else{
            p->next->coef=p->next->coef*p->next->expn;
            p->next->expn=p->next->expn-1;
        }//否则按照求导法则重定义节点
    }
    RedefPolyn(head);
    //[多项式重定义模块]
}
```

[拓展模块-2]多项式求值

- **函数功能设计**：接收一个x值，将x按数学运算规则代入多项式中进行运算，返回一个浮点数。

代码及必要注释如下：

```
int EvalPolyn(Linkedlist head,double x){
    double sum=0;
    for(Linkedlist p=head->next;p!=NULL;p=p->next){
        sum=sum+p->coef*pow(x,p->expn);
    }//遍历多项式，按系数*x^指数的运算规则获取每项值并求和
    return sum;
}
```

UI设计

- **操作界面设计**：根据输入的操作数（OPTR）执行相应操作。

```
void UIPrinter(){
    cout<<"##Sparse polynomial calculator##"<<endl<<endl;
    cout<<"enter an OPTR to select an operation: "<<endl;
    cout<<"1.create a polynomial."<<endl;
    cout<<"2.print your polynomial."<<endl;
    cout<<"3.add two polynomials(LA + LB)."<<endl;
    cout<<"4.subtract two polynomials(LA - LB/LB - LA)."<<endl;
    cout<<"5.differentiate a polynomial."<<endl;
    cout<<"6.evaluate the polynomial at x."<<endl;
    cout<<"7.reset a polynomial."<<endl;
    cout<<"8.terminate the process."<<endl;
    cout<<"(//warning:to make it easier for continuous operations,op3 and op4 will reset LA to t
}
```

- **UI包装设计**：根据OPTR的值（1-8）设置有八个不同分支，每个分支执行必要的指示与核心操作。本程序可同时存储两个多项式LA，LB，并对它们执行多种操作。
- **主函数与UI对接**：UI函数的返回值为真时循环执行UI函数，UI函数的返回值为假时终结程序。

主函数代码如下：

```
int main(){
    int interface;
    UIPrinter();
    do{
        interface=UI();
    }
    while(interface==1);
    return 0;
}
```

UI函数代码及必要注释如下：

```
//设置全局变量LA，LB存储多项式地址
Linkedlist LA=NULL,LB=NULL;
//设置全局变量，elemnumA，elemnumB存储从终端录入得多项式项数值
int elemnumA,elemnumB;
```

```
int UI(){
    int OPTR;
    cin>>OPTR;
    switch(OPTR){
        case 1:{
            if(LA==NULL){
                cout<<"initialize the polynomial A:"<<endl;
                cout<<"enter the itemnum:"<<endl;
                cin>>elemnumA;
                cout<<"enter the coefficients and the exponents:"<<endl;
                CreatePolyn(LA,elemnumA);
                if(LA!=NULL)cout<<"succeeded."<<endl;
                else cout<<"error."<<endl;
            }
            else if(LA!=NULL&&LB==NULL){
                cout<<"initialize the polynomial B:"<<endl;
                cout<<"enter the itemnum:"<<endl;
                cin>>elemnumB;
                cout<<"enter the coefficients and the exponents:"<<endl;
                CreatePolyn(LB,elemnumB);
                if(LB!=NULL)cout<<"succeeded."<<endl;
                else cout<<"error."<<endl;
            }
            else cout<<"you already have two polynomials."<<endl;
            return 1;
        }//优先初始化多项式LA，若多项式LA已初始化，则初始化多项式LB
        case 2:{
            cout<<"choose the polynomial you want to check:"<<endl;
            cout<<"1.--LA"<<endl<<"2.--LB"<<endl;
            int optr;
            cin>>optr;
            switch(optr){
                case 1:{
                    if(LA!=NULL)PrintPolyn(LA);
                    else cout<<"you do not have LA"<<endl;
                    break;
                }
                case 2:{
                    if(LB!=NULL)PrintPolyn(LB);
                    else cout<<"you do not have LB"<<endl;
                    break;
                }
                default:cout<<"error"<<endl;
            }
            return 1;
        }//嵌套分支：内部操作数决定输出目标（对已被定义的LA或LB执行输出）
        case 3:{
            if(LA==NULL||LB==NULL)cout<<"you do not have enough polynomials for the operation."<
            else{
                AddPolyn(LA,LB);
```

```
        cout<<"the result is: "<<endl;
        PrintPolyn(LA);
    }
    return 1;
}//若LA，LB均已被定义则执行LA+LB操作并输出LA
case 4:{
    if(LA==NULL||LB==NULL)cout<<"you do not have enough polynomials for the operation."<
    else{
        cout<<"choose a way to operate:"<<endl;
        cout<<"1.LA-LB"<<endl<<"2.LB-LA"<<endl;
        int optr;
        cin>>optr;
        switch(optr){
        case 1:{
            NegPolyn(LB);
            AddPolyn(LA,LB);
            cout<<"the result is: "<<endl;
            PrintPolyn(LA);
            break;
        }
        case 2:{
            NegPolyn(LA);
            AddPolyn(LA,LB);
            cout<<"the result is: "<<endl;
            PrintPolyn(LA);
            break;
        }
        default:cout<<"error"<<endl;
        }
    }
    return 1;
}//若LA，LB均已被定义则执行嵌套分支：内部操作数决定运算方式（LA+negLB或LB+negLA）最终输出LA
case 5:{
    cout<<"choose the polynomial you want to differentiate:"<<endl;
    cout<<"1.--LA"<<endl<<"2.--LB"<<endl;
    int optr;
    cin>>optr;
    switch(optr){
        case 1:{
            if(LA!=NULL){
                DiffPolyn(LA);
                cout<<"succeeded."<<endl;
            }
            else cout<<"you do not have LA"<<endl;
            break;
        }
        case 2:{
            if(LB!=NULL){
                DiffPolyn(LB);
                cout<<"succeeded."<<endl;
            }
```

```
            else cout<<"you do not have LB"<<endl;
            break;
        }
        default:cout<<"error"<<endl;
    }
    return 1;
}//嵌套分支：内部操作数决定操作目标（对已被定义的LA或LB执行求导运算）
case 6:{
   cout<<"choose the polynomial you want to evaluate:"<<endl;
    cout<<"1.--LA"<<endl<<"2.--LB"<<endl;
    int optr;
    cin>>optr;
    switch(optr){
        case 1:{
            if(LA!=NULL){
                double x,eval;
                cout<<"set a value for x:"<<endl;
                cin>>x;
                eval=EvalPolyn(LA,x);
                cout<<"the value here is: "<<eval<<endl;
            }
            else cout<<"you do not have LA"<<endl;
            break;
        }
        case 2:{
            if(LB!=NULL){
                double x,eval;
                cout<<"set a value for x:"<<endl;
                cin>>x;
                eval=EvalPolyn(LB,x);
                cout<<"the value here is: "<<eval<<endl;
            }
            else cout<<"you do not have LB"<<endl;
            break;
        }
        default:cout<<"error"<<endl;
    }
    return 1;
}//嵌套分支：内部操作数决定操作目标（输入一个x值，对已被定义的LA或LB在x处求值）
case 7:{
    cout<<"choose the polynomial you want to reset:"<<endl;
    cout<<"1.--LA"<<endl<<"2.--LB"<<endl;
    int optr;
    cin>>optr;
    switch(optr){
        case 1:{
            if(LA!=NULL){
                DestroyPolyn(LA);
                cout<<"reset the polynomial A:"<<endl;
                cout<<"enter the itemnum:"<<endl;
                cin>>elemnumA;
```

```
                    cout<<"enter the coefficients and the exponents:"<<endl;
                    CreatePolyn(LA,elemnumA);
                    if(LA!=NULL)cout<<"succeeded."<<endl;
                    else cout<<"error."<<endl;
                }
                else cout<<"you haven't initialized LA"<<endl;
                break;
            }
            case 2:{
                if(LB!=NULL){
                    DestroyPolyn(LB);
                    cout<<"reset the polynomial B:"<<endl;
                    cout<<"enter the itemnum:"<<endl;
                    cin>>elemnumB;
                    cout<<"enter the coefficients and the exponents:"<<endl;
                    CreatePolyn(LB,elemnumB);
                    if(LA!=NULL)cout<<"succeeded."<<endl;
                    else cout<<"error."<<endl;
                }
                else cout<<"you haven't initialized LB"<<endl;
                break;
            }
            default:cout<<"error"<<endl;
        }
        return 1;
    }//拓展：重置多项式（仅在多项式已被初始化的情况下对多项式进行重定义）
    case 8:return 0;//结束进程
    default: {
        cout<<"error"<<endl;
        return 1;
    }
  }
}
```