RajvaibhavRahane

17u283 223045

SE-C Comp,Viit,Pune

***Title- Extra Assignment-Polynomial Parser***

***Performs actions such as create,sort,add(merge),multiply,evaluate polynomial***

***CODE:***

#include<bits/stdc++.h>

#include<string>

//#include<string.h>

using namespace std;

/\*

\*@RajvaibhavRahane

\*/

//-------------------------------------------------------------------

class Term{ //class Term

int degree;

float coefficient;

public:

Term(){ //default constructor

degree=coefficient=0;

}

Term(intdegree,float coefficient){ //constructor

this->degree=degree;

this->coefficient=coefficient;

}

Term addTerms(Term term1);

Term multiplyTerms(Term secondTerm);

//friend ostream& operator<<(ostream&,Term&);

intgetDegree(){return degree;}

float getCoefficient(){return coefficient;}

float evaluateTerm(float);

};

float Term::evaluateTerm(float value){

float answer=(float)pow(value,degree);

answer\*=coefficient;

return answer;

}

Term Term::addTerms(Term secondTerm){ //add 2 Terms

if(secondTerm.getDegree()!=this->getDegree()){

cout<<"cannot add terms\n";

return Term();

}

else{

return Term(secondTerm.getDegree(),secondTerm.getCoefficient()+this->getCoefficient());

}

}

Term Term::multiplyTerms(Term secondTerm){ //multiply 2 Terms

return Term(this->getDegree()+secondTerm.getDegree(),this->getCoefficient()\*secondTerm.getCoefficient());

}

int compare(const void \*a,const void \*b){

//compare function to compare 2 Terms based on degree,sorts in descending order

Term \*p1=(Term\*)a;

Term \*p2=(Term\*)b;

return p2->getDegree()-p1->getDegree();

}

//------------------------------------------------------------------------------------

float splitString(string str){

std::string::size\_typesz;

bool isSplit=false;

string variable;

float coefficient;

for(inti=str.length()-1;i>-1;i--){

if(!isSplit){

try{

coefficient=atof (str.substr(0,i+1).c\_str());

//cout<<"i in split"<<i<<endl;

isSplit=true;

}catch(conststd::invalid\_argument&ia){

cout<<"failed\t";

}

}

else break;

}

//cout<<"op"<<coefficient<<"endOp"<<endl;

return coefficient;

}

//------------------------------------------------------------------------------------

class Polynomial{ //class Polynomial

vector<Term>termsInPolynomial;

public:

Polynomial(){}

Polynomial(string);

void printPolynomial();

//void createPolynomial(string s);

void insertTerm(Term term);

Polynomial addPolynomial(Polynomial);

Polynomial multiplyPolynomial(Polynomial);

void sortPolynomial();

int size(){return termsInPolynomial.size();}

float evaluatePolynomial(float);

};

float Polynomial::evaluatePolynomial(float value){

float answer=0;

for(inti=0;i<size();i++){

answer+=termsInPolynomial[i].evaluateTerm(value);

}

return answer;

}

Polynomial Polynomial::addPolynomial(Polynomial secondPolynomial){ //add 2 Polynomials

Polynomial resultant;

int p1Size=this->size(),p2Size=secondPolynomial.size();

inti=0,j=0;

while(i<p1Size||j<p2Size){

if(j==p2Size || this->termsInPolynomial[i].getDegree()>secondPolynomial.termsInPolynomial[j].getDegree()){

//1st polynomial's current degree is greater or 2nd polynomial ended

Term term(this->termsInPolynomial[i].getDegree(),0);

do{

term=term.addTerms(this->termsInPolynomial[i]);

i++;

}while(i<p1Size && this->termsInPolynomial[i].getDegree()==term.getDegree());

if(!(term.getDegree()==0&&term.getCoefficient()==0))

resultant.insertTerm(term);

}

else if(i==p1Size || secondPolynomial.termsInPolynomial[j].getDegree()>this->termsInPolynomial[i].getDegree()){

//2st polynomial's current degree is greater or 1st polynomial ended

Term term(secondPolynomial.termsInPolynomial[j].getDegree(),0);

do{

term=term.addTerms(secondPolynomial.termsInPolynomial[j]);

j++;

}while(j<p2Size &&secondPolynomial.termsInPolynomial[j].getDegree()==term.getDegree());

if(!(term.getDegree()==0&&term.getCoefficient()==0))

resultant.insertTerm(term);

}

else{

Term term(this->termsInPolynomial[i].getDegree(),0);

do{

term=term.addTerms(this->termsInPolynomial[i]);

i++;

}while(i<p1Size && this->termsInPolynomial[i].getDegree()==term.getDegree());

do{

term=term.addTerms(secondPolynomial.termsInPolynomial[j]);

j++;

}while(j<p2Size &&secondPolynomial.termsInPolynomial[j].getDegree()==term.getDegree());

if(!(term.getDegree()==0&&term.getCoefficient()==0))

resultant.insertTerm(term);

}

}

return resultant;

}

Polynomial Polynomial::multiplyPolynomial(Polynomial secondPolynomial){ //multiply 2 polynomials

Polynomial resultant;

for(inti=0;i<this->size();i++){

for(int j=0;j<secondPolynomial.size();j++){

Term term=this->termsInPolynomial[i].multiplyTerms(secondPolynomial.termsInPolynomial[j]);

resultant.insertTerm(term); //insert all products

}

}

resultant.sortPolynomial(); //sort the array

resultant=resultant.addPolynomial(Polynomial()); //add like terms

return resultant;

}

void Polynomial::sortPolynomial(){ //sort the terms in Polynomial in descending order

if(termsInPolynomial.size()>0){

Term\* terms=&termsInPolynomial[0];

qsort(terms,termsInPolynomial.size(),sizeof(Term),compare);

}

}

void Polynomial::insertTerm(Term term){ //intert a Term in Polynomial

termsInPolynomial.push\_back(term);

}

void Polynomial::printPolynomial(){ //print a Polynomial

float coefficient;

int degree;

for(inti=0;i<this->termsInPolynomial.size();i++){

coefficient=termsInPolynomial[i].getCoefficient();

degree=termsInPolynomial[i].getDegree();

if(coefficient!=1 ||(coefficient==1&&degree==0))

cout<<coefficient;

if(degree!=0){

cout<<"x";

if(degree!=1)

cout<<"^"<<degree;

}

cout<<" ";

if(i!=termsInPolynomial.size()-1)cout<<"+ ";

}

if(this->termsInPolynomial.size()==0)cout<<"Zero Polynomial";

cout<<endl;

}

Polynomial::Polynomial(string s){ //create a Polynomial from a string

vector<string>polynomialTerms;

stringstreamstrstream(s);

string intermediate;

while(getline(strstream,intermediate,'+')){

polynomialTerms.push\_back(intermediate);

}

for(inti=0;i<polynomialTerms.size();i++){

vector<string>args;

std::size\_tcurrent,previous=0;

current = polynomialTerms[i].find\_first\_of("^");

while(current!=std::string::npos){

args.push\_back(polynomialTerms[i].substr(previous,current-previous));

previous=current+1;

current=polynomialTerms[i].find\_first\_of("^",previous);

}

args.push\_back(polynomialTerms[i].substr(previous,current-previous));

/\*for(inti = 0; i<args.size(); i++){

cout<<args[i] << '\n';

//splitString(args[i]);

}\*/

float coefficient;

int degree;

if(isdigit(args[0][0])){

coefficient=splitString(args[0]);

}

else{

coefficient=1;

}

if(args.size()>1)

degree=splitString(args[1]);

else if(isalpha(args[0][args[0].length()-1]))

degree=1;

else

degree=0;

//cout<<"size:"<<args.size()<<endl<<endl;

//cout<<coefficient<<" "<<degree<<endl;

termsInPolynomial.push\_back(Term(degree,coefficient));

}

sortPolynomial();

}

//----------------------------------------------------------------------------------------

int main (){

string s[2];

float value;

Polynomial \*p[2];

inti=0;

do{

cout<<"Enter polynomial"<<i+1<<endl;

cin>>s[i];

p[i]=new Polynomial(s[i]);

i++;

}while(i<2);

//cout<<endl;

i=0;do{

\*p[i]=p[i]->addPolynomial(Polynomial());

cout<<"\nPolynomial "<<i+1<<"\t";

p[i]->printPolynomial();

cout<<"Enter value of variable\t";

cin>>value;

cout<<p[i++]->evaluatePolynomial(value)<<endl;

}while(i<2);

Polynomial sum=p[0]->addPolynomial(\*p[1]);

cout<<"\nSum of Polynomials\n";

sum.printPolynomial();

//cout<<" = "sum.evaluatePolynomial();

Polynomial product=p[0]->multiplyPolynomial(\*p[1]);

cout<<"\nProduct of Polynomials\n";

product.printPolynomial();

return 0;

}

/\*

\*ui 1.0

Term squareTerm1(3,45.97),squareTerm2(2,34.55);

Term sumOfSquareTerms=squareTerm1.addTerms(squareTerm2);

cout<<sumOfSquareTerms.getCoefficient()<<" x^"<<sumOfSquareTerms.getDegree()<<endl;

Term productOfTwoTerms=squareTerm1.multiplyTerms(squareTerm2);

cout<<productOfTwoTerms.getCoefficient()<<" x^"<<productOfTwoTerms.getDegree()<<endl;

Polynomial p1,p2;

p1.insertTerm(squareTerm1);

p2.insertTerm(squareTerm2);

p1.insertTerm(squareTerm2);

p1.insertTerm(squareTerm2);

p1.insertTerm(Term(1,88.4));

p2.insertTerm(Term(7,55.77));

p1.sortPolynomial();

p2.sortPolynomial();

p1.printPolynomial();

p1=p1.addPolynomial(Polynomial());

p1.printPolynomial();

p2.printPolynomial();

Polynomial sum=p2.addPolynomial(p1);

sum.printPolynomial();

p1.printPolynomial();

p2.printPolynomial();

Polynomial product=p1.multiplyPolynomial(p1);

//product.sortPolynomial();

product.printPolynomial();

string ipString;

cin>>ipString;

Polynomial myP(ipString);

myP.printPolynomial();

\*/

***Output:***

