## Po.h



## Po.cpp



## Polynomial.h



## Polynomial.cpp

#include "polynomial.h"

Polynomial::Polynomial() = default;

Polynomial::Polynomial(const double& poVal, const int& poExp) {

Monom temp(poVal, poExp);

addPoly(temp);

}

Polynomial::Polynomial(const Polynomial& P) {

\_head = nullptr;

\_rear = nullptr;

Po\* temp = P.get\_Head();

\_head = new Monom(\*temp);

\_head->\_next = nullptr;

\_rear = \_head;

temp = temp->\_next;

while (temp != nullptr) {

Po\* p = new Monom(\*temp);

p->\_next = \_rear->\_next;

\_rear->\_next = p;

\_rear = \_rear->\_next;

temp = temp->\_next;

}

}

std::ostream& operator<<(std::ostream& out, const Polynomial& P) {

Po\* list = P.get\_Head();

out << "Length of polynomial : "

<< P.retLength() <<

std::endl;

out << "Polynomial (coefficient, power of exponent): \n";

while (list) {

out << list->getVal() << " " << list->getExp() << " |";

list = list->\_next;

}

out << "\n";

return out;

}

void Polynomial::enterNape(const double& poVal, const int& poExp) {

Monom temp(poVal, poExp);

addPoly(temp);

}

// get \_head;

Po\* Polynomial::get\_Head() const {

return \_head;

}

// operator

Polynomial& Polynomial::operator=(const Polynomial& P) {

\_head = nullptr;

\_rear = nullptr;

Po\* temp = P.get\_Head();

\_head = new Monom(\*temp);

\_head->\_next = nullptr;

\_rear = \_head;

temp = temp->\_next;

while (temp != nullptr) {

Po\* p = new Monom(\*temp);

p->\_next = \_rear->\_next;

\_rear->\_next = p;

\_rear = \_rear->\_next;

temp = temp->\_next;

}

return \*this;

}

void Polynomial::addPoly(Monom& M) {

StructOfMon tepLinkNode = new Monom(M);

if (\_head == nullptr) {

\_head = tepLinkNode;

\_head->\_next = nullptr;

\_rear = \_head;

}

else {

tepLinkNode->\_next = \_rear->\_next;

\_rear->\_next = tepLinkNode;

\_rear = \_rear->\_next;

}

}

void Polynomial::add(const Polynomial& P1, Polynomial& P2) {

Polynomial addP(P1);

Polynomial addS(P2);

addP.MergeSort\_FindRear();

addS.MergeSort\_FindRear();

Po\* pp = addP.get\_Head();

Po\* ps = addS.get\_Head();

//\_head = nullptr;

Po\* ph = \_head;

while (pp != nullptr && ps != nullptr) {

if (pp->getExp() == ps->getExp()) {

double sum = pp->getVal() + ps->getVal();

if (sum) {

Po\* ak = new Monom(sum, pp->getExp());

pp = pp->\_next;

ps = ps->\_next;

ak->\_next = nullptr;

ph->\_next = ak;

ph = ak;

}

else {

pp = pp->\_next;

ps = ps->\_next;

}

}

else {

if (pp->getExp() > ps->getExp()) {

Po\* ak = new Monom(ps->getVal(), ps->getExp() );

ps = ps->\_next;

ak->\_next = nullptr;

ph->\_next = ak;

ph = ak;

}

else {

Po\* ak = new Monom(pp->getVal(), pp->getExp());

pp = pp->\_next;

ak->\_next = nullptr;

ph->\_next = ak;

ph = ak;

}

}

}

Po\* np;

np = pp ? pp : ps;

while (np != nullptr) {

Po\* ak = new Monom(\*np);

ak->\_next = nullptr;

ph->\_next = ak;

ph = ak;

\_rear = ak;

np = np->\_next;

}

\_head = \_head->\_next;

//while (\_head) {

// std::cout << \_head->getVal();

// \_head = \_head->\_next;

//}

//MergeSort\_FindRear();

}

void Polynomial::negative( Polynomial& P) {

Po\* n\_ps = P.get\_Head();

while (n\_ps != nullptr) {

n\_ps->enterVal((-1) \* n\_ps->getVal());

n\_ps = n\_ps->\_next;

}

}

void Polynomial::subtract(const Polynomial& P1, Polynomial& P2) {

negative(P2);

add(P1, P2);

negative(P2);

}

void Polynomial::multiply(const Polynomial& P1, const Polynomial& P2) {

Polynomial addP(P1);

Polynomial addS(P2);

addP.MergeSort\_FindRear();

addS.MergeSort\_FindRear();

Po\* pp = addP.get\_Head();

Po\* ps = addS.get\_Head();

Po\* nodeLink;

nodeLink = pp;

Po\* ph;

ph = \_head;

while (nodeLink) {

Po\* tempLink;

tempLink = ps;

while (tempLink) {

double val = nodeLink->getVal() \* tempLink->getVal();

int exp = nodeLink->getExp() + tempLink->getExp();

Po\* p = new Monom(val, exp);

p->\_next = nullptr;

ph->\_next = p;

ph = p;

tempLink = tempLink->\_next;

}

nodeLink = nodeLink->\_next;

}

\_head = \_head->\_next;

MergeSort\_FindRear();

}

int Polynomial::retLength() const {

Po\* faHead;

faHead = get\_Head();

int leng = 0;

while (faHead) {

leng++;

faHead = faHead->\_next;

}

return leng;

}

void Polynomial::reverse() {

//

Po\* ph;

ph = get\_Head();

Po\* h;

h = nullptr;

while (ph) {

Po\* t = ph;

ph = ph->\_next;

t->\_next = h;

h = t;

}

\_head = h;

ph = \_head;

while (ph) {

\_rear = ph;

ph = ph->\_next;

}

}

void Polynomial::printInfo() const {

StructOfMon travelLink = \_head;

while (travelLink != nullptr) {

std::cout << travelLink->getVal() << " " << travelLink->getExp() << std::endl;

travelLink = travelLink->\_next;

}

}

Po\* Polynomial::merge(Po\* pHead1, Po\* pHead2) {

if (pHead1 == nullptr) {

return pHead2;

}

if (pHead2 == nullptr) {

return pHead1;

}

Po\* falseHead = new Po(0, 0);

Po\* cur = falseHead;

while (pHead1 && pHead2) {

if (pHead1->getExp() < pHead2->getExp()) {

cur->\_next = pHead1;

pHead1 = pHead1->\_next;

}

else if (pHead1->getExp() == pHead2->getExp()) {

pHead1->enterVal(pHead1->getVal() + pHead2->getVal());

Po\* temp = pHead2;

cur->\_next = pHead1;

pHead1 = pHead1->\_next;

pHead2 = pHead2->\_next;

delete temp;

}

else {

cur->\_next = pHead2;

pHead2 = pHead2->\_next;

}

cur = cur->\_next;

}

cur->\_next = pHead1 ? pHead1 : pHead2;

Po\* lastp = falseHead->\_next;

delete falseHead;

return lastp;

}

Po\* Polynomial::mergeSort(Po\* nowLink) {

if (nowLink == nullptr || nowLink->\_next == nullptr) return nowLink;

Po\* left = nowLink;

Po\* mid = nowLink->\_next;

Po\* right = mid->\_next;

while (right != nullptr && right->\_next != nullptr) {

left = left->\_next;

mid = mid->\_next;

right = right->\_next->\_next;

}

left->\_next = nullptr;

return merge(mergeSort(nowLink), mergeSort(mid));

}

void Polynomial::MergeSort\_FindRear() {

Po\* startHead = \_head;

Po\* lastHead = mergeSort(startHead);

\_head = lastHead;

while (lastHead->\_next != nullptr) {

lastHead = lastHead->\_next;

}

\_rear = lastHead;

}

Polynomial::~Polynomial() {

Po\* temp = \_head;

while (\_head != nullptr) {

temp = \_head;

\_head = \_head->\_next;

delete temp;

}

\_rear = \_head;

}

## Datasources.h



## Datasources.cpp

#include "datasources.h"

int DataSource::getIntData() {

std::default\_random\_engine \_eg(std::time(0));

std::uniform\_int\_distribution<int> \_ui(-10000, 10000);

Sleep(1000);

return \_ui(\_eg);

}

double DataSource::getDoubleData() {

std::default\_random\_engine \_eg(std::time(0));

std::uniform\_real\_distribution<double> \_ur(-10000, 10000);

Sleep(1000);

return \_ur(\_eg);

}

void DataSource::generatePolynomial() {

std::ofstream datatext;

datatext.open("datas.txt", std::ios::out);

if (!datatext.is\_open()) {

std::cout << "error about opening file!" << std::endl;

exit(0);

}

datatext << 10 << "\n";

number Number = 10;

for (int i = 0; i < 5; ++i) {

int P\_length = (i + 1) \* 5;

datatext << P\_length << "\n";

// get error: --pre not sif--

while (P\_length--) {

number \_llnt = 1;

while (\_llnt--) {

datatext << getDoubleData() << " " << getIntData() << " ";

}

}

datatext << "\n";

}

for (int i = 0; i < 5; ++i) {

int P\_length = (i + 1) \* 10;

datatext << P\_length << "\n";

while (P\_length --) {

datatext << getDoubleData() << " " << getIntData() << " ";

}

datatext << "\n";

}

datatext.close();

}

Polynomial formula[\_MAX\_LENGTH + 1];

void getAllPolynomial() {

std::ifstream datatext;

datatext.open("datas.txt", std::ios::in);

if (!datatext.is\_open()) {

std::cout << "error file open" << std::endl;

return ;

}

else {

std::cout << "ok" << std::endl;

}

int \_cnt = 0;

datatext >> \_cnt;

while (\_cnt--) {

double \_val;

int \_exp;

int \_poCnt = 0;

datatext >> \_poCnt;

// error poCnt

//\_poCnt--;

while (\_poCnt--) {

datatext >> \_val >> \_exp;

formula[\_cnt].enterNape(\_val, \_exp);

//std::cout<< \_cnt <<" " << \_poCnt << " " << \_val << " " << \_exp << std::endl;

}

formula[\_cnt].MergeSort\_FindRear();

}

datatext.close();

}

## Testitem.p



## Testitem.cpp

#include "testitem.h"

void testSample\_\_TimeKeeping() {

std::clock\_t startTime, endTime;

startTime = std::clock();

for (int i = 0; i < 343444; ++i) {

std::cout << i << std::endl;

}

endTime = std::clock();

std::cout << (double)(endTime - startTime) / CLOCKS\_PER\_SEC << std::endl;

}

void test\_WritTxt() {

std::ofstream datatext;

datatext.open("resultText.txt", std::ios::out);

if (!datatext.is\_open()) {

std::cout << "error about opening file!" << std::endl;

exit(0);

}

std::ofstream datadd;

datadd.open("hasadd.txt", std::ios::out);

if (!datadd.is\_open()) {

std::cout << "error about opening file!" << std::endl;

exit(0);

}

//std::ofstream datasub;

//datasub.open("hassub.txt", std::ios::out);

//if (!datasub.is\_open()) {

// std::cout << "error about opening file!" << std::endl;

// exit(0);

//}

//std::ofstream datamulit;

//datamulit.open("resultText.txt", std::ios::out);

//if (!datamulit.is\_open()) {

// std::cout << "error about opening file!" << std::endl;

// exit(0);

//}

int \_\_\_cnt = 1;

for (int pre = 0; pre < \_MAX\_LENGTH; ++pre) {

for (int sif = 0; sif < \_MAX\_LENGTH; ++sif) {

if (pre == sif) continue;

datatext << "------------------------"

<< \_\_\_cnt++

<< "--------------------------\n";

std::clock\_t \_startTime, \_endTime;

datatext << "pre polynomial: \n";

datatext << formula[pre];

datatext << "sif polynomial \n";

datatext << formula[sif];

Polynomial \_res(0, 0);

datadd << formula[pre].retLength() << " + " << formula[sif].retLength() << " :";

\_startTime = std::clock();

\_res.add(formula[pre], formula[sif]);

datatext << "add about polynomial \n";

datatext << \_res;

\_res.subtract(formula[pre], formula[sif]);

datatext << "subtract about polynomial \n";

datatext << \_res;

\_res.multiply(formula[pre], formula[sif]);

datatext << "multiply about polynomial \n";

datatext << \_res;

datatext << "\n\n";

for (int k = 0; k <= 1000; ++k) {

\_res.add(formula[pre], formula[sif]);

\_res.subtract(formula[pre], formula[sif]);

\_res.multiply(formula[pre], formula[sif]);

}

\_endTime = std::clock();

datadd << ((double)(\_endTime - \_startTime) / CLOCKS\_PER\_SEC) << "\n";

}

}

datatext.close();

}