Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

Механіко-технологічний факультет

Кафедра кібербезпеки та програмного забезпечення

Дисципліна: Об’єктно-орієнтоване програмування

**Лабораторна робота №2**

**Тема:** **«КОНСТРУКТОРИ І ДЕСТРУКТОРИ»**

|  |
| --- |
| Виконав: ст. гр. КI-23 |
| Чепіль В.О. |
| Перевірив викладач:  Козірова Н. Л. |
|  |

Кропивницький 2024

**Тема:** Конструктори і деструктори

**Мета:** ознайомитись з основними поняттями конструктор і деструктор в ООП та навчитись їх програмно реалізовувати мовою С++.

**Варіант:** 4

**Завдання:**

**Завдання 1**

1. Реалізуйте конструктор за замовчуванням, конструктор з параметрами та копіюючий конструктор для вашого класу з лабораторної роботи 1.

• Конструктор за замовчуванням має встановлювати значення полів за замовчуванням.

• Конструктор з параметрами має приймати значення для кожного поля.

• Копіюючий конструктор має копіювати значення полів з іншого об'єкта класу.

2. Реалізуйте деструктор для класу. Деструктор має виводити повідомлення про знищення об'єкта.

3. У функції main створіть об'єкт за допомогою конструктора за замовчуванням та виведіть значення його полів.

4. Створіть новий об'єкт за допомогою конструктора з параметрами та встановіть значення для полів. Виведіть значення полів цього об'єкта.

5. Створіть ще один об'єкт і скопіюйте значення полів з першого об'єкта за допомогою копіюючого конструктора. Виведіть значення полів цього об'єкта.

6. Завершіть функцію main, що призведе до виходу з області видимості створених об'єктів і виклику їх деструкторів. Переконайтесь, що повідомлення про знищення об'єктів виводяться.  
**Завдання 2**

Розробіть клас «Многочлен» – Polynom ступеня 𝑛. Клас повинен містити конструктори. Реалізуйте методи для обчислення значення многочлена, додавання, віднімання та множення многочленів. Створіть масив об'єктів класу Polynom. Передайте його у функцію, яка обчислює суму многочленів масиву та повертає результуючий многочлен, який виводиться на екран у головній програмі.

**Лістинг завдання 1(Task\_1):**

main.cpp:

#include <Employee.h>

using namespace std;

int main()

{

// Використання значень за замовчуванням

Employee Human1;

cout << "Name human1: " << Human1.getName() << endl;

cout << "Id human1: " << Human1.getId() << endl;

cout << "Salary human1: " << Human1.getSalary() << "$" << endl;

cout << " " << endl;

// Створення об'єкта з параметрами

Employee Human2("Oleg", 5454545, 50000);

cout << "Name human2: " << Human2.getName() << endl;

cout << "Id human2: " << Human2.getId() << endl;

cout << "Salary human2: " << Human2.getSalary() << "$" << endl;

cout << " " << endl;

// Копіюювання даних об'єкта

Employee Human3(Human1);

cout << "Name human3: " << Human3.getName() << endl;

cout << "Id human3: " << Human3.getId() << endl;

cout << "Salary human3: " << Human3.getSalary() << "$" << endl;

cout << " " << endl;

return 0;

}

Employee.cpp:

#include "employee.h"

void Employee::setName(string newName)

{

name = newName;

}

string Employee::getName()

{

return name;

}

void Employee::setId(int newId)

{

id = newId;

}

int Employee::getId()

{

return id;

}

void Employee::setSalary(int newSalary)

{

salary = newSalary;

}

int Employee::getSalary()

{

return salary;

}

Employee.h:

#ifndef EMPLOYEE\_H

#define EMPLOYEE\_H

#include <string>

#include <iostream>

using namespace std;

class Employee

{

public:

// Конструктор за замовчуванням

Employee() {

name = "Vadim";

id = 44557997;

salary = 10000;

cout << "Default constructor called!" << endl;

}

// Конструктор з параметрами

Employee(string name, int id, int salary) {

this->name = name;

this->id = id;

this->salary = salary;

cout << "Constructor with parameters called!" << endl;

}

// Конструктор копіюючий

Employee(const Employee& other) {

name = other.name;

id = other.id;

salary = other.salary;

cout << "Copy constructor called!" << endl;

}

// Деструктор

~Employee() {

cout << "Destructor called!" << endl;

}

void setName(string newName);

string getName();

void setId(int newId);

int getId();

void setSalary(int newSalary);

int getSalary();

private:

string name;

int id;

int salary;

};

#endif // EMPLOYEE\_H

**Лістинг завдання 2(Task\_2):**

main.cpp:

#include "polynom.h"

#include <iostream>

int main() {

Polynom polynom1("-3x^2 + 3xy^2 - 30x - 40 - 5y - 3y^2 - 15xy");

Polynom polynom2("x^2 - 4x^2 + 5x - 5 - 35 + y + 2y^2 + 10xy - 5xy^2");

Polynom polynom3("2x -x");

Polynom polynom4("x + 1");

string resultAdd = solutionPolynomials(polynom3, '+', polynom4);

cout << "result add : " << resultAdd << endl;

string resultMultiply = solutionPolynomials(polynom3, '\*', polynom4);

cout << "result multiply : " << resultMultiply << endl;

string resultSubtract = solutionPolynomials(polynom3, '-', polynom4);

cout << "result subtruct : " << resultSubtract << endl;

return 0;

}

Polynom.cpp:

#include "polynom.h"

#include <iostream>

Polynom::Polynom() {}

Polynom::Polynom(const string& expression)

{

correctData = checkData(expression);

if (correctData != "Invalid data!")

{

divisionIntoGroups(correctData);

}

else

{

cout << correctData << endl;

}

}

string Polynom::checkData(const string& getData)

{

char allowSymbols[] = {'x', 'y', 'z', '-', '+', '\*', '^'};

bool checkSymbol = false;

string correctData;

for (char c : getData) {

if (isdigit(c)) {

checkSymbol = true;

correctData += c;

} else {

checkSymbol = false;

for (char d : allowSymbols) {

if (c == d) {

correctData += c;

checkSymbol = true;

break;

}

}

}

if (!checkSymbol && c != ' ')

{

return "Invalid data!";

}

}

return correctData;

}

void Polynom::divisionIntoGroups(const string& polynom)

{

string currentTerm;

for (size\_t i = 0; i < polynom.size(); ++i)

{

char c = polynom[i];

if (c == '+' || c == '-')

{

if (!currentTerm.empty())

{

if (currentTerm.find("xy^") != string::npos)

{

xyPowerGroup.push\_back(currentTerm);

}

else if (currentTerm.find("xy") != string::npos)

{

xyGroup.push\_back(currentTerm);

}

else if (currentTerm.find("x^") != string::npos)

{

xPowerGroup.push\_back(currentTerm);

}

else if (currentTerm.find("y^") != string::npos)

{

yPowerGroup.push\_back(currentTerm);

}

else if (currentTerm.find("x") != string::npos)

{

xGroup.push\_back(currentTerm);

}

else if (currentTerm.find("y") != string::npos)

{

yGroup.push\_back(currentTerm);

}

else

{

constantGroup.push\_back(currentTerm);

}

}

currentTerm = c;

}

else

{

currentTerm += c;

}

}

if (!currentTerm.empty())

{

if (currentTerm.find("xy^") != string::npos)

{

xyPowerGroup.push\_back(currentTerm);

}

else if (currentTerm.find("xy") != string::npos)

{

xyGroup.push\_back(currentTerm);

}

else if (currentTerm.find("x^") != string::npos)

{

xPowerGroup.push\_back(currentTerm);

}

else if (currentTerm.find("y^") != string::npos)

{

yPowerGroup.push\_back(currentTerm);

}

else if (currentTerm.find("x") != string::npos)

{

xGroup.push\_back(currentTerm);

}

else if (currentTerm.find("y") != string::npos)

{

yGroup.push\_back(currentTerm);

}

else

{

constantGroup.push\_back(currentTerm);

}

}

}

void Polynom::changeSignInGroups() {

for (auto& term : xGroup) {

changeSign(term);

}

for (auto& term : yGroup) {

changeSign(term);

}

for (auto& term : xPowerGroup) {

changeSign(term);

}

for (auto& term : yPowerGroup) {

changeSign(term);

}

for (auto& term : xyGroup) {

changeSign(term);

}

for (auto& term : xyPowerGroup) {

changeSign(term);

}

for (auto& term : constantGroup) {

changeSign(term);

}

}

void Polynom::changeSign(string& term) {

if (term[0] == '+') {

term[0] = '-';

}

else if (term[0] == '-') {

term[0] = '+';

}

else {

term = "-" + term;

}

}

string Polynom::processSimpleGroup(const vector<string>& group, char var, char action) const

{

int sum = 0;

for (const auto& term : group)

{

char sign = '+';

int coefficient = 0;

bool hasNumber = false;

for (char ch : term)

{

if (isdigit(ch))

{

coefficient = coefficient \* 10 + (ch - '0');

hasNumber = true;

}

else if (ch == '+' || ch == '-')

{

sign = ch;

}

}

if (!hasNumber)

{

coefficient = 1;

}

if (action == '+')

{

sum += (sign == '-') ? -coefficient : coefficient;

}

else if (action == '-')

{

sum += (sign == '-') ? coefficient : -coefficient;

}

}

string result;

if (sum < 0)

{

result = to\_string(sum) + var + " ";

}

else if (sum > 0)

{

result = "+" + to\_string(sum) + var + " ";

}

return result;

}

string Polynom::processGroupWithPower(const vector<string>& group, char var1, char var2, char action, bool isPower) const

{

int sum = 0;

for (const auto& term : group)

{

char sign = '+';

int coefficient = 0;

bool hasNumber = false;

for (char ch : term)

{

if (isdigit(ch))

{

coefficient = coefficient \* 10 + (ch - '0');

hasNumber = true;

}

else if (ch == '+' || ch == '-')

{

sign = ch;

}

else if (ch == '^')

{

break;

}

}

if (!hasNumber)

{

coefficient = 1;

}

if (action == '+')

{

sum += (sign == '-') ? -coefficient : coefficient;

}

else if (action == '-')

{

sum += (sign == '-') ? coefficient : -coefficient;

}

}

string result;

if (sum != 0)

{

if (isPower && var2 != 'q')

{

result = (sum < 0) ? to\_string(sum) + var1 + var2 + "^2 " : "+" + to\_string(sum) + var1 + var2 + "^2 ";

}

else if (!isPower && var2 != 'q')

{

result = (sum < 0) ? to\_string(sum) + var1 + var2 + " " : "+" + to\_string(sum) + var1 + var2 + " ";

}

else

{

result = (sum < 0) ? to\_string(sum) + var1 + "^2 " : "+" + to\_string(sum) + var1 + "^2 ";

}

}

return result;

}

string Polynom::add(const Polynom& other) const

{

if (correctData == "Invalid data!")

{

return "Invalid data!";

}

Polynom resultAdd;

resultAdd.divisionIntoGroups(this->correctData);

resultAdd.divisionIntoGroups(other.correctData);

string resultString;

int sumConstant = 0;

for (const string& term : resultAdd.constantGroup)

{

sumConstant += stoi(term);

}

if (sumConstant != 0)

{

resultString += (sumConstant < 0) ? to\_string(sumConstant) + " " : "+" + to\_string(sumConstant) + " ";

}

resultString += processSimpleGroup(resultAdd.xGroup, 'x', '+');

resultString += processSimpleGroup(resultAdd.yGroup, 'y', '+');

resultString += processGroupWithPower(resultAdd.xPowerGroup, 'x', 'q', '+', true);

resultString += processGroupWithPower(resultAdd.yPowerGroup, 'y', 'q', '+', true);

resultString += processGroupWithPower(resultAdd.xyGroup, 'x', 'y', '+', false);

resultString += processGroupWithPower(resultAdd.xyPowerGroup, 'x', 'y', '+', true);

if (resultString.back() == ' ')

{

resultString.pop\_back();

}

return resultString;

}

string Polynom::subtract(const Polynom& other) const

{

Polynom subtract;

Polynom resultSubtract;

subtract.divisionIntoGroups(other.correctData);

subtract.changeSignInGroups();

resultSubtract.divisionIntoGroups(this->correctData);

resultSubtract.constantGroup.insert(resultSubtract.constantGroup.end(), subtract.constantGroup.begin(), subtract.constantGroup.end());

resultSubtract.xGroup.insert(resultSubtract.xGroup.end(), subtract.xGroup.begin(), subtract.xGroup.end());

resultSubtract.yGroup.insert(resultSubtract.yGroup.end(), subtract.yGroup.begin(), subtract.yGroup.end());

resultSubtract.xPowerGroup.insert(resultSubtract.xPowerGroup.end(), subtract.xPowerGroup.begin(), subtract.xPowerGroup.end());

resultSubtract.yPowerGroup.insert(resultSubtract.yPowerGroup.end(), subtract.yPowerGroup.begin(), subtract.yPowerGroup.end());

resultSubtract.xyGroup.insert(resultSubtract.xyGroup.end(), subtract.xyGroup.begin(), subtract.xyGroup.end());

resultSubtract.xyPowerGroup.insert(resultSubtract.xyPowerGroup.end(), subtract.xyPowerGroup.begin(), subtract.xyPowerGroup.end());

string resultString;

int sumConstant = 0;

for (const string& term : resultSubtract.constantGroup)

{

sumConstant += stoi(term);

}

if (sumConstant != 0)

{

resultString += (sumConstant < 0) ? to\_string(sumConstant) + " " : "+" + to\_string(sumConstant) + " ";

}

resultString += processSimpleGroup(resultSubtract.xGroup, 'x', '+');

resultString += processSimpleGroup(resultSubtract.yGroup, 'y', '+');

resultString += processGroupWithPower(resultSubtract.xPowerGroup, 'x', 'q', '+', true);

resultString += processGroupWithPower(resultSubtract.yPowerGroup, 'y', 'q', '+', true);

resultString += processGroupWithPower(resultSubtract.xyGroup, 'x', 'y', '+', false);

resultString += processGroupWithPower(resultSubtract.xyPowerGroup, 'x', 'y', '+', true);

if (resultString.back() == ' ')

{

resultString.pop\_back();

}

return resultString;

}

string Polynom::multiply(const Polynom& other) const

{

Polynom result;

for (const auto& term1 : xGroup) {

for (const auto& term2 : other.xGroup) {

result.xPowerGroup.push\_back(multiplyTerms(term1, term2, 'x', true));

}

}

for (const auto& term1 : xGroup) {

for (const auto& term2 : other.constantGroup) {

result.xGroup.push\_back(multiplyTerms(term1, term2, 'x', false));

}

}

for (const auto& term1 : constantGroup) {

for (const auto& term2 : other.constantGroup) {

result.constantGroup.push\_back(multiplyTerms(term1, term2, ' ', false));

}

}

string resultString;

resultString += processGroupWithPower(result.xPowerGroup, 'x', 'q', '+', true);

resultString += processSimpleGroup(result.xGroup, 'x', '+');

resultString += processSimpleGroup(result.constantGroup, ' ', '+');

if (resultString.back() == ' ') {

resultString.pop\_back();

}

return resultString;

}

string Polynom::multiplyTerms(const string& term1, const string& term2, char var, bool isPower) const

{

int coef1 = extractCoefficient(term1);

int coef2 = extractCoefficient(term2);

int resultCoef = coef1 \* coef2;

if (isPower) {

return to\_string(resultCoef) + var + "^2";

} else if (var == ' ') {

return to\_string(resultCoef);

} else {

return to\_string(resultCoef) + var;

}

}

int Polynom::extractCoefficient(const string& term) const

{

int coefficient = 0;

bool negative = false;

for (char ch : term) {

if (ch == '-') {

negative = true;

} else if (isdigit(ch)) {

coefficient = coefficient \* 10 + (ch - '0');

}

}

coefficient = coefficient == 0 ? 1 : coefficient;

return negative ? -coefficient : coefficient;

}

string solutionPolynomials(const Polynom &value1, char action, const Polynom &value2)

{

if (action == '+')

{

return value1.add(value2);

}

else if (action == '-')

{

return value1.subtract(value2);

}

else if (action == '\*')

{

return value1.multiply(value2);

}

return "Invalid operation!";

}

polynom.h:

#ifndef POLYNOM\_H

#define POLYNOM\_H

#include <string>

#include <vector>

using namespace std;

class Polynom {

public:

Polynom();

Polynom(const string& expression);

string add(const Polynom& other) const;

string subtract(const Polynom& other) const;

string multiply(const Polynom& other) const;

private:

string correctData;

vector<string> xGroup;

vector<string> yGroup;

vector<string> xPowerGroup;

vector<string> yPowerGroup;

vector<string> xyGroup;

vector<string> xyPowerGroup;

vector<string> constantGroup;

string checkData(const string& getData);

void divisionIntoGroups(const string& polynom);

void changeSignInGroups();

void changeSign(string& term);

string processSimpleGroup(const vector<string>& group, char var, char action) const;

string processGroupWithPower(const vector<string>& group, char var1, char var2, char action, bool isPower) const;

string multiplyTerms(const string& term1, const string& term2, char var, bool isPower) const;

int extractCoefficient(const string& term) const;

};

string solutionPolynomials(const Polynom &value1, char action, const Polynom &value2);

#endif // POLYNOM\_H

**Висновок:**

Конструктори та деструктори є ключовими елементами об'єктно-орієнтованого програмування на C++. Конструктор автоматично запускається при створенні об'єкта для його початкового налаштування, а деструктор викликається під час знищення об'єкта для звільнення ресурсів. Існують різні типи конструкторів: за замовчуванням, параметризований і конструктор копіювання. Деструктор виконується автоматично, гарантуючи правильне звільнення ресурсів і запобігання витокам пам'яті. Отже, конструктори і деструктори сприяють керуванню життєвим циклом об'єктів і раціональному використанню ресурсів програми.