**Лабораторна робота №11**

**Приклади створення класів**

*Мета роботи*: набуття навичок роботи з класами у мові С++.

*Теоретичні основи*:

*class <тег\_класу>*

*{*

* *визначення даних-членів класу*

*private: // захищені члени класу - дані*

*<тип\_1> <ідентифікатор\_1>;*

*<тип\_2> <ідентифікатор\_2>;*

*…*

* *декларації функцій-членів класу public: //відкриті члени класу - функції*

*<тип\_результату> <ідентифікатор\_1> (<параметри\_функції>); <тип\_результату> <ідентифікатор\_2> (<параметри\_функції>);*

*…*

*};*

***Порядок виконання роботи:***

1. Описати клас *ДАТА* та визначити необхідні дії для цього класу.

*#pragma once*

*#include "Header.h"*

class Date {

private:

int day;

int month;

int year;

public:

Date() {

this->day = 0;

this->month = 0;

this->year = 0;

}

Date(int day, int month, int year) {

this->day = day;

this->month = month;

this->year = year;

}

Date(Date& date) {

this->day = date.day;

this->month = date.month;

this->year = date.year;

}

void setDay(int day) {

this->day = day;

}

void setMonth(int month) {

this->month = month;

}

void setYear(int year) {

this->year = year;

}

int getDay() {

return day;

}

int getMonth() {

return month;

}

int getYear() {

return year;

}

virtual void show() {

cout << "Date: " << this->getDay() << ":" << this->getMonth() << ":" << this->getYear() << endl;

cout << endl;

}

virtual void input() {

cout << "Enter day: ";

this->day = GetNumber\_I();

cout << "Enter month: ";

this->month = GetNumber\_I();

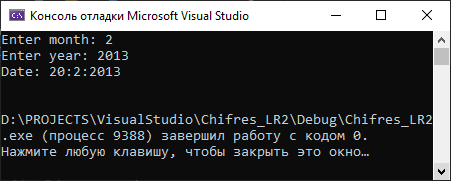
cout << "Enter year: ";

this->year = GetNumber\_I();

}

};

Результат роботи програми наведений нижче:



2. Описати клас *ЧАС* та визначити необхідні дії для цього класу.

*#pragma once*

*#include "Header.h"*

class Time

{

private:

int hours;

int minutes;

int seconds;

public:

Time() {

this->hours = 0;

this->minutes = 0;

this->seconds = 0;

}

Time(int hours, int minutes, int seconds) {

this->hours = hours;

this->minutes = minutes;

this->seconds = seconds;

}

Time(Time& time) {

this->hours = time.hours;

this->minutes = time.minutes;

this->seconds = time.seconds;

}

void setHours(int hours) {

this->hours = hours;

}

void setMinutes(int minutes) {

this->minutes = minutes;

}

void setSeconds(int seconds) {

this->seconds = seconds;

}

int getHours() {

return hours;

}

int getSeconds() {

return seconds;

}

int getMinutes() {

return minutes;

}

virtual void input() {

cout << "Enter hours: ";

this->hours = GetNumber\_I();

cout << "Enter minutes: ";

this->minutes = GetNumber\_I();

cout << "Enter seconds: ";

this->seconds = GetNumber\_I();

}

virtual void show() {

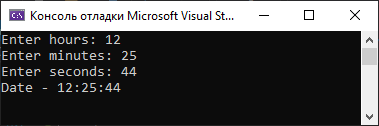
cout << "Date - " << this->getHours() << ":" << this->getMinutes() << ":" << this->getSeconds() << endl;

cout << endl;

}

};

Результат роботи програми наведений нижче:



3. Описати клас *КОМПЛЕКСНЕ ЧИСЛО* та визначити необхідні дії для цього класу.

*#pragma once*

*#include "Header.h"*

class Complex {

private:

float real;

float imaginary;

public:

Complex() {

this->imaginary = 0;

this->real = 0;

}

Complex(float real, float imaginary) {

this->real = real;

this->imaginary = imaginary;

}

Complex(Complex& complex) {

this->imaginary = complex.imaginary;

this->real = complex.real;

}

void setReal(float real) {

this->real = real;

}

void setImaginary(float imaginary) {

this->imaginary = imaginary;

}

float getReal() {

return real;

}

float getImaginary() {

return imaginary;

}

virtual void show() {

cout << this->getReal() << " + " << this->getImaginary() << "i";

}

virtual void input() {

cout << "Enter real: ";

real = GetNumber\_I();

cout << "Enter imaginary: ";

imaginary = GetNumber\_I();

}

void operator += (Complex complex)

{

this->setReal(this->getReal() + complex.getReal());

this->setImaginary(this->getImaginary() + complex.getImaginary());

}

Complex operator \* (int number)

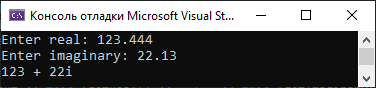
{

return Complex(this->getReal() \* number, this->getImaginary() \* number);

}

};

Результат роботи програми наведений нижче:



**Лабораторна робота №12**

**Статичні члени даних у класі**

*Мета роботи*: набуття навичок роботи зі статичними членами даних класів у мові С++.

*Теоретичні основи*:

Статичний член даних в класі оголошується з допомогою ключового слова ***static***. Оскільки, це є статичний член даних, то потрібно його визначити (описати змінну) за межами класу. Доступ до цього статичного члена даних мають усі екземпляри (об’єкти) класу.

Щоб використовувати статичний член даних у класі, потрібно:

* оголосити статичний член даних у тілі класу (у будь-якому розділі класу);
* визначити статичний член даних за межами класу.

*Приклад 12.1.* У класі CMyClass оголошується статичний член даних з іменем d цілого типу.

// оголошення класу

class CMyClass

{

static int d; // статичний член даних - тільки оголошення

public:

CMyClass(void); // конструктор

~CMyClass(void); // деструктор

// методи доступу до статичного члена даних

int Get(void) { return d; }

void Set(int nd) { d = nd; }

};

// визначення статичного члена даних за межами класу

int CMyClass::d; // виділяється пам’ять

// явно заданий конструктор за замовчуванням

CMyClass::CMyClass(void)

{

d = 0;

}

// деструктор

CMyClass::~CMyClass(void)

{

}

Як видно з вищенаведеного коду, з допомогою оператора розширення доступу ‘**::’** відбувається визначення статичного члена даних.

Використання статичного члена d, оголошеного в класі CMyClass з іншого методу (програмного коду):

// використання статичного члена класу

CMyClass MC1; // об'єкт (екземпляр) класу 1

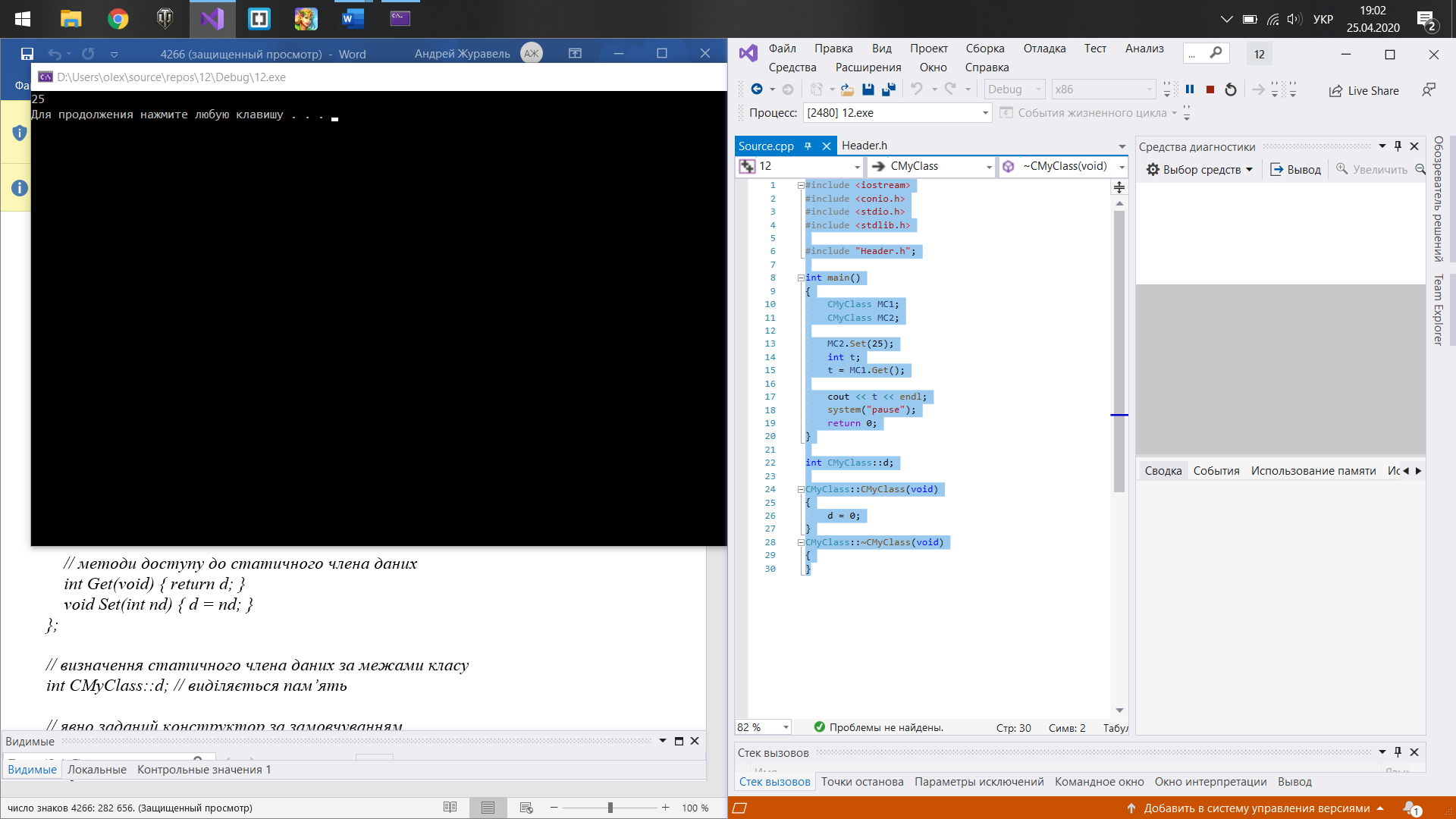
CMyClass MC2; // об'єкт (екземпляр) класу 2

MC2.Set(25);

int t;

t = MC1.Get(); // t = 25 - обидва об'єкти мають доступ до однієї і тієї ділянки пам'яті

Результат роботи програми наведений нижче:



***Порядок виконання роботи:***

Розглянути приклади, наведені у теоретичних відомостях.

***Запитання до самоконтролю:***

1. Як оголошується статичний член даних у класі?

* Статичний член даних в класі оголошується з допомогою ключового слова ***static***.

1. Як визначити статичний член даних?

* За допомогою оператора розширення доступу ‘**::’**

1. Як використати статичний член данних в іншому методі?

* За допомогою методів доступу, що повертають значення члену, чи присвоюють йому нове.

**Лабораторна робота №13**

**Об’єкти класу як члени даних класу**

*Мета роботи*: набуття навичок роботи зоб’єктами класу як членами даних класу у мові С++.

*Теоретичні основи*:

*Приклад 13.1.* Оголошення та використання об’єкту класу, який є членом-даних іншого класу. Як відомо, у класах можуть оголошуватись члени даних класу, які є об’єктами інших класів. У даному прикладі продемонстровано використання двох об’єктів, що є членами даних класу.

У прикладі оголошується 2 класи:

* клас CMyPoint, що визначає точку з координатами x, y;
* клас CMyLine, що визначає лінію, яка містить два об’єкти типу CMyPoint. Ці об’єкти містять координати кінців відрізка, з яких будується лінія.

Клас CMyPoint містить:

* внутрішні члени даних x, y;
* конструктор за замовчуванням CMyPoint();
* методи читання внутрішніх даних класу GetX(), GetY();
* метод SetXY(), що встановлює нові значення x, y.

Клас CMyLine містить:

* внутрішні члени-даних класу типу CMyPoint з іменами mp1, mp2. Ці члени-даних є об’єктами класу CMyPoint. Вони визначають 2 точки, з яких складається лінія;
* конструктор за замовчуванням CMyLine();
* методи читання значення об’єктів mp1, mp2 які мають відповідно імена GetPoint1(), GetPoint2();
* метод SetLine() для встановлення нового значення координат кінців відрізку CMyLine;
* метод Length(), що обчислює довжину відрізка.

Оголошення та реалізація методів класів CMyPoint та CMyLine

// оголошення класу CMyPoint

class CMyPoint

{

int x;

int y;

public:

CMyPoint(void); // конструктор за замовчуванням

// методи класу

int GetX(void)

{

return x;

}

int GetY(void)

{

return y;

}

void SetXY(int nx, int ny)

{

x = nx; y = ny;

}

};

// конструктор за замовчуванням

CMyPoint::CMyPoint(void)

{

x = y = 0;

}

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

// оголошення та реалізація методів класу CMyLine

class CMyLine

{

CMyPoint mp1;

CMyPoint mp2;

public:

// конструктор за замовчуванням

CMyLine(void);

// методи доступу

CMyPoint GetPoint1(void)

{

return mp1;

}

CMyPoint GetPoint2(void)

{

return mp2;

}

void SetLine(CMyPoint p1, CMyPoint p2);

// метод, що повертає довжину лінії

double Length(void);

};

// реалізація методів класу CMyLine

CMyLine::CMyLine(void)

{

mp1.SetXY(0, 0);

mp2.SetXY(1, 1);

}

// встановити нові значення точок

void CMyLine::SetLine(CMyPoint p1, CMyPoint p2)

{

int tx, ty;

tx = p1.GetX();

ty = p1.GetY();

mp1.SetXY(tx, ty);

tx = p2.GetX();

ty = p2.GetY();

mp2.SetXY(tx, ty);

}

// метод, що повертає довжину лінії

double CMyLine::Length(void)

{

int x1, y1, x2, y2;

double len;

x1 = mp1.GetX();

y1 = mp1.GetY();

x2 = mp2.GetX();

y2 = mp2.GetY();

len = System::Math::Sqrt((x1 - x2) \* (x1 - x2) + (y1 - y2) \* (y1 - y2));

return len;

}

Демонстрація використання класу CMyLine

CMyPoint mp1, mp2;

CMyLine ml;

int d;

// виклик методів класу CPoint

// заповнення значень об'єктів mp1, mp2

mp1.SetXY(5, 8);

d = mp1.GetX(); // d = 5

mp2.SetXY(9, 11);

// виклик методів з класу CMyLine

ml.SetLine(mp1, mp2); // встановити нові значення точок

// виклик методу, що повертає першу точку з лінії

mp1 = ml.GetPoint2();

// перевірка

d = mp1.GetX(); // d = 9

d = mp1.GetY(); // d = 11

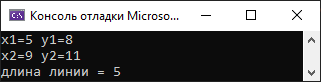
// обчислення відстані відрізка ml

// виклик методу Length

double length;

length = ml.Length(); // length = 5.0

Результат роботи програми наведений нижче:



Якщо в класі A оголошується об’єкт класу B, то при оголошенні об’єкту класу A конструктори обох класів викликаються у такій послідовності:

* конструктор класу B;
* конструктор класу A.

Таку послідовність дій компілятор формує автоматично. Пам’ять для членів-даних класу, які є об’єктами класу, виділяється при виділенні пам’яті для класу, в якому ці об’єкти реалізовані. Тобто, якщо описати об’єкт класу A, в якому є об’єкт класу B, то пам’ять для об’єкту класу A виділяється з врахуванням (включає в себе) пам’яті об’єкту класу B. Якщо взяти клас CMyPoint, то при створенні об’єкту класу CMyLine, спочатку викликається відповідний конструктор класу CMyPoint (виділяється пам’ять), а потім викликається конструктор класу CMyLine.

***Порядок виконання роботи:***

Розглянути приклади, наведені у теоретичних відомостях.

***Запитання до самоконтролю:***

1. В якій послідовності викликаються конструктори класів А та В, якщо в класі A оголошується об’єкт класу B?

**Лабораторна робота №14**

**Покажчики, посилання на об’єкт класу**

*Мета роботи*: набуття навичок роботи зпокажчиками та посиланнями на об’єкти класів у мові С++.

*Теоретичні основи*:

Розглянемо приклад опису покажчику на клас.

*Приклад 14.1.* Нехай у модулі “MyClass.h” дано опис:

// опис класу в модулі "MyClass.h"

class MyClass

{

private:

int x; // поля класу

int y;

public:

// методи класу

void SetXY(int nx, int ny);

int GetX(void);

int GetY(void);

MyClass(void);

};

У модулі “MyClass.cpp” дано реалізацію методів класу :

// реалізація методів класу в модулі "MyClass.cpp"

#include "MyClass.h"

MyClass::MyClass(void)

{

x = 0;

y = 0;

}

void MyClass::SetXY(int nx, int ny)

{

x = nx;

y = ny;

}

int MyClass::GetX(void)

{

return x;

}

int MyClass::GetY(void)

{

return y;

}

Щоб отримати доступ до полів та методів класу через покажчик можна написати такий код:

// Доступ до методів та полів класу через покажчик

MyClass\* p; // покажчик на клас

// виділення пам'яті для покажчика

p = (MyClass\*)malloc(sizeof(MyClass));

// доступ до методів класу через покажчик

p->SetXY(5, 6); // виклик методу SetXY() класу

int x = p->GetX(); // x = 5

int y;

y = p->GetY(); // y = 6

Розглянемо принципи оголошення та використання масиву покажчиків на клас.

*Приклад 14.2.* Нехай задано клас, що описує точку на екрані монітора (піксель).

Клас описаний у двох модулях:

* модуль “MyPixelClass.h” визначає оголошення класу, його методів (функцій) та полів (внутрішніх змінних);
* модуль “MyPixelClass.cpp” визначає реалізацію методів (функцій), оголошених в класі (в модулі “MyPixelClass.h”).

Текст модуля “MyPixelClass.h” (оголошення класу):

*class MyPixelClass*

*{*

*int x;*

*int y;*

*int color;*

*public:*

*MyPixelClass(void);*

*int GetX(void);*

*int GetY(void);*

*int GetColor(void);*

*void SetXYColor(int nx, int ny, int nc);*

*};*

Текст модуля “MyPixelClass.cpp” (реалізація класу):

*#include "MyPixelClass.h"*

*// Реалізація методів класу MyPixelClass*

*MyPixelClass::MyPixelClass(void)*

*{*

*x = y = color = 0;*

*}*

*int MyPixelClass::GetX(void)*

*{*

*return x;*

*}*

*int MyPixelClass::GetY(void)*

*{*

*return y;*

*}*

*int MyPixelClass::GetColor(void)*

*{*

*return color;*

*}*

*void MyPixelClass::SetXYColor(int nx, int ny, int nc)*

*{*

*x = nx;*

*y = ny;*

*color = nc;*

*}*

Демонстрація використання масиву покажчиків на клас MyPixelClass з іншого програмного коду:

*// масив покажчиків на клас*

*MyPixelClass \*mp[5];*

*// виділення пам'яті для об'єктів класу,*

*// на які будуть вказувати покажчики*

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

*mp[i] = new MyPixelClass;*

*// заповнення об'єктів класу довільними значеннями*

*// демонстрація виклику методу SetXYColor()*

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

*mp[i]->SetXYColor(i+5, 2\*i+8, i);*

*// демонстрація інших методів класу*

*int d;*

*d = mp[1]->GetX(); // d = 6*

*d = mp[2]->GetColor(); // d = 2*

*d = mp[3]->GetY(); // d = 2\*3+8 = 14*

Посилання на клас оголошується з допомогою символу &. При оголошенні змінної, що є посиланням, потрібно одразу її ініціалізувати значенням об’єкту, для якого вже виділена пам’ять.

Загальний вигляд оголошення посилання на об’єкт класу має вигляд:

CMyClass & ref = obj;

де

* CMyClass – ім’я класу;
* Ref – ім’я змінної, яка є посиланням на об’єкт obj;
* Obj – ім’я об’єкту, для якого виділена пам’ять.

*Приклад 14.3.* Приклад оголошення посилання на об’єкт класу та його використання. Нехай задано клас CYear, що реалізує рік. Оголошення класу має такий вигляд:

*// клас, що реалізує категорію "Рік"*

*class CYear*

*{*

*int year; // внутрішня змінна*

*public:*

*CYear(void); // конструктор за замовчуванням*

*// конструктор з 1 параметром*

*CYear(int year)*

*{*

*this->year = year;*

*}*

*// методи доступу*

*int Get(void) { return year; }*

*void Set(int year) { this->year = year; }*

*// метод, що визначає чи високосний рік*

*bool IsLeapYear(void);*

*};*

*// конструктор за замовчуванням*

*CYear::CYear(void)*

*{*

*year = 2000; // початкова ініціалізація*

*}*

*// метод, що визначає чи високосний рік*

*bool CYear::IsLeapYear(void)*

*{*

*if ((year % 4) == 0) return true;*

*else return false;*

*}*

Використання класу в іншому методі

*CYear c1; // c1 - об'єкт*

*CYear & c2 = c1; // c1 і c2 вказують на одну й туж ділянку пам'яті*

*int d;*

*// доступ до об'єкту c1*

*c1.Set(2010);*

*d = c1.Get(); // d = 2010*

*// доступ через посилання c2*

*d = c2.Get(); // d = 2010*

*c2.Set(3333);*

*d = c1.Get(); // d = 3333; c1 і c2 вказують на ту саму ділянку пам'яті*

У вищенаведеному коді з допомогою символу & оголошується змінна-посилання c2 на об’єкт класу c1 з допомогою рядка

CYear & c2 = c1;

Для змінної c1 пам’ять вже виділена. Змінні c1 і c2 посилаються на одну й туж ділянку пам’яті (на той самий об’єкт).

*Приклад 14.4.* Оголошення та використання посилання на об’єкт класу, доступ до якого здійснюється з допомогою покажчика. Приклад демонструє використання покажчика та посилання на клас CYear.

*CYear \* pc = new CYear(); // покажчик, виділяється пам'ять*

*CYear & rc = (\*pc); // посилання на пам'ять, виділену для покажчика*

*int d;*

*// доступ до об'єкту за покажчиком*

*pc->Set(2040);*

*d = pc->Get(); // d = 2040*

*d = rc.Get(); // d = 2040*

*rc.Set(2100);*

*d = pc->Get(); // d = 2100*

*Приклад 14.5.* Використання посилання на об’єкт класу, яке є членом даних іншого класу.

Посилання (змінна-посилання) на об’єкт класу може бути членом даних іншого класу. При оголошенні змінної-посилання в класі, ця змінна має бути ініціалізована одразу в спеціально розробленому конструкторі. У цьому випадку для змінної-посилання класу пам’ять виділяється динамічно.

Розглянемо застосування класу Cline (відрізок), в якому оголошуються два посилання на клас CPoint, що описує точку.

У класі CPoint реалізовано такі члени даних та методи:

* внутрішні змінні x, y – координати точки;
* конструктор за замовчуванням;
* методи доступу GetXY(), SetXY().

У класі Cline оголошуються такі члени даних та методи:

* внутрішні змінні-посилання типу CPoint& (посилання на об’єкт класу CPoint) з іменами p1, p2;
* конструктор з двома параметрами, що динамічно ініціалізує посилання p1, p2 при оголошенні об’єкту класу;
* методи доступу GetPoints(), SetPoints().

Оголошення класів CPoint та Cline має вигляд:

// клас, що реалізує точку

class CPoint

{

int x, y;

public:

// конструктор за замовчуванням

CPoint() { x = y = 0; }

// методи доступу

void GetXY(int\* nx, int\* ny)

{

\*nx = x;

\*ny = y;

}

void SetXY(int nx, int ny)

{

x = nx;

y = ny;

}

};

// клас, що реалізує відрізок

class CLine

{

// посилання на об'єкти типу CPoint

CPoint& p1;

CPoint& p2;

public:

// конструктор за замовчуванням

// динамічно ініціалізуються значення посилань p1 та p2

CLine() :p1(\*new CPoint), p2(\*new CPoint)

{

p1.SetXY(0, 0);

p2.SetXY(1, 1);

}

// методи доступу

// повернути координати точок

void GetPoints(CPoint\* pt1, CPoint\* pt2)

{

int x, y;

p1.GetXY(&x, &y); // взяти значення x, y для точки p1

pt1->SetXY(x, y); // встановити x,y в нову точку pt1

p2.GetXY(&x, &y); // взяти x,y

pt2->SetXY(x, y); // записати x,y в pt2

}

// встановити нові значення точок

void SetPoints(CPoint\* pt1, CPoint\* pt2)

{

int x, y;

// p1 => pt1

p1.GetXY(&x, &y);

pt1->SetXY(x, y);

// p2 => pt2

p2.GetXY(&x, &y);

pt2->SetXY(x, y);

}

};

У класі Cline здійснюється динамічна ініціалізація посилань p1 та p2 за допомогою конструктора:

*// динамічна ініціалізація змінних-посилань p1, p2 класу CLine*

*CLine():p1(\* new CPoint), p2(\* new CPoint)*

*{*

*// ...*

*}*

Використання класу Cline в іншому методі:

// посилання на об'єкти як члени-даних класу

CLine cl1; // викликається конструктор, який ініціалізує змінні-посилання p1, p2

// додаткові змінні

CPoint point1, point2;

int x, y;

// перевірка, як конструктор заповнив коодинати точок відрізка

cl1.GetPoints(&point1, &point2);

point1.GetXY(&x, &y); // x = 0; y = 0

point2.GetXY(&x, &y); // x = 1; y = 1

// встановити нові значення

point1.SetXY(3, 8); // записати точку (3; 8)

point2.SetXY(5, 9); // записати точку (5; 9)

cl1.SetPoints(&point1, &point2);

// перевірка

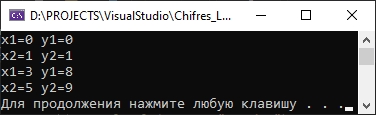
CPoint pp1, pp2;

cl1.GetPoints(&pp1, &pp2);

pp1.GetXY(&x, &y); // x = 3; y = 8

pp2.GetXY(&x, &y); // x = 5; y = 9

Результат роботи програми наведений нижче:



***Порядок виконання роботи:***

1. Розглянути приклади, наведені у теоретичних відомостях.

2. Створити клас «Матриця 2х2». Клас містить:

– властивості:

* покажчик на об‘єкт;
* інші (за завданням);

– методи:

* ініціалізація;
* заповнення даними;
* виведення значень елементів;
* основні методи.

В головній програмі створити декілька об’єктів створеного класу і продемонструвати роботу всіх функцій.

*#pragma once*

*#include <cstdlib>*

*#include <iostream>*

*using namespace std;*

*class matrix*

*{*

*private:*

*int n;*

*int \*\*mas;*

*public:*

*matrix();*

*void enter(int \*\*);*

*void print();*

*matrix operator\*(int);*

*};*

*matrix::matrix()*

*{*

*n = 2;*

*mas = (int \*\*)calloc(n, sizeof(int\*));*

*for (int i = 0; i < n; i++)*

*mas[i] = (int\*)calloc(n, sizeof(int));*

*}*

*void matrix::enter(int \*\*mas\_1)*

*{*

*mas = mas\_1;*

*}*

*void matrix::print()*

*{*

*for (int i = 0; i < n; i++)*

*{*

*for (int j = 0; j < n; j++)*

*{*

*cout << mas[i][j] << " ";*

*}*

*cout << endl;*

*}*

*cin.get();*

*}*

*matrix matrix::operator\*(int number)*

*{*

*matrix tmp;*

*for (int i = 0; i < 2; i++)*

*for (int j = 0; j < 2; j++)*

*{*

*mas[i][j] \*= number;*

*}*

*tmp.mas = mas;*

*return tmp;*

*}*

*#include "pch.h"*

*#include "Matrix.h"*

*#include <iostream>*

*int main()*

*{*

*std::cout << "Hello World!\n";*

*int \*\*mass, number, n;*

*cout << "enter razmer matrix";*

*cin >> n;*

*mass = (int \*\*)calloc(n, sizeof(int\*));*

*for (int i = 0; i < n; i++)*

*mass[i] = (int\*)calloc(n, sizeof(int));*

*for (int i = 0; i < n; i++)*

*for (int j = 0; j < n; j++)*

*{*

*cout << "enter matrix = ";*

*cin >> mass[i][j];*

*}*

*matrix a;*

*a.enter(mass);*

*a.print();*

*cout << "enter number";*

*cin >> number;*

*a = a \* number;*

*a.print();*

*cin.get();*

*return 0;*

*}*

**Лабораторна робота №15**

**Статичні методи класу**

*Мета роботи*: набуття навичок роботи зістатичними методамикласів у мові С++.

*Теоретичні основи*:

У класах можна оголошувати такі види функцій:

* «звичайні» **функції-члени**. Це функції, в оголошенні яких не використовуються додаткові ключові слова (const, volatile, static);
* **статичні** функції. Це функції, які оголошені з ключовим словом static;
* **константні** функції. Це функції, які оголошені з ключовим словом const;
* функції-члени, що повертають непостійний об’єкт. Це функції, які оголошені з ключовим словом volatile;
* функції-члени з константним покажчиком this;
* функції-члени з непостійним покажчиком this;
* вбудовані функції-члени (inline).

*Приклад 15.1.* У даному прикладі оголошено клас CLine, що описує лінію на координатній площині. Лінія складається з координат точок (x1;y1), (x2;y2). Клас містить прості функції члени, які оголошені в розділі public (загальнодоступні). З допомогою цих функцій-членів здійснюється читання/модифікація внутрішніх змінних класу.

*// клас, що описує лінію на координатній площині*

class CLine

{

int x1, y1;

int x2, y2;

public:

CLine(void);

~CLine(void);

// методи класу

void SetLine(int nx1, int ny1, int nx2, int ny2);

int GetX1(void) { return x1; }

int GetY1(void) { return y1; }

int GetX2(void) { return x2; }

int GetY2(void) { return y2; }

};

// Реалізація методів класу

// конструктор за замовчуванням

CLine::CLine(void)

{

x1 = y1 = 0;

x2 = y2 = 1;

}

// деструктор

CLine::~CLine(void)

{

}

// реалізація простої функції-члена класу

void CLine::SetLine(int nx1, int ny1, int nx2, int ny2)

{

x1 = nx1;

y1 = ny1;

x2 = nx2;

y2 = ny2;

}

Використання класу в іншому програмному коді

// оголошення об'єкту класу CLine

CLine CL;

int t;

// виклик функцій-членів класу

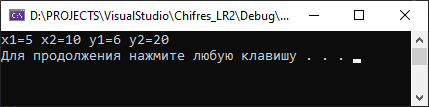
CL.SetLine(5, 6, 10, 20);

t = CL.GetX2(); // t = 10

t = CL.GetY1(); // t = 6

cout << "x1=" << CL.GetX1() << " x2=" << CL.GetX2() << " y1=" << CL.GetY1() << " y2=" << CL.GetY2()<<endl;

Результат роботи програми наведений нижче:



*Приклад 15.2.* Приклад оголошення класу, що містить статичні (static) функції-члени. У даному прикладі оголошено клас CCircle, що описує коло на координатній площині. Клас реалізує такі внутрішні члени даних та методи:

* внутрішні змінні (члени) класу x, y, r (координати центру та радіус кола);
* конструктор класу CCircle(), який визначає одиничне коло (r=1) з центром на початку координат;
* статичну функцію-член класу SetData1(), яка встановлює значення глобальних змінних xx, yy, rr оголошених за межами класу;
* звичайну (нестатичну) функцію-член класу SetData2(), яка змінює значення внутрішніх змінних x, y, r класу;
* звичайні (нестатичні) методи класу, що читають значення внутрішніх змінних класу x, y, r та глобальних змінних xx, yy, rr.

*// клас, що описує коло на координатній площині*

class CCircle

{

// внутрішні змінні класу

int x, y; // координати центру кола

int r; // радіус кола

public:

// конструктор класу

CCircle(void);

// статичний метод класу - тільки для глобальних змінних

static void SetData1(int nx, int ny, int nr);

// звичайні (нестатичні) методи класу

// встановити нові значення внутрішніх змінних класу

void SetData2(int nx, int ny, int nr);

// зчитати значення внутрішніх змінних класу

int GetX(void) { return x; }

int GetY(void) { return y; }

int GetR(void) { return r; }

// зчитати значення глобальних змінних класу

int GetXX(void);

int GetYY(void);

int GetRR(void);

};

// глобальні змінні

int xx, yy, rr; // ці змінні може змінювати статична функція-член класу

// конструктор класу

CCircle::CCircle(void)

{

// заповнення значень внутрішніх (локальних) змінних

x = 0; y = 0; r = 1;

}

// статична функція-член класу

void CCircle::SetData1(int nx, int ny, int nr)

{

// присвоювання значень глобальним змінним,

// що оголошені поза межами класу

xx = nx;

yy = ny;

rr = nr;

// працювати з внутрішніми змінними класу у статичній функції заборонено

// x = nx; // Заборонено! Помилка: "... illegal reference to non-static member 'CCircle::x'

// y = ny; // також заборонено

}

// встановити нові значення звичайних (нестатичних) методів класу

void CCircle::SetData2(int nx, int ny, int nr)

{

x = nx;

y = ny;

r = nr;

}

// зчитати значення глобальних змінних класу

int CCircle::GetXX(void)

{

return xx;

}

int CCircle::GetYY(void)

{

return yy;

}

int CCircle::GetRR(void)

{

return rr;

}

Змінювати значення внутрішніх змінних класу зі статичної функції-члена класу заборонено. Якщо у функції SetData1() спробувати змінити значення внутрішніх змінних класу x, y:

*void CCircle::SetData1(int nx, int ny, int nr)*

*{*

*...*

*x = nx;*

*y = ny;*

*...*

*}*

то компілятор видасть помилку

**... illegal reference to non-static member 'CCircle::x'**

Якщо потрібно змінити значення внутрішніх змінних класу зі статичної функції-члена, тоді цій функції потрібно передати покажчик this класу.

Нижченаведений програмний код демонструє використання статичної функції-члена для класу Circle

*// використання класу CCircle*

CCircle C1;

int tx, ty, tr; // додаткові змінні

// виклик статичної функції-члена

C1.SetData1(2, 3, 4); // змінюються тільки глобальні змінні xx, yy, rr

// прочитати значення внутрішніх змінних класу

tx = C1.GetX(); // tx = 0 - внутрішні змінні x,y,r ініціалізовані конструктором класу

tr = C1.GetR(); // tr = 1

// прочитати значення глобальних змінних, що оголошені за межами класу

tx = C1.GetXX(); // tx = 2

ty = C1.GetYY(); // ty = 3

// виклик звичайної (нестатичної) функції-члена класу

C1.SetData2(2, 3, 4); // змінюються тільки внутрішні змінні x, y, r

tx = C1.GetX(); // tx = 2

tr = C1.GetR(); // tr = 4

cout << "x=" << C1.GetX() << " r=" << C1.GetR() << " xx=" << C1.GetXX() << " yy=" << C1.GetYY() << endl;

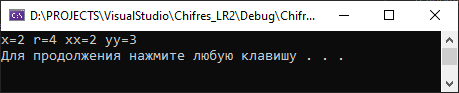
*// інший спосіб виклику статичної функції-члена*

*CCircle::SetData1(7, 8, 9);*

*tx = C1.GetXX(); // tx = 7*

*ty = C1.GetYY(); // ty = 8*

Результат роботи програми наведений нижче:



Основні відмінності статичних функцій-членів від звичайних:

* викликати статичну функцію-член можна без посилання на об’єкт класу, наприклад CMyClass::MyStaticFuntion(…). У цьому випадку використовується оператор розширення області видимості **‘::’**;
* статичні функції-члени не отримують покажчика this класу. Це означає, що вони не можуть отримати доступ до членів даних класу (за винятком членів-даних класу, які оголошені як статичні – з ключовим словом static);
* у C++ статична функція-член класу є глобальною. Тому її доцільно використовувати для зміни значень глобальних змінних, які оголошені за межами класу але мають застосування в класі.

Статичні функції-члени класу доцільно застосовувати у випадках, коли в класі потрібно використовувати глобальні змінні, що є спільними для різних об’єктів (екземплярів) цього класу.

***Порядок виконання роботи:***

1. Розглянути приклади, наведені у теоретичних відомостях.

**Лабораторна робота №16**

**Константні методи класу. Функції-члени, що повертають непостійний об’єкт**

*Мета роботи*: набуття навичок роботи зконстантними методамикласів у мові С++.

*Теоретичні основи*:

Функції, які повертають константний об’єкт називаються **константними функціями**. Якщо такі функції оголошені в класі, то ці функції називаються константними функціями-членами класу. Щоб оголосити функцію, яка повертає константний об’єкт, потрібно перед оголошенням функції розмістити ключове слово ‘const’

*const* returned\_type ***FunName*** (parameters)

*{*

*// function's body*

*// ...*

*}*

де

* returned\_type – тип, що повертається функцією;
* parameters – параметри функції;
* FunName – ім’я функції, що повертає константний об’єкт (константної функції).

Якщо функція оголошена в класі, то приблизне її оголошення буде таким:

*class CMyClass*

*{*

*private:*

*// приватні члени та методи класу*

*// ...*

*public:*

*// загальнодоступні члени та методи класу*

*// ...*

*// оголошення константної функції в класі*

*const* returned\_type***FunName*** *(*parameters*);*

*}*

*// реалізація функції FunName*

*const* returned\_type *CMyClass::FunName(*parameters*)*

*{*

*// тіло функції FunName()*

*// ...*

*}*

*Приклад 16.1.* У прикладі оголошується клас CMyClass, що містить константну функцію GetX(), яка повертає константне значення типу double.

*// клас, що містить оголошення константної функції*

*class CMyClass*

*{*

*double x; // внутрішня змінна*

*public:*

*CMyClass(void); // конструктор*

*void SetX(double nx);*

*const double GetX(void); // константна функція-член класу*

*};*

Використання константної функції в деякому програмному коді:

*// використання константної функції*

*CMyClass MC;*

*MC.SetX(25.08); // встановлення значення x*

*// використання константної функції GetX() класу CMyClass*

*double t1 = MC.GetX(); // t1 = 25.08 - ініціалізація змінної*

*const double t2 = MC.GetX(); // t2 = 25.08 - ініціалізація константи*

Об’єкт (змінна), що оголошений зі специфікатором volatile може бути змінений у програмі неявно без застосування явно заданих команд. Ключове слово volatile інформує компілятор, що значення змінної у програмі може бути змінене неявно (програмою обробки переривань, фоновим процесом, тощо). Знаючи про неявну мінливість змінної у програмі (ключове слово volatile), компілятор сформує код, що буде опитувати значення змінної перед кожним її використанням у програмі. Таким чином, буде сформовано реальне значення змінної (а не те, що було до зміни).

*Приклад 16.2.* Нехай у деякій програмі P реалізована глобальна змінна X, що містить адресу, яка може бути змінена програмою-обробником переривання. Програма-обробник переривання викликається незалежно від виконання даної програми P і відповідним чином змінює значення цієї глобальної змінної X. Якщо у програмі P не вказати ключового слова volatile перед X, то, при зчитуванні значення X у програмі P може відображатись старе значення X. Якщо вказати слово volatile, то компілятор буде оновлювати значення змінної X при кожному звертанні до неї, і, спотворення результату не буде.

Функції-члени, які оголошені зі специфікатором volatile просто повертають непостійний (volatile) об’єкт. Такий об’єкт може бути присвоєний змінній, що оголошена з специфікатором volatile.

Загальний вигляд класу, що містить функцію, яка повертає volatile-значення

*class CMyClass*

*{*

*private:*

*// приватні члени та методи класу*

*// ...*

*public:*

*// загальнодоступні члени та методи класу*

*// ...*

*// оголошення константної функції в класі*

*volatile* returned\_type***FunName****(*parameters*);*

*}*

*// реалізація функції FunName*

*volatile* returned\_type *CMyClass::****FunName****(*parameters*)*

*{*

*// тіло функції FunName()*

*// ...*

*}*

де

* returned\_type – тип, що повертається функцією;
* parameters – параметри функції;
* FunName – ім’я функції, що повертає volatile-об’єкт.

*Приклад 16.3.* У класі CMyVolatileClass оголошено функцію Get(), що повертає volatile-значення типу double.

*// клас, що містить volatile-функцію*

class CMyVolatileClass

{

int d;

public:

CMyVolatileClass(void);

// функції-члени класу

// звичайна функція-член класу

void Set(int nd) { d = nd; }

// функція-член класу, що повертає volatile-значення

volatile int Get(void);

};

volatile int CMyVolatileClass::Get(void) {

cout << this->d<<endl;

return this->d;

}

Демонстрація використання класу в деякому методі чи програмному коді:

*// клас, що містить volatile-функцію*

CMyVolatileClass VC;

VC.Set(33); // виклик звичайної функції

// виклик volatile-функції для звичайної змінної

int t1 = VC.Get(); // t1 = 33

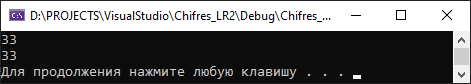
volatile int t2;

// виклик volatile-функції для volatile-змінної

t2 = VC.Get(); // t2 = 33

Як видно з вищенаведеного коду, volatile-функції можна використовувати у поєднанні зі звичайними та volatile-змінними.

Результат роботи програми наведений нижче:



***Порядок виконання роботи:***

1. Розглянути приклади, наведені у теоретичних відомостях.

**Лабораторна робота №17**

**Функції з константним та непостійним покажчиком this**

*Мета роботи*: набуття навичок роботи функціями із константним та непостійним покажчиком this.

*Теоретичні основи*:

*Приклад 17.1.* Оголошення функції у класі (на прикладі функцій MyFun1(), MyFun2()) з константним покажчиком this має такий загальний вигляд:

*class CMyClass*

*{*

*private:*

*// приховані члени даних та методи класу*

*// ...*

*public:*

*// функція оголошена з константним покажчиком this*

return\_type *MyFun1(*parameters*) const; // out-of-line*

return\_type *MyFun2(*parameters*) const // inline реалізація*

*{*

*// тіло функції*

*// ...*

*}*

*}*

*// реалізація out-of-line функції*

*return\_type CMyClass::MyFun1(parameters) const*

*{*

*// тіло функції MyFun1()*

*// ...*

*}*

де

* return\_type – тип, що повертається тією чи іншою функцією;
* parameters – параметри, які отримує функція.

У вищенаведеному фрагменті, у класі CMyClass оголошуються дві функції з константним покажчиком this. Функція MyFun1() реалізована за межами класу (типу out-of-line). Функція MyFun2() реалізована в класі (типу inline). Перед тілом кожної функції стоїть ключове слово const.

У функції з константним покажчиком this перед тілом функції ставиться ключове слово const. Це означає, що тіло функції має обмежені можливості використання.

Слід зауважити, що у цьому випадку ключове слово const не має відношення до значення, що повертається функцією. Воно має значення тільки для покажчика this, що використовується у функції.

Якщо функція оголошена з константним покажчиком this, то в тілі функції заборонено змінювати дані класу. При спробі змінити дані класу, буде виникати помилка компіляції.

*Приклад 17.2.* Відмінність між звичайною функцією та функцією з константним покажчиком this добре видно в реалізації нижченаведеного класу.

Нехай задано клас CRadius, що реалізує радіус деякого об’єкта чи геометричної фігури. У класі реалізовано:

* внутрішній член даних типу double;
* конструктор CRadius();
* звичайну функцію GetRadius(), яка збільшує радіус удвічі і повертає його значення;
* функцію GetRadius2(), оголошену з константним покажчиком this. Дана функція повертає значення внутрішньої змінної radius.

*// клас CRadius*

class CRadius

{

double radius;

public:

CRadius(void); // конструктор класу за замовчуванням

// Звичайна функція-член класу,

// цій функції неявно передається покажчик: CRadius \* const this

double GetRadius(void)

{

radius \*= 2.0; // дозволено

return radius;

}

// Функція-член класу, оголошена з const

// цій функції передається покажчик: const CRadius \* const this

double GetRadius2(void) const

{

//radius = 2.0; // Заборонено! Помилка компіляції

return radius;

}

};

Якщо у функції GetRadius2() спробувати змінити значення внутрішньої змінної radius, то вийде помилка компіляції:

Ключове слово const перед тілом функції в класі означає, що функції заборонено вносити будь-які зміни в члени даних класу.

Використання класу CRadius в деякому програмному коді, наприклад, обробнику події:

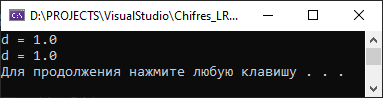
CRadius CR;

double d;

d = CR.GetRadius2(); // d = 1.0 - виклик функції з константним this

d = CR.GetRadius(); // d = 1.0 - виклик звичайної функції-члена

Результат роботи програми наведений нижче:



*Приклад 17.3.* Нехай задано клас CRadius, в якому є функція-член з константним покажчиком this:

*// клас CRadius*

*class CRadius*

*{*

*double radius;*

*public:*

*CRadius(void); // конструктор класу за замовчуванням*

*// Звичайна функція-член класу,*

*// цій функції неявно передається покажчик: CRadius \* const this*

*double GetRadius(void)*

*{*

*radius \*= 2.0; // дозволено*

*return radius;*

*}*

*// Функція-член класу, оголошена з const*

*// цій функції передається покажчик: const CRadius \* const this*

*double GetRadius2(void) const*

*{*

*//radius = 2.0; // Заборонено! Помилка компіляції*

*return radius;*

*}*

*};*

Покажчик this у класі є невидимий, функціям класу він передається неявно. Це означає, що при спробі явного опису покажчика this у класі, компілятор видасть помилку. Звичайній функції-члену класу GetRadius() покажчик this передається в неявному вигляді як:

CRadius \* const this;

Функції-члену GetRadius2() невидимий покажчик this передається з ключовим словом const:

// модифікований покажчик this

const CRadius \* const this;

Запис

const CRadius

означає, що об’єкт типу CRadius є константним об’єктом. І тому, змінювати значення об’єкту у тілі функції GetRadius2() не можна. Отже, не можна змінювати значення внутрішніх членів-даних класу у функції GetRadius2(). Спроба змінити значення внутрішнього члену даних radius у тілі функції GetRadius2() викличе помилку компіляції.

Функції з константним покажчиком this доцільно використовувати у випадках, коли потрібно, щоб функція-член базового класу випадково не перевизначилась у похідних класах. У цьому випадку константний покажчик this служить захистом даних базового класу від випадкової їх зміни у похідних класах.

Це є важливо. У цьому випадку, функція повертає константне значення. Але змінювати внутрішні дані класу у тілі функції можна. Ключове слово const перед іменем функції відноситься до значення, яке повертається функцією. Ключове слово const перед тілом функції відноситься до покажчика this.

Оголошення статичної функції-члена з ключовим словом const не має змісту. При виклику статичної функції-члена, невидимий покажчик this не передається цій функції. При спробі оголошення статичної функції-члена з константним покажчиком this компілятор видасть помилку

**... modifiers not allowed on static members functions**

*Приклад 17.4.* Для того, щоб оголосити функцію-член з непостійним покажчиком this використовується ключове слово volatile. Загальна форма оголошення функції з непостійним покажчиком this в класі має приблизно наступний вигляд:

*class CMyClass*

*{*

*// приховані члени даних та методи класу*

*// ...*

*public:*

*// загальнодоступні члени даних та методи класу*

*// функція-член з непостійним покажчиком this*

return\_type *MyFun1(*parameters*) volatile; // out-of-line*

return\_type *MyFun2(*parameters*) volatile // inline*

*{*

*// тіло функції*

*// ...*

*}*

*}*

*// реалізація out-of-line* функції

return\_type *CMyClass::MyFun1 volatile*

*{*

*// тіло функції MyFun1()*

*// ...*

*}*

де

* return\_type – тип, що повертається тією чи іншою функцією;
* parameters – параметри, які отримує функція.

У вищенаведеному фрагменті, у класі CMyClass оголошуються дві функції з непостійним (volatile) покажчиком this. Функція MyFun1() реалізована за межами класу (типу out-of-line). Функція MyFun2() реалізована у класі (типу inline). Перед тілом кожної функції стоїть ключове слово volatile.

Оголошення функції як volatile доцільно використовувати у випадках, коли потрібно обробляти об’єкт класу різними процесами, до яких можна віднести:

* обробка об’єкту процесором;
* обробка з допомогою фонових прикладних програм;
* обробка з допомогою програм обробки переривань.

Функція-член класу, що оголошена з ключовим словом volatile трактується компілятором у різних випадках по різному. Для таких функцій компілятор відключає деякі види оптимізації в залежності від типу оголошеного об’єкту.

Функції-члени класу, що оголошені з ключовим словом volatile можна використовувати як для автоматичних об’єктів так і для непостійних (volatile) об’єктів.

Для константних об’єктів використання функцій з непостійним покажчиком заборонено. Знову ж таки, все залежить від налаштувань та реалізації компілятора.

*Приклад 17.5.* Нехай дано клас CRadius. У класі реалізовано:

* внутрішній член-даних radius;
* конструктор CRadius();
* звичайну функцію-член класу GetRadius();
* дві фунції-члени класу з непостійним покажчиком this: GetRadius2() та SetRadius().

*// клас CRadius*

class CRadius

{

double radius;

public:

CRadius(void); // конструктор класу за замовчуванням

// Звичайна функція-член класу,

// цій функції неявно передається CRadius \* const this

double GetRadius(void)

{

return radius;

}

// Функція-член класу, оголошена з volatile

// цій функції передається volatile CRadius \* const this

double GetRadius2(void) volatile

{

return radius;

}

// Функція-член класу, оголошена з volatile

void SetRadius(double r) volatile

{

radius = r;

}

};

Використання класу в іншому програмному коді

CRadius CR1; // автоматичний об'єкт

volatile CRadius CR2; // непостійний об'єкт

double d;

// автоматичний об'єкт

CR1.SetRadius(5.5);

d = CR1.GetRadius2(); // d = 5.5

cout << CR1.GetRadius2() << endl;

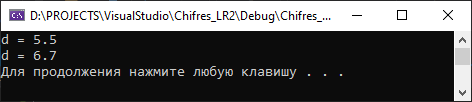
// непостійний об'єкт

CR2.SetRadius(6.7);

d = CR2.GetRadius2(); // d = 6.7

cout << CR2.GetRadius2() << endl;

Результат роботи програми наведений нижче:



Як видно з програмного коду, функції-члени з непостійним (volatile) покажчиком this можна використовувати для автоматичних та непостійних об’єктів. У цьому випадку, компілятор включає різні види оптимізації в залежності від типу об’єкту.

***Порядок виконання роботи:***

1. Розглянути приклади, наведені у теоретичних відомостях.

**Лабораторна робота №18**

**Inline функції**

*Мета роботи*: набуття навичок роботи з вбудованими функціями із константним та непостійним покажчиком this.

*Теоретичні основи*:

У мові C++ в класах можуть використовуватись два типи функцій:

* функції, які підставляються або **вбудовані функції**. Такі функції ще називаються inline-функціями;
* звичайні функції, код яких не підставляється безпосередньо у тіло викликаючої програми.

Inline-функції обробляються так само як макрос. При виклику такої функції з викликаючого коду, тіло функції безпосередньо вставляється у цей код. Інакше кажучи, код inline-функції підставляється у те місце рядка програми, з якого вона викликається. У результаті, виклик inline-функції дає виграш у часі виконання програми (часі обробки функції). Це пов’язано з тим, що зникають накладні витрати на додаткову обробку при передачі (отриманні) параметрів у функцію.

Існує два способи оголошення (створення) inline-функції.

**Спосіб 1**. З використанням модифікатора inline. Загальний вигляд функції, оголошеної з модифікатором inline наступний:

*inline* returned*\_*type***FunName****(*parameters*)*

*{*

*// ...*

*}*

де returned\_type та parameters відповідно тип та параметри, що повертаються функцією.

Не всі компілятори підтримують цей спосіб.

**Спосіб 2.** Реалізація коду функції-члена класу безпосередньо в тілі оголошення класу. У цьому випадку використання модифікатора inline необов’язкове.

Загальний вигляд класу з оголошеними inline-функціями має вигляд:

*class CMyClass*

*{*

*// ...*

*// без модифікатора inline*

returned\_type***MyInlineFun1****(*parameters*)*

*{*

*// ...*

*}*

*// з модифікатором inline*

*inline* returned\_type***MyInlineFun2****(*parameters*)*

*{*

*// ...*

*}*

*// ...*

*}*

де

* returned\_type – тип, що повертається функціями;
* parameters параметри, що отримує функція.

*Приклад 18.1.* У прикладі оголошується клас CMyPoint. Клас містить внутрішні члени даних а також inline-функцію GetY(). Дві інші функції класу є звичайними, тому що реалізація цих функцій винесена за межі класу.

*// клас, що містить inline-функції*

class CMyPoint

{

int x, y;

public:

// конструктор класу

CMyPoint(void);

int GetX(void); // звичайна (не inline) функція класу

int GetY(void) // inline-функція

{

return y; // реалізація у тілі класу

}

// звичайна (не inline) функція

void SetXY(int nx, int ny);

};

// конструктор класу

CMyPoint::CMyPoint(void)

{

x = y = 0;

}

// реалізація не inline-функції GetX()

int CMyPoint::GetX(void)

{

return x;

}

// реалізація не inline функції SetXY()

void CMyPoint::SetXY(int nx, int ny)

{

x = nx;

y = ny;

}

*Використання класу в іншому програмному коді (наприклад, обробнику події)*

CMyPoint MP1; // об'єкт класу, що містить inline-функцію

// виклик звичайної (не inline) функції

MP1.SetXY(25, 30);

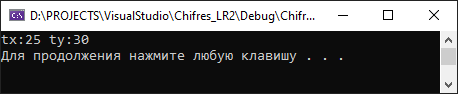
int tx, ty;

tx = MP1.GetX(); // виклик не inline функції

ty = MP1.GetY(); // виклик inline функції

cout << "tx:" << tx << " ty:" << ty << endl;

Результат роботи програми наведений нижче:



Основна перевага використання вбудованих (inline) функцій – це пришвидшення часу виконання програми. Це пов’язано з тим, що при виклику inline-функції не витрачається час на:

* запис аргументів у стек;
* читання аргументів зі стеку при поверненні з функції.

Додавати модифікатор inline в оголошення функції доцільно у випадках, коли виконуються дві основні умови:

* виклик функції відбувається настільки часто, що негативно впливає на швидкість виконання програми або просто уповільнює виконання програми. Наприклад, функція може викликатись багатократно в операторі циклу;
* об’єм коду функції є малим. Функції, що мають великі об’єми програмного коду суттєво збільшують розмір самої програми, що, інколи, теж небажано. Тому в якості вбудованих, рекомендовано використовувати тільки дуже малі функції.

Оголошення inline-функції є запитом а не командою. Тому, компілятор може не виконати запит на генерування коду для оголошеної inline-функції. У цьому випадку, функція може бути використана як звичайна (не inline).

Наприклад:

* у більшості випадків рекурсивні функції не можуть бути використані як inline-функції;
* не можна згенерувати inline функцію, яка містить статичні члени даних.

***Порядок виконання роботи:***

1. Розглянути приклади, наведені у теоретичних відомостях.

**Лабораторна робота №19**

**Класи і масиви**

*Мета роботи*: набуття навичок роботи з класами і масивами.

*Теоретичні основи*:

*Приклад 19.1.* Задано клас CMyPoint, що реалізує точку на координатній площині. Задано клас CPolygon, що реалізує ламану лінію. У класі CPolygon оголошується масив з 10 об’єктів типу CMyPoint. Реалізація класу CMyPoint:

#include <iostream>

#include <cstdlib>

#include <math.h>

using namespace std;

class CMyPoint

{

int x, y;

public:

// конструктор класу

CMyPoint(void)

{

x = y = 0;

}

// методи доступу

int GetX(void) { return x; }

int GetY(void) { return y; }

void SetXY(int nx, int ny)

{

x = nx;

y = ny;

}

};

// клас, що описує ламану лінію

class CPolygon

{

int n; // кількість точок

CMyPoint cp[10]; // масив точок

public:

// конструктор класу

CPolygon(void) { n = 0; }

// методи доступу

CMyPoint GetPoint(int index)

{

return cp[index];

}

// встановлює нове значення заданої точки

void SetPoint(int index, CMyPoint& pt)

{

if (index >= n) return;

cp[index] = pt;

}

// додає точку до ламаної

void AddPoint(CMyPoint pt)

{

if (n < 10) n++;

else return;

cp[n - 1] = pt;

}

// визначає довжину ламаної

double Length(void)

{

double len, t;

int x1, y1, x2, y2;

len = 0;

for (int i = 0; i < n - 1; i++)

{

x1 = cp[i].GetX();

y1 = cp[i].GetY();

x2 = cp[i + 1].GetX();

y2 = cp[i + 1].GetY();

t = sqrt((x1 - x2) \* (x1 - x2) + (y1 - y2) \* (y1 - y2));

len += t;

}

return len;

}

};

int main()

{

setlocale(LC\_ALL, "Russian");

// демонстрація використання класу CPolygon

CPolygon p1;

CMyPoint pt1;

// сформувати ламану з координатами (1; 2), (3; 4), (5;6)

pt1.SetXY(1, 2);

p1.AddPoint(pt1);

pt1.SetXY(3, 4);

p1.AddPoint(pt1);

pt1.SetXY(5, 6);

p1.AddPoint(pt1);

// перевірка

int x, y;

CMyPoint pt2;

pt2 = p1.GetPoint(0);

x = pt2.GetX(); // x = 1

y = pt2.GetY(); // y = 2

cout << "x1:" << x << " y1:" << y << endl;

pt2 = p1.GetPoint(1);

x = pt2.GetX(); // x = 3

y = pt2.GetY(); // y = 4

cout << "x2:" << x << " y2:" << y << endl;

pt2 = p1.GetPoint(2);

x = pt2.GetX();

y = pt2.GetY();

cout << "x3:" << x << " y3:" << y << endl;

// довжина ламаної

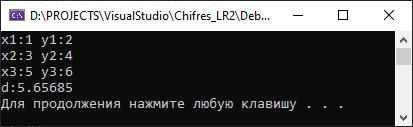
double d = p1.Length(); // d = 5.6568...

cout << "d:" << d<< endl;

system("pause");

return 0; }

Результат роботи програми наведений нижче:



*Приклад 19.2.* Реалізація класу CMyPoint

#include <iostream>

#include <cstdlib>

#include <math.h>

using namespace std;

class CMyPoint

{

int x, y;

public:

// конструктор

CMyPoint(void)

{

x = y = 0;

}

// методи доступу

int GetX(void) { return x; }

int GetY(void) { return y; }

void SetXY(int x, int y)

{

this->x = x;

this->y = y;

}

};

// клас, що описує ламану лінію

class CPolygon

{

int n;

CMyPoint\* mp[10]; // масив покажчиків на CMyPoint

public:

// конструктор

CPolygon(void)

{

n = 0;

// виділення пам'яті - лише один раз

for (int i = 0; i < 10; i++)

{

mp[i] = new CMyPoint;

}

}

// методи доступу

CMyPoint GetPoint(int index)

{

return \*mp[index];

}

void SetPoint(int index, CMyPoint& p)

{

int x, y;

x = p.GetX();

y = p.GetY();

mp[index]->SetXY(x, y);

}

// додати нову точку

void AddPoint(int x, int y)

{

mp[n++]->SetXY(x, y);

}

// обчислення довжини ламаної

double Length(void)

{

double len, t;

int x1, y1, x2, y2;

len = 0;

for (int i = 0; i < n - 1; i++)

{

x1 = mp[i]->GetX();

y1 = mp[i]->GetY();

x2 = mp[i + 1]->GetX();

y2 = mp[i + 1]->GetY();

t = sqrt((x1 - x2) \* (x1 - x2) + (y1 - y2) \* (y1 - y2));

len = len + t;

}

return len;

}

};

int main()

{

setlocale(LC\_ALL, "Russian");

// демонстрація використання масиву unmanaged-покажчиків на об'єкт класу

CPolygon cp; // об'єкт класу

// сформувати ламану з трьох точок

cp.AddPoint(1, 2);

cp.AddPoint(3, 4);

cp.AddPoint(5, 6);

// перевірка

int x, y;

CMyPoint mp;

mp = cp.GetPoint(0);

x = mp.GetX(); // x = 1

y = mp.GetY(); // y = 2

cout << "x1:" << x << " y1:" << y << endl;

mp = cp.GetPoint(1);

x = mp.GetX();

y = mp.GetY();

cout << "x2:" << x << " y2:" << y << endl;

mp = cp.GetPoint(2);

x = mp.GetX();

y = mp.GetY();

cout << "x3:" << x << " y3:" << y << endl;

// визначення довжини ламаної

double d;

d = cp.Length(); // d = 5.6568...

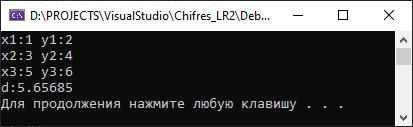
cout << "d:" << d<< endl;

system("pause");

return 0;

}

Результат роботи програми наведений нижче:



Можливі випадки, коли потрібно об’єднати змінні, які є членами об’єкта класу в масив. У нижченаведеному прикладі ініціалізується масив, що в якості ініціалізаторів використовує члени-даних об’єкту. На момент оголошення члени-даних об’єкту мають мати якісь змістовні значення.

Приклад визначає клас Radius, в якому оголошуються:

* внутрішня private змінна radius;
* внутрішній private метод CalcValues(), що заповнює значеннями public змінні length, area, volume;
* загальнодоступні (public) внутрішні змінні класу length, area, volume, які містять відповідні значення довжини кола, площі круга та об’єму кулі для радіуса r;
* конструктори класу;
* методи доступу GetR(), SetR().

Лістинг класу наступний:

*// клас, що реалізує величину радіуса деякої геометричної фігури*

*class Radius*

*{*

*double radius; // змінна, що визначає радіус*

*// внутрішній метод, що формує значення змінних length, area, volume*

*void CalcValues(void)*

*{*

*const double pi = 3.141592653589;*

*length = 2 \* pi \* radius;*

*area = pi \* radius \* radius;*

*volume = 4.0/3.0 \* pi \* radius \* radius \* radius;*

*}*

*public:*

*double length; // довжина кола*

*double area; // площа круга*

*double volume; // об'єм кулі*

*// конструктори*

*Radius()*

*{*

*radius = 1;*

*CalcValues(); // заповнити змінні length, area, volume*

*}*

*Radius(double radius)*

*{*

*this->radius = radius;*

*CalcValues(); // заповнити змінні length, area, volume*

*}*

*// методи доступу*

*int GetR(void) { return radius; }*

*void SetR(double radius)*

*{*

*this->radius = radius;*

*CalcValues(); // заповнити змінні length, area, volume*

*}*

*};*

Використання класу у функції main() або іншій функції:

*Radius r(3); // об'єкт r класу Radius*

*// ініціалізація масиву V значеннями, що є членами даних об'єкту r*

*double V[] = {*

*r.length,*

*r.area,*

*r.volume*

*};*

*double x;*

*x = V[0]; // x = 18.8496 - довжина кола*

*x = V[1]; // x = 28.2743 - площа круга*

*x = V[2]; // x = 113.097 - об'єм кулі*

*Приклад 19.3.* Елементи масиву можуть бути ініціалізовані значеннями, які повертають методи деякого класу. У нижченаведеному прикладі демонструється ініціалізація масиву V значеннями, які повертають public методи класу Radius: Length(), Area(), Volume().

Оголошення класу має вигляд:

*// клас, що реалізує величину радіуса деякої геометричної фігури*

*class Radius*

*{*

*double radius; // змінна, що визначає радіус*

*public:*

*double Length()*

*{*

*return (2 \* 3.1415 \* radius);*

*}*

*double Area()*

*{*

*return (3.1415 \* radius \* radius);*

*}*

*double Volume()*

*{*

*return (4.0/3.0 \* 3.1415 \* radius \* radius \* radius);*

*}*

*// конструктори*

*Radius()*

*{*

*radius = 1;*

*}*

*Radius(double radius)*

*{*

*this->radius = radius;*

*}*

*// методи доступу*

*int GetR(void) { return radius; }*

*void SetR(double radius)*

*{*

*this->radius = radius;*

*}*

*};*

Використання класу в деякому методі

*Radius r(3); // об'єкт r класу Radius*

*// ініціалізація масиву V значеннями, які повертають методи об'єкту класу r*

*double V[] = {*

*r.Length(),*

*r.Area(),*

*r.Volume()*

*};*

*double x;*

*x = V[0]; // x = 18.8496 - довжина кола*

*x = V[1]; // x = 28.2743 - площа круга*

*x = V[2]; // x = 113.097 - об'єм кулі*

*Приклад 19.4.* Щоб оголосити покажчик на статичний член даних класу потрібно просто описати покажчик на тип даних, який має статичний член даних. Потім, у деякому програмному коді, цьому покажчику потрібно присвоїти адресу статичного члена даних об’єкту цього класу.

Послідовність кроків наступна:

* оголосити клас зі статичним членом деякого типу;
* описати статичний член даних класу за межами класу та ініціалізувати його при необхідності;
* у деякому методі або за межами усіх методів (глобальний покажчик) описати покажчик на тип, що має статичний член даних класу;
* описати об’єкт класу зі статичним членом класу;
* присвоїти покажчику адресу статичного члена даних об’єкту класу.

У прикладі оголошується клас CPi зі статичним членом даних Pi. Також оголошується декілька статичних членів, які не є членами класу. У функції \_tmain() демонструється використання покажчиків на оголошеністатичні члени даних.

*#include "stdafx.h"*

*#include <iostream>*

*using namespace std;*

*// оголошення класу, що містить статичний член даних*

*class CPi*

*{*

*public:*

*static double Pi; // статичний член даних - константа*

*};*

*// оголошення статичної змінної за межами класу з одночасною ініціалізацією*

*static double Exp = 2.7182818284; // експонента*

*// ініціалізація статичного члена класу CPi*

*double CPi::Pi = 3.141592653589793;*

*// неініціалізовані статичні змінні, значення яких = 0 (за замовчуванням)*

*static double ZeroD;*

*static int ZeroI;*

*static char ZeroC;*

*int \* pZ; // зовнішній покажчик на int*

*int \_tmain(int argc, \_TCHAR\* argv[])*

*{*

*double d;*

*CPi obj; // об'єкт класу*

*double \* pPi; // покажчик на double*

*double \* pExp; // покажчик на double*

*// покажчик вказує на статичний член даних об'єкту obj класу CPi*

*pPi = &obj.Pi;*

*pExp = &::Exp; // вказує на статичний член даних Exp - інший вид доступу через ::*

*// перевірка*

*d = obj.Pi; // d = 3.14159*

*d = Exp; // d = 2.71828*

*d = ::Exp; // d = 2.71828 - так теж можна*

*d = \*pPi; // d = 3.14159*

*// зовнішній покажчик на int*

*pZ = &ZeroI;*

*d = \*pZ; // d = 0*

*\*pZ = 5;*

*d = ZeroI; // d = 5*

*return 0;*

*}*

*Приклад 19.5.* Задано 3 класи з іменами CCounter1, CCounter2, CCounter3. У кожному класі реалізовано по одному статичному члену даних відповідно з іменами counter1, counter2, counter3.

Оскільки, в класах є статичні члени даних, то їх відповідним чином можна ініціалізувати. Також у прикладі оголошується глобальний масив з іменем arCount1[], який ініціалізований статичними членами даних класів CCounter1, CCounter2, CCounter3.

Лістинг оголошення класів та ініціалізації статичних членів наступний:

*// оголошується 3 класи зі статичними змінними*

*// клас, що містить статичну змінну count*

*class CCount1*

*{*

*public:*

*static int count1; // статичний член даних, ключове слово static*

*};*

*class CCount2*

*{*

*public:*

*static int count2; // статичний член даних, ключове слово static*

*};*

*class CCount3*

*{*

*public:*

*static int count3; // статичний член даних, ключове слово static*

*};*

*// оголошення ініціалізація статичних членів даних класів*

*// тут ключового слова static не потрібно*

*int CCount1::count1; // за замовчуванням значення статичного члена = 0*

*int CCount2::count2 = 5;*

*int CCount3::count3 = 100;*

*// оголошення масиву, що ініціалізований статичними членами даних*

*int arCount1[] =*

*{*

*CCount1::count1,*

*CCount2::count2,*

*CCount3::count3*

*};*

Нижче продемонстровано використання глобального масиву arCount1[] та локального масиву arCount2[], оголошеного безпосередньо у функції \_tmain() для консольних додатків:

*int \_tmain(int argc, \_TCHAR\* argv[])*

*{*

*// використання масиву arCount1*

*int d;*

*d = arCount1[0]; // d = 0*

*d = arCount1[1]; // d = 5*

*d = arCount1[2]; // d = 100*

*// змінити значення масиву arCount1[2] так,*

*// щоб воно містило значення статичного члена класу CCount1*

*arCount1[2] = CCount1::count1;*

*d = arCount1[2]; // d = 5*

*// Оголошення локального масиву arCount2*

*int arCount2[5];*

*// заповнення масиву arCount2 деякими значеннями статичних членів*

*arCount2[0] = CCount1::count1;*

*arCount2[1] = CCount2::count2;*

*d = arCount2[1]; // d = 5*

*return 0;*

*}*

*Приклад 19.6.* У даному прикладі за зразком попереднього прикладу оголошується 3 класи з іменами CCounter1, CCounter2, CCounter3. У цих класах оголошується по одному статичному члену даних відповідно з іменем counter1, counter2, counter3, які окремо ініціалізуються початковими значеннями. Приклад демонструє оголошення та використання масивів покажчиків arPCount1 та arPCount2 на статичні члени даних, які є членами даних деякого класу.

Лістинг оголошення класів, статичних членів даних та масиву покажчиків має наступний вигляд:

*// оголошується 3 класи зі статичними змінними*

*// клас, що містить статичну змінну count*

#include <iostream>

#include <cstdlib>

#include <math.h>

using namespace std;

class CCount1

{

public:

static int count1; // статичний член даних

};

class CCount2

{

public:

static int count2; // статичний член даних

};

class CCount3

{

public:

static int count3; // статичний член даних

};

// оголошення ініціалізація статичних членів даних класів

int CCount1::count1; // за замовчуванням значення статичного члена = 0

int CCount2::count2 = 5;

int CCount3::count3 = 100;

int main()

{

setlocale(LC\_ALL, "Russian");

// оголошення глобального масиву покажчиків на статичні члени даних

int\* arPCount1[] =

{

&CCount1::count1,

&CCount2::count2,

&CCount3::count3

};

// використання масиву покажчиків arPCount1

int d = \*arPCount1[0]; // d = 0

cout << "d:" << d << endl;

d = \*arPCount1[1]; // d = 5

cout << "d:" << d << endl;

// змінити значення масиву покажчиків arPCount1[2] так,

// щоб він вказував на статичний член класу CCount2

arPCount1[2] = &CCount2::count2;

d = \*arPCount1[2]; // d = 100

cout << "d:" << d << endl;

// Оголошення локального масиву arPCount2

int\* arPCount2[5];

// заповнення масиву arPCount2 деякими значеннями та їх використання

arPCount2[0] = &CCount1::count1;

arPCount2[1] = &CCount3::count3;

d = \*arPCount2[1]; // d = 100

cout << "d:" << d << endl;

//cout << "x3:" << x << " y3:" << y << endl;

// визначення довжини ламаної

//double d;

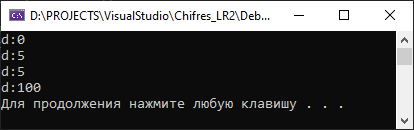
//d = cp.Length(); // d = 5.6568...

system("pause");

return 0;

}

Результат роботи програми наведений нижче:



Якщо статичний член не ініціалізований, то встановлюється значення за замовчуванням:

* для числових типів (int, float, double …) значення неініціалізованого статичного члена рівне 0
* для типу char значення неініціалізованого статичного члена рівне символу з кодом 0.
* для типу bool значення рівне false, що також відповідає значенню 0.

***Порядок виконання роботи:***

1. Розглянути приклади, наведені у теоретичних відомостях.

**Лабораторна робота №20**

**Дружні функції**

*Мета роботи*: набуття навичок роботи з класами і масивами.

*Теоретичні основи*:

Бувають випадки, коли для заданого класу потрібно оголосити інший клас або функцію, які повинні мати необмежений доступ до внутрішніх змінних та методів класу. Така необхідність виникає з суті задачі, що розв’язується.

Якщо клас A оголошується “дружнім” до класу B, то об’єкти класу A мають доступ до усіх членів даних і методів класу B. Якщо функція оголошується “дружньою” до деякого класу, то у цій функції також є необмежений доступ до членів даних та методів цього класу.

*Приклад 20.1.* Щоб оголосити “дружній” клас до даного класу, використовується ключове слово friend. Загальна форма оголошення “дружнього” класу до даного має вигляд:

*class CClass*

*{*

*// ...*

*friend class CFriendClass;*

*// ...*

*};*

*class CFriendClass*

*{*

*// ...*

*};*

де

* CClass – клас, в якому оголошується “дружній” клас CFriendClass. Усі змінні (навіть і private) та методи цього класу є доступними для об’єктів класу CFriendClass;
* CFriendClass – клас, який є “дружнім” до класу CClass. Оголошення “дружнього” класу CFriendClass до класу CClass може бути в будь-якому місці тіла класу – у межах оголошення класу (між фігурними дужками { }).

Оголошення “дружньої” функції до класу починається з ключового слова friend. Загальна форма оголошення “дружньої” функції до класу має вигляд:

friend type **FunName**(parameters);

де

* FunName – ім’я “дружньої” функції;
* Type – тип, що повертається функцією FunName();
* Parameter – параметри “дружньої” функції. Щоб отримати об’єкт потрібного класу у функції FunName() доцільно передати у цю функцію посилання (або покажчик) на об’єкт цього класу.

*Приклад 20.2.* Якщо потрібно оголосити “дружню” функцію у деякому класі, то загальний вигляд такого оголошення наступний:

*class CClass*

*{*

*// ...*

*friend* type***FunName****(*parameters*);*

*// ...*

*};*

де

* FunName – ім’я “дружньої” функції;
* Type – тип, що повертається функцією FunName();
* parameters– параметри “дружньої” функції.

Оголошувати “дружній” клас або функцію до заданого класу можна у будь-якому місці чи розділі класу в межах його оголошення (між фігурними дужками { } ).

*Приклад 20.3.* Щоб отримати об’єкт потрібного класу у функції доцільно передати у цю функцію посилання (або покажчик) на об’єкт цього класу.

Нехай задано клас з іменем CMyClass. Потрібно оголосити “дружню” функцію до цього класу, яка має ім’я FriendFun(). Функція повертає параметр типу int.

Оголошення класу CMyClass та “дружньої” функції у класі має вигляд:

*// оголошення класу CMyClass, в якому є "дружня" функція FriendFun()*

*class CMyClass*

*{*

*// тіло класу*

*// ...*

*friend int FriendFun(CMyClass &);*

*// ...*

*};*

Реалізація “дружньої” функції FriendFun():

*int FriendFun(CMyClass & mc)*

*{*

*// використання об'єкту класу mc для доступу до членів класу CMyClass*

*// ...*

*};*

*Приклад 20.4.* Нехай задано клас Number, що містить цілочисельну величину. Також задано клас RangeNum, який містить величину Number але в межах заданого діапазону.

Щоб з класу RangeNum можна було мати доступ до приватної змінну num класу Number, клас RangeNum оголошується “дружнім” до класу Number.

*// клас, що реалізує ціле число*

*class Number*

*{*

*// оголошення дружнього класу RangeNum до класу Number*

*friend class RangeNum;*

*int num;*

*public:*

*// конструктори*

*Number() { num = 0; }*

*Number(int num) { this->num = num; }*

*};*

*// оголошення класу RangeNum, який тримає число Number в заданих межах*

*class RangeNum*

*{*

*Number num; // об'єкт класу Number - просте ціле число*

*int min; // нижня межа числа num*

*int max; // верхня межа числа num*

*public:*

*// конструктор класу*

*RangeNum()*

*{*

*// доступ до private-члена класу Number, тому що RangeNum є дружнім до Number*

*num.num = 0;*

*// встановлення діапазону 0..99 за замовчуванням*

*min = 0;*

*max = 99;*

*}*

*// методи доступу*

*int GetNum(void)*

*{*

*return num.num; // доступ до приватного члена з "дружнього" класу Range*

*}*

*void SetNum(int nnum)*

*{*

*num.num = nnum; // доступ до приватного члена з "дружнього" класу Range*

*if (num.num>max) num.num = max-1;*

*if (num.num<min) num.num = min;*

*}*

*// встановлення діапазону для num в заданих межах*

*void SetRange(int min, int max)*

*{*

*this->min = min;*

*this->max = max;*

*if (num.num>max) num.num = max-1; // знову доступ через дружній клас*

*if (num.num<min) num.num = min;*

*}*

*};*

Використання класів у іншому програмному коді

*int \_tmain(int argc, \_TCHAR\* argv[])*

*{*

*// об'єкт класу Range*

*RangeNum r;*

*int d;*

*d = r.GetNum(); // d = 0*

*r.SetRange(100, 200);*

*r.SetNum(101);*

*d = r.GetNum(); // d = 101*

*r.SetRange(10, 20); // корегується значення num*

*d = r.GetNum(); // d = 19*

*r.SetNum(-10);*

*d = r.GetNum(); // d = 10*

*return 0;*

*}*

Якщо у класі Number в оголошенні “дружнього” класу RangeNum

friend class RangeNum;

забрати ключове слово friend, то у конструкторі та усіх методах класу при доступі до num.num компілятор видасть помилку:

**Number::num: cannot access private member declared in class 'Number'**

*Приклад 20.5.* У прикладі оголошується клас Radius, що містить величину радіуса деякої геометричної фігури. У класі оголошуються:

* одна прихована (private) змінна radius;
* методи Get() та Set() для доступу до змінної radius;
* дві зовнішні “дружні” функції GetLength() та GetArea();
* один “дружній” клас Volume. У класі Volume оголошується зовнішній (public) метод GetVolume(), який повертає об’єм кулі заданого радіуса.

*#include "stdafx.h"*

*#include <iostream>*

*using namespace std;*

*// клас, що реалізує величину радіуса геометричної фігури*

*class Radius*

*{*

*private:*

*double radius; // прихована змінна radius*

*// оголошення "дружніх" функцій - в будь-якому розділі класу Radius*

*friend double GetLength(Radius &);*

*friend double GetArea(Radius &);*

*// оголошення "дружнього" класу*

*friend class Volume;*

*public:*

*// методи доступу до radius*

*double Get(void) { return radius; }*

*void Set(double nradius) { radius = nradius; }*

*};*

*// "дружні" функції до класу Radius*

*// довжина кола*

*double GetLength(Radius & r)*

*{*

*// доступ до private-члена radius класу з "дружньої" функції GetLength()*

*return (double)(2 \* r.radius \* 3.1415);*

*}*

*// площа круга*

*double GetArea(Radius & r)*

*{*

*// доступ до private-члена radius класу з "дружньої" функції GetArea()*

*return (double)(r.radius \* r.radius \* 3.1415);*

*}*

*// оголошення "дружнього" класу*

*class Volume*

*{*

*public:*

*// клас містить тільки одну функцію Volume*

*double GetVolume(Radius \* r) // функція отримує покажчик на Radius*

*{*

*// доступ до private-члена класу Radius з "дружнього" класу Volume*

*return (double)(4.0 / 3.0 \* 3.1415 \* r->radius \* r->radius \* r->radius);*

*}*

*};*

*int \_tmain(int argc, \_TCHAR\* argv[])*

*{*

*// об'єкт класу Radius*

*Radius r;*

*Volume v;*

*double len, area, vol;*

*r.Set(3);*

*// виклик зовнішньої "дружньої" функції GetLength()*

*len = GetLength(r); // передача об'єкту класу Radius за посиланням*

*// виклик зовнішньої "дружньої" функції GetArea()*

*area = ::GetArea(r); // передача за посиланням*

*// виклик функції "дружнього" класу v*

*vol = v.GetVolume(&r); // передача за покажчиком*

*cout << "Length = " << len << endl; // Length = 9.4245*

*cout << "Area = " << area << endl; // Area = 28.2735*

*cout << "Volume = " << vol << endl; // Volume = 113.094*

*return 0;*

*}*

Як видно з вищенаведеного коду, “дружні” функції отримують вхідним параметром посилання або покажчик на об’єкт класу, до членів даних та методів якого вони мають необмежений доступ.

***Порядок виконання роботи:***

1. Розглянути приклади, наведені у теоретичних відомостях.

**Лабораторна робота №21**

**Перевантаження операторів**

*Мета роботи*: набуття навичок роботи з перевантаженими операторами.

*Теоретичні основи*:

Мова C++ має широкі можливості для перевантаження більшості операторів. Перевантаження оператору означає використання оператору для оперування об’єктами класів. Перевантаження оператору – спосіб оголошення та реалізації оператору таким чином, що він обробляє об’єкти конкретних класів або виконує деякі інші дії. При перевантаженні оператору в класі викликається відповідна операторна функція (*operator function*), яка виконує дії, що стосуються даного класу. Якщо оператор “перевантажено”, то його можна використовувати в інших методах у звичному для нього вигляді. Наприклад, команди поелементного сумування двох масивів a1 та a2.

*a1.add(a2);*

*a3 = add(a1, a2);*

краще викликати більш природнім способом:

*a1 = a1 + a2;*

*a3 = a1 + a2;*

У даному прикладі оператор **‘+’** вважається перевантаженим.

Для заданого класу операторну функцію в класі можна реалізувати:

* всередині класу. У цьому випадку, операторна функція є методом класу;
* за межами класу. У цьому випадку операторна функція оголошується за межами класу як “дружня” (з ключовим словом *friend*). Більш детально про реалізацію “дружніх” функцій за межами класу описується.

Загальна форма операторної функції, яка реалізована в класі, має такий вигляд:

*return\_type ClassName::operator#(arguments\_list)*

*{*

*// деякі операції*

*// ...*

*}*

де

* return\_type – тип значення, що повертається операторною функцією;
* ClassName – ім’я класу, в якому реалізована операторна функція;
* operator# – ключове слово, що визначає операторну функцію в класі. Символ # замінюється оператором мови C++, який перевантажується. Наприклад, якщо перевантажується оператор **+**, то потрібно вказати operator+;
* argument\_list – список параметрів, які отримує операторна функція. Якщо перевантажується бінарний оператор, то argument\_list містить один аргумент. Якщо перевантажується унарний оператор, то список аргументів пустий.

*Приклад 21.1.* Оголошується клас Point, що реалізує точку на координатній площині. У класі реалізовано:

* дві внутрішні змінні x, y, що є координатами точки;
* два конструктори класу;
* методи доступу до внутрішніх змінних класу GetX(), GetY(), SetX(), SetY();
* дві операторні функції operator+() та operator-(). Операторна функція operator+() перевантажує бінарний оператор **‘+’**. Операторна функція operator-() перевантажує унарний оператор **‘-‘**.

*// Клас, що реалізує точку на координатній площині*

*// клас містить дві операторні функції*

*class Point*

*{*

*private:*

*int x, y; // координати точки*

*public:*

*// конструктори класу*

*Point()*

*{*

*x = y = 0;*

*}*

*Point(int nx, int ny)*

*{*

*x = nx;*

*y = ny;*

*}*

*// методи доступу до членів класу*

*int GetX(void) { return x; }*

*int GetY(void) { return y; }*

*void SetX(int nx) { x = nx; }*

*void SetY(int ny) { y = ny; }*

*// перевантажений бінарний оператор '+'*

*Point operator+(Point pt)*

*{*

*// p - тимчасовий об'єкт, який створюється з допомогою конструктора без параметрів*

*Point p;*

*p.x = x + pt.x;*

*p.y = y + pt.y;*

*return p;*

*}*

*// перевантажений унарний оператор '-'*

*Point operator-(void)*

*{*

*Point p;*

*p.x = -x;*

*p.y = -y;*

*return p;*

*}*

*};*

Як видно з вищенаведеного коду, операторна функція operator+() отримує один параметр. Це означає, що ця функція реалізує бінарний оператор **‘+’**. Цей параметр відповідає операнду, що розміщується у правій частині бінарного оператора **‘+’**. Операнд, що розміщується в лівій частині оператора **‘+’** передається операторній функції неявно з допомогою покажчика this цього класу.

Виклик операторної функції здійснює об’єкт, який розміщується в лівій частині оператора присвоювання. Демонстрація використання перевантажених операторів класу Point в іншому методі:

*// оголошення змінних - об'єктів класу CPoint*

*Point P1(3,4);*

*Point P2(5,7);*

*Point P3;*

*int x, y; // додаткові змінні*

*// 1. Використання перевантаженого бінарного оператора '+'*

*P3 = P1 + P2; // об'єкт P1 викликає операторну функцію*

*// перевірка*

*x = P3.GetX(); // x = 8*

*y = P3.GetY(); // y = 11*

*// 2. Використання перевантаженого унарного оператора '-'*

*P3 = -P2;*

*x = P3.GetX(); // x = -5*

*y = P3.GetY(); // y = -7*

У вищенаведеному коді в операції додавання **‘+’** об’єкт P1 викликає операторну функцію. Тобто фрагмент рядка

*P1 + P2*

замінюється викликом

*P1.operator+(P2)*

Реалізувати операторну функцію operator+() в класі можна й по іншому

*Point operator+(Point pt)*

*{*

*// виклик конструктора з двома параметрами*

*return Point(x+pt.x, y+pt.y); // створюється тимчасовий об'єкт, який потім копіюється*

*}*

У вищенаведеній функції в операторі return створюється тимчасовий об’єкт шляхом виклику конструктора з двома параметрами, який реалізований в класі. Якщо (у даному випадку) з тіла класу забрати конструктор з двома параметрами

*// конструктор з двома параметрами*

*Point(int nx, int ny)*

*{*

*// ...*

*}*

то вищенаведений варіант функції operator+() працювати не буде, тому що для створення об’єкту типу Point ця функція використовує конструктор з двома параметрами. У цьому випадку компілятор видасть повідомлення

Point::Point : no overloaded function takes 2 arguments

що означає: немає методу (конструктора) Point::Point() який приймає 2 аргументи.

*Приклад 21.2.* У прикладі реалізується операторна функція operator\*(), яка множить поелементно значення внутрішніх масивів об’єктів класу ArrayFloat. Якщо розмір масивів неоднаковий, то перемножується тільки та кількість елементів, яка є мінімальною між двома розмірами масивів.

*// масив дійсних чисел*

*class ArrayFloat*

*{*

*private:*

*float A[10]; // масив дійсних чисел, фіксований розмір масиву*

*int size;*

*public:*

*// конструктори*

*ArrayFloat()*

*{*

*size = 0;*

*}*

*ArrayFloat(int nsize, float nA[])*

*{*

*size = nsize;*

*for (int i=0; i<nsize; i++)*

*A[i] = nA[i];*

*}*

*// методи доступу*

*float GetAi(int i)*

*{*

*if ((i>=0) && (i<=size-1))*

*return A[i];*

*else*

*return 0;*

*}*

*void SetAi(int i, float value)*

*{*

*if ((i>=0) && (i<=size-1))*

*A[i] = value;*

*}*

*// перевантажений оператор '\*'*

*ArrayFloat operator\*(ArrayFloat AF)*

*{*

*ArrayFloat tmp;*

*int n;*

*if (size<AF.size)*

*n = AF.size;*

*else*

*n = size;*

*for (int i=0; i<n; i++)*

*tmp.A[i] = A[i] \* AF.A[i];*

*tmp.size = n;*

*return tmp;*

*}*

*};*

Використання класу ArrayFloat в іншому методі

*// додаткові змінні та масиви*

*float x, y;*

*float AF1[] = { 2, 5, 7, 9, 12 };*

*float AF2[] = { 3, 4, 9, 8, 10, 13 };*

*// створити об'єкти класу ArrayFloat*

*ArrayFloat A1(5, AF1);*

*ArrayFloat A2(6, AF2);*

*ArrayFloat A3;*

*// виклик операторної функції operator\**

*A3 = A1 \* A2; // здійснюється поелементне множення*

*// перевірка*

*x = A3.GetAi(0); // x = 6*

*y = A3.GetAi(1); // y = 20*

*x = A3.GetAi(2); // x = 63*

*y = A3.GetAi(4); // y = 120*

*Приклад 21.3.* Задано клас ArrayFloat, що реалізує динамічний масив чисел типу float. У класі реалізовано:

* внутрішні змінні size, **A**, які визначають розмір масиву та сам масив;
* два конструктори, які ініціалізують початковими значення елементи масиву;
* конструктор копіювання;
* методи доступу GetSize(), SetSize(), GetAi(), SetAi(), які реалізують доступ до внутрішніх змінних масиву з відповідними операціями (виділення пам’яті, перевірка на допустимі межі);
* операторну функцію operator=(), яка реалізує копіювання об’єктів;
* операторну функцію operator+(), яка реалізує перевантаження оператора **‘+’** для масивів типу ArrayFloat. Операторна функція додає поелементно значення масивів, які є операндами операції ‘+’;
* деструктор.

*// клас - масив типу float*

*class ArrayFloat*

*{*

*private:*

*int size; // розмір масиву*

*float \* A; // динамічний розмір масиву*

*public:*

*// конструктори класу*

*// конструктор без параметрів*

*ArrayFloat()*

*{*

*size = 0;*

*A = NULL;*

*}*

*// конструктор з двома параметрами*

*ArrayFloat(int nsize, float \* nA)*

*{*

*if (size>0)*

*delete A;*

*size = nsize;*

*A = new float[size];*

*for (int i=0; i<nsize; i++)*

*A[i] = nA[i];*

*}*

*// конструктор копіювання*

*ArrayFloat(const ArrayFloat& \_A)*

*{*

*size = \_A.size;*

*A = new float[size];*

*for (int i = 0; i < size; i++)*

*A[i] = \_A.A[i];*

*}*

*// методи доступу*

*int GetSize(void) { return size; }*

*void SetSize(int nsize)*

*{*

*if (size>0)*

*{*

*delete A;*

*size = 0;*

*}*

*size = nsize;*

*A = new float[size]; // виділити новий фрагмент пам'яті*

*// заповнити масив нулями*

*for (int i=0; i<size; i++)*

*A[i] = 0.0f;*

*}*

*float GetAi(int index)*

*{*

*if ((index>=0) && (index<size))*

*return A[index];*

*else*

*return 0;*

*}*

*void SetAi(int index, float value)*

*{*

*if ((index>=0) && (index<size))*

*A[index] = value;*

*}*

*// оператор копіювання operator=()*

*ArrayFloat operator=(const ArrayFloat& \_A)*

*{*

*if (size > 0)*

*delete[] A;*

*size = \_A.size;*

*A = new float[size]; // виділити пам'ять*

*for (int i = 0; i < size; i++)*

*A[i] = \_A.A[i];*

*return \*this;*

*}*

*// перевантаження оператора '+',*

*// операторна функція*

*ArrayFloat operator+(ArrayFloat AF2)*

*{*

*int n;*

*// взяти мінімальний розмір з двох масивів*

*if (size<AF2.size) n = size;*

*else n = AF2.size;*

*ArrayFloat tmpA; // об'єкт класу*

*tmpA.SetSize(n); // новий розмір масиву*

*// поелементне додавання*

*for (int i=0; i<n; i++)*

*tmpA.A[i] = A[i] + AF2.A[i];*

*return tmpA;*

*}*

*// деструктор*

*~ArrayFloat()*

*{*

*if (size > 0)*

*delete[] A;*

*}};*

Нижче продемонстровано використання класу ArrayFloat та операторної функції operator+() цього класу.

*// додаткові змінні, масиви*

*float x, y;*

*float F1[] = { 3.8f, 2.9f, 1.5f };*

*float F2[] = { 4.3f, 1.5f, 7.0f, 3.3f };*

*// оголошення об'єктів класу ArrayFloat*

*ArrayFloat AF1(3, F1);*

*ArrayFloat AF2(4, F2);*

*// перевірка*

*x = AF1.GetAi(0); // x = 3.8*

*y = AF2.GetAi(3); // y = 3.3*

*// оголошення додаткового об'єкта - результату*

*ArrayFloat AF3;*

*AF3 = AF1 + AF2; // виклик операторної функції operator+*

*// перевірка*

*x = AF3.GetAi(0); // x = 3.8 + 4.3 = 8.1*

*y = AF3.GetAi(1); // y = 2.9 + 1.5 = 4.4*

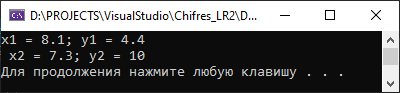
*// ще один виклик*

*AF3 = AF1 + AF1 + AF2;*

*x = AF3.GetAi(1); // x = 2.9 + 2.9 + 1.5 = 7.3*

*y = AF3.GetAi(2); // y = 1.5 + 1.5 + 7.0 = 10.0*

Результат роботи програми наведений нижче:



На використання перевантажених операторів накладаються наступні обмеження:

* при перевантаженні оператору не можна змінити пріоритет цього оператора;
* не можна змінити кількість операндів оператора. Однак, у коді операторної функції можна один з параметрів (операндів) не використовувати;
* не можна перевантажувати оператори :: . \* ?:;
* не можна викликати операторну функцію з аргументами за замовчуванням. Виняток – операторна функція виклику функції operator()().

Не можна перевантажувати наступні оператори:

* **::** – розширення області видимості;
* **.** (крапка) – вибір члена класу або структури;
* – доступ за покажчиком;
* **?:** – тернарний оператор?

Операторна функція може повертати об’єкти будь-яких типів. Найчастіше операторна функція повертає об’єкт типу класу, в якому вона реалізована або з якими вона працює.

*Приклад 21.4.* Задано клас Complex, в якому перевантажується два оператори:

* унарний оператор **‘+’**, який повертає модуль комплексного числа (тип double);
* бінарний оператор **‘+’**, який повертає суму комплексних чисел. Операторна функція повертає об’єкт типу Complex;
* бінарний оператор **‘+’**, який додає до комплексного числа деяке дійсне число. У цьому випадку операторна функція отримує вхідним параметром дійсне число і повертає об’єкт типу Complex.

Текст класу наступний:

*// клас Complex*

*class Complex*

*{*

*private:*

*float real; // дійсна частина*

*float imag; // уявна частина*

*public:*

*// конструктори*

*Complex(void)*

*{*

*real = imag = 0;*

*}*

*Complex(float \_real, float \_imag)*

*{*

*real = \_real;*

*imag = \_imag;*

*}*

*// методи доступу*

*float GetR(void) { return real; }*

*float GetI(void) { return imag; }*

*void SetRI(float \_real, float \_imag)*

*{*

*real = \_real;*

*imag = \_imag;*

*}*

*// оголошення операторної функції, яка перевантажує бінарний '+'*

*// функція повертає об'єкт, що містить суму двох комплексних чисел*

*Complex operator+(Complex c)*

*{*

*Complex c2; // тимчасовий об'єкт*

*// додавання комплексних чисел*

*c2.real = real + c.real;*

*c2.imag = imag + c.imag;*

*return c2;*

*}*

*// оголошення операторної функції, яка перевантажує унарний '+'*

*// функція повертає модуль комплексного числа*

*float operator+(void)*

*{*

*float res;*

*res = std::sqrt(real\*real+imag\*imag);*

*return res;*

*}*

*// оголошення операторної функції operator+()*

*// функція додає до комплексного числа деяке число, яке є вхідним параметром*

*Complex operator+(float real)*

*{*

*Complex c2; // результуючий об'єкт*

*c2.real = this->real + real;*

*c2.imag = this->imag;*

*return c2;*

*}*

*};*

Нижче демонструється використання класу Complex та перевантажених операторних функцій в деякому іншому методі

*Complex c1(1,5);*

*Complex c2(3,-8);*

*Complex c3; // результуючий об'єкт*

*double d;*

*// перевірка*

*c3 = c1 + c2;*

*d = c3.GetR(); // d = 1 + 3 = 4*

*d = c3.GetI(); // d = 5 + (-8) = -3*

*// перевантажений унарний оператор '+'*

*d = +c1; // d = |1 + 5j| = 5.09902 - модуль числа*

*d = +c2; // d = |3 + (-8)j| = 8.544*

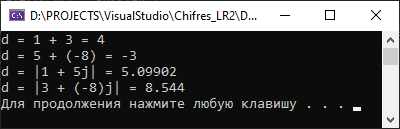
*// виклик перевантаженого бінарного '+',*

*// додати до комплексного числа число*

*c3 = c1 + 5.0;*

*d = c3.GetR(); // d = 1 + 5 = 6*

Результат роботи програми наведений нижче:



Так, можна. Однак такі дії не є корисними з точки зору здорового глузду. Так, наприклад, операція множення

6 \* 9

не змінює значення своїх операндів 6 та 9. Результат рівний 54. Якщо операторна функція operator\*() буде змінювати значення своїх операндів, то це може призвести до невидимих помилок в програмах, оскільки програміст за звичкою, буде вважати, що значення операндів є незмінні.

Операторна функція не може мати декілька реалізацій в класі з однаковою сигнатурою параметрів (коли типи та кількість параметрів співпадають). У випадку порушення цього правила компілятор видає помилку:

cannot overload functions distinguished by return type alone

що означає не можна перевантажувати функції, що відрізняються тільки типом повернення

*Приклад 21.5.*Не можна в класі перевантажувати оператор **‘+’** так як показано нижче

*class SomeClass*

*{*

*// ...*

*SomeClass operator+(SomeClass c1)*

*{*

*// ...*

*}*

*double operator+(SomeClass c1) // це є помилка!*

*{*

*// ...*

*}*

*// ...*

*}*

Це правило стосується будь-яких функцій класу.

***Порядок виконання роботи:***

1. Розглянути приклади, наведені у теоретичних відомостях.

**Лабораторна робота №22**

**“Дружні” операторні функції: відмінності, реалізація, особливості застосування. Перевантаження операторів +, –, \*, / з допомогою “дружніх” операторних функцій**

*Мета роботи*: набуття навичок роботи з перевантаженими операторами.

*Теоретичні основи*:

Як відомо, існує два способи перевантаження будь-якого оператору:

* з допомогою операторної функції, яка реалізована всередині класу;
* з допомогою операторної функції, яка реалізована як “дружня” (friend) до класу.

Між способами реалізації операторних функцій існує відмінність у кількості параметрів, які отримує операторна функція:

* для унарних операторів операторна функція всередині класу не отримує параметрів. А “дружня” до класу операторна функція отримує один параметр;
* для бінарних операторів операторна функція всередині класу отримує один параметр. А “дружня” функція отримує два параметри. У цьому випадку першим параметром “дружньої” функції є лівий операнд, а другим параметром є правий операнд.

Ці відмінності виникають через те, що “дружня” операторна функція не отримує неявного покажчика this. Тому в ній потрібно явно задавати параметри

Операторна функція може бути реалізована за межами класу. Якщо операторна функція перевантажує унарний оператор, то вона містить один параметр. Якщо операторна функція перевантажує бінарний оператор, то вона містить два параметри. Загальна форма операторної функції, що є дружньою до класу має вигляд

*return\_type operator#(arguments\_list)*

*{*

*// деякі операції*

*// ...*

*}*

де

* return\_type – тип значення, що повертає операторна функція;
* operator# – ключове слово, що визначає операторну функцію в класі. Символ # визначає оператор мови C++, який перевантажується. Наприклад, якщо перевантажується оператор **+**, то потрібно вказати operator+;
* argument\_list – список параметрів, які отримує операторна функція. Якщо у “дружній” функції перевантажується бінарний оператор, то argument\_list містить два аргументи. Якщо перевантажується унарний оператор, то argument\_list містить один аргумент.

У класі ця функція має бути оголошена як “дружня” з ключовим словом friend.

*Приклад 22.1.*За даним прикладом можна розробляти власні операторні функції, які є “дружніми” до заданого класу. Задано клас Complex, який реалізує комплексне число. У класі оголошуються внутрішні змінні, конструктори, методи доступу та “дружня” функція operator-(). “Дружня” функція operator-(), що реалізована за межами класу, здійснює віднімання комплексних чисел.

*// клас Complex*

*class Complex*

*{*

*private:*

*float real; // дійсна частина*

*float imag; // уявна частина*

*public:*

*// конструктори*

*Complex(void)*

*{*

*real = imag = 0;*

*}*

*Complex(float \_real, float \_imag)*

*{*

*real = \_real;*

*imag = \_imag;*

*}*

*// методи доступу*

*float GetR(void) { return real; }*

*float GetI(void) { return imag; }*

*void SetRI(float \_real, float \_imag)*

*{*

*real = \_real;*

*imag = \_imag;*

*}*

*// оголошення "дружньої" до класу Complex операторної функції*

*friend Complex operator-(Complex c1, Complex c2);*

*};*

*// "дружня" до класу Complex операторна функція,*

*// реалізована за межами класу,*

*// здійснює віднімання комплексних чисел*

*Complex operator-(Complex c1, Complex c2)*

*{*

*Complex c; // створити об'єкт класу Complex*

*// віднімання комплексних чисел*

*c.real = c1.real - c2.real;*

*c.imag = c1.imag - c2.imag;*

*return c;*

*}*

Використання класу Complex в іншому методі

*// використання "дружньої" операторної функції*

*Complex c1(5,6);*

*Complex c2(3,-2);*

*Complex c3; // результат*

*float a, b;*

*// перевірка*

*a = c1.GetR(); // a = 5*

*b = c1.GetI(); // b = 6*

*// виклик "дружньої" до класу Complex операторної функції*

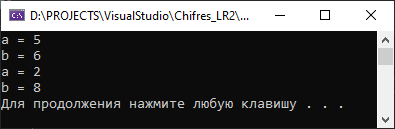
*c3 = c1 - c2;*

*// результат*

*a = c3.GetR(); // a = 5-3 = 2*

*b = c3.GetI(); // b = 6-(-2) = 8*

Результат роботи програми наведений нижче:



Як видно з вищенаведеного прикладу, “дружня” функція operator-() отримує два параметри. Перший параметр відповідає лівому операнду в бінарному операторі віднімання **‘–’**. Другий параметр відповідає правому операнду.

*Приклад 22.2.* У прикладі продемонстровано, як правильно реалізовувати перевантаження “дружніх” операторних функцій для класів, в яких пам’ять виділяється динамічно. Клас містить динамічний масив елементів типу float. Оголошується клас ArrayFloat, який реалізує динамічний масив дійсних чисел. У класі реалізовані такі елементи:

* внутрішня змінна size – розмір масиву;
* покажчик A на тип float. Це є масив, пам’ять для якого виділяється динамічно;
* конструктор класу ArrayFloat() без параметрів;
* конструктор з двома параметрами ArrayFloat(int, float\*);
* конструктор копіювання ArrayFloat(const ArrayFloat&). Цей конструктор необхідний для уникнення недоліків побітового копіювання. Без конструктора копіювання робота об’єктів класу буде мати помилки;
* деструктор ~ArrayFloat(). У деструкторі звільняється пам’ять, виділена для динамічного масиву A;
* методи GetSize(), SetSize() для доступу до змінної size;
* методи GetAi(), SetAi() для доступу до конкретного елементу масиву з заданим індексом;
* метод FormArray(), який формує елементи масиву за деякою формулою;
* метод Print(), який виводить масив;
* операторна функція operator=(const ArrayFloat&). Ця функція перевантажує оператор присвоєння **=**, який реалізує копіювання об’єктів. Функція необхідна для уникнення недоліків побітового копіювання у випадку присвоювання об’єктів obj1=obj2;
* “дружня” до класу ArrayFloat операторна функція operator/(const ArrayFloat&, const ArrayFloat&), яка перевантажує оператор ділення **/** для даного класу. Функція реалізує поелементне ділення масивів. Якщо кількість елементів масивів не співпадає, то результуючий масив встановлюється в розмір найменшого масиву.

Реалізація класу для додатку типу Console Application наступна:

*#include <iostream>*

*using namespace std;*

*// клас - масив типу float*

*class ArrayFloat*

*{*

*private:*

*int size;*

*float \* A; // динамічний розмір масиву*

*public:*

*// конструктори класу*

*// конструктор без параметрів*

*ArrayFloat()*

*{*

*size = 0;*

*A = nullptr;*

*}*

*// конструктор з двома параметрами*

*ArrayFloat(int nsize, float \* nA)*

*{*

*size = nsize;*

*A = new float[size];*

*for (int i = 0; i < nsize; i++)*

*A[i] = nA[i];*

*}*

*// конструктор копіювання,*

*// потрібний для уникнення недоліків побітового копіювання*

*ArrayFloat(const ArrayFloat& nA)*

*{*

*if (size > 0)*

*{*

*delete[] A; // звільнити попередньо виділену пам'ять*

*}*

*// виділити пам'ять для A по іншому*

*A = new float[nA.size];*

*// виконати копіювання \*this <= nA*

*size = nA.size;*

*for (int i = 0; i < size; i++)*

*A[i] = nA.A[i];*

*}*

*// деструктор*

*~ArrayFloat()*

*{*

*// звільнити пам'ять, виділену для масиву A*

*if (size > 0)*

*delete[] A;*

*}*

*// методи доступу*

*int GetSize(void) { return size; }*

*void SetSize(int nsize)*

*{*

*if (size > 0)*

*delete[] A;*

*size = nsize;*

*A = new float[size]; // виділити новий фрагмент пам'яті*

*// заповнити масив нулями*

*for (int i = 0; i < size; i++)*

*A[i] = 0.0f;*

*}*

*// зчитати значення за індексом index*

*float GetAi(int index)*

*{*

*if ((index >= 0) && (index < size))*

*return A[index];*

*else*

*return 0;*

*}*

*// встановити нове значення*

*void SetAi(int index, float value)*

*{*

*if ((index >= 0) && (index < size))*

*A[index] = value;*

*}*

*// метод, що формує масив*

*void FormArray()*

*{*

*for (int i = 0; i < size; i++)*

*A[i] = (float)i;*

*}*

*// метод, що відображає масив*

*void Print(const char\* objName)*

*{*

*cout << "Object: " << objName << endl;*

*for (int i = 0; i < size; i++)*

*cout << A[i] << " ";*

*cout << endl << endl;*

*}*

*// Перевантаження операторів*

*// Перевантаження оператору копіювання operator=(const ArrayFloat&)*

*ArrayFloat operator=(const ArrayFloat& nA)*

*{*

*if (size > 0)*

*delete[] A; // звільнити попередньо виділену пам'ять*

*// виділити пам'ять для A по іншому*

*A = new float[nA.size];*

*// виконати копіювання \*this <= nA*

*size = nA.size;*

*for (int i = 0; i < size; i++)*

*A[i] = nA.A[i];*

*return \*this;*

*}*

*// "дружня" операторна функція '/',*

*// тільки оголошення, реалізація за межами класу*

*friend ArrayFloat operator/(const ArrayFloat& A1, const ArrayFloat& A2);*

*};*

*// "дружня" до класу ArrayFloat операторна функція operator/()*

*// функція отримує два параметри A1, A2*

*ArrayFloat operator/(const ArrayFloat& A1, const ArrayFloat& A2)*

*{*

*// створити об'єкт класу ArrayFloat*

*ArrayFloat A; // викликається конструктор без параметрів*

*int n;*

*// взяти мінімальне значення з розміру двох масивів A1, A2*

*n = A1.size;*

*if (n > A2.size) n = A2.size;*

*// встановити новий розмір масиву A, перерозподіляється пам'ять*

*A.SetSize(n);*

*// поелементне ділення*

*for (int i = 0; i < n; i++)*

*{*

*if (A2.A[i] != 0) // обійти ділення на 0*

*A.A[i] = A1.A[i] / A2.A[i];*

*else*

*A.A[i] = 0.0f;*

*}*

*return A; // повернути новий об'єкт*

*}*

*void main()*

*{*

*// Демонстрація роботи класу ArrayFloat*

*ArrayFloat AF1; // конструктор без параметрів*

*AF1.FormArray();*

*int size; // додаткові змінні*

*float x;*

*size = AF1.GetSize(); // size = 0*

*x = AF1.GetAi(3); // x = 0*

*cout << "size = " << size << endl;*

*cout << "x = " << x << endl;*

*ArrayFloat AF2 = AF1; // виклик конструктора копіювання*

*cout << AF2.GetAi(3) << endl;*

*ArrayFloat AF3;*

*AF3.SetSize(10);*

*AF3.FormArray();*

*AF2 = AF3; // виклик оператора копіювання operator=()*

*cout << "AF2[3] = " << AF2.GetAi(3) << endl;*

*ArrayFloat AF4 = AF2; // виклик конструктора копіювання*

*cout << "AF4[5] = " << AF2.GetAi(5) << endl;*

*// оператор копіювання у вигляді "ланцюжка"*

*AF1 = AF2 = AF4 = AF3; // виклик оператора копіювання operator=()*

*cout << "AF1[8] = " << AF1.GetAi(8) << endl;*

*// Демонстрація "дружньої" операторної функції operator/*

*AF3 = AF2 / AF1; // виклик "дружньої" функції operator/()*

*cout << endl << endl;*

*AF3.Print("AF3");*

*// "Дружня" функція працює у вигляді ланцюжка*

*ArrayFloat AF5;*

*AF5 = AF1 / AF2 / AF3;*

*cout << endl;*

*AF5.Print("AF5");*

*//*

*float F[] = { 1.1, 2.3, 4.5 };*

*ArrayFloat AF6(3, F); // конструктор з двома параметрами*

*AF6.Print("AF6");*

*}*

Результат роботи програми:

*size = 0*

*x = 0*

*0*

*AF2[3] = 3*

*AF4[5] = 5*

*AF1[8] = 8*

*Object: AF3*

*0 1 1 1 1 1 1 1 1 1*

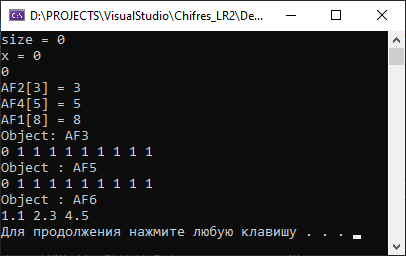
*Object: AF5*

*0 1 1 1 1 1 1 1 1 1*

*Object: AF6*

*1.1 2.3 4.5*

Результат роботи програми наведений нижче:



***Порядок виконання роботи:***

1. Розглянути приклади, наведені у теоретичних відомостях.

**Лабораторна робота №23**

**Перевантаження оператора +. Динамічне виділення пам’яті у класі**

*Мета роботи*: набуття навичок роботи з перевантаженими операторами.

*Теоретичні основи*:

*Приклад 23.1.* Оголошується клас DoubleArray, в якому реалізовані:

* внутрішня прихована (private) змінна-покажчик p на тип double. Ця змінна є масив чисел. Для покажчика p пам’ять буде виділятись динамічно;
* внутрішня private-змінна size – розмір масиву;
* конструктор без параметрів DoubleArray();
* конструктор з одним параметром DoubleArray(int size). У цьому конструкторі пам’ять для масиву p виділяється динамічно. Розмір масиву визначається вхідним параметром size;
* конструктор копіювання DoubleArray(const DoubleArray& A). Цей конструктор необхідний для уникнення побітового копіювання. Побітове копіювання має суттєві недоліки у випадку динамічного виділення пам’яті в класі. У конструкторі копіювання для новоствореного об’єкту пам’ять виділяється власним способом;
* операторна функція operator+(DoubleArray&). Ця функція перевантажує оператор **+** таким чином, що можна додавати об’єкти класу DoubleArray між собою за формою obj1+obj2. В операторній функції масиви додаються поелементно. Якщо один масив коротше за інший, тоді елементи меншого масиву вважаються рівними 0;
* операторна функція operator=(const DoubleArray&). Ця функція перевантажує оператор присвоєння **=**. Функція є необхідною для того, щоб уникнути побітового копіювання при присвоюванні об’єктів на зразок obj3=obj1+obj2 або obj3=obj2=obj1. Якщо не визначити цю функцію в класі, то буде викликаний неявний оператор присвоювання, який здійснює побітове копіювання що є неприпустимим для нашого класу;
* деструктор класу ~DoubleArray(). У деструкторі класу виділена пам’ять звільняється;
* функція Show(), яка відображає вміст масиву p.

Реалізація класу DoubleArray та демонстрація його роботи у функції main() наступна

*#include <iostream>*

*#include <time.h>*

*using namespace std;*

*// динамічний масив*

*class DoubleArray*

*{*

*double \* p;*

*int size;*

*public:*

*// конструктори*

*DoubleArray()*

*{*

*size = 0;*

*p = nullptr;*

*}*

*DoubleArray(int size)*

*{*

*this->size = size;*

*p = new double[size];*

*for (int i=0; i<size; i++)*

*p[i] = 0;*

*}*

*// деструктор*

*~DoubleArray()*

*{*

*if (size>0)*

*delete p;*

*}*

*// сформувати масив випадково*

*void FormArrayRandom()*

*{*

*srand(time(NULL));*

*for (int i=0; i<size; i++)*

*p[i] = rand()%10;*

*}*

*// сформувати масив за формулою*

*void FormArray()*

*{*

*for (int i=0; i<size; i++)*

*p[i] = i;*

*}*

*// методи доступу*

*double GetAi(int index)*

*{*

*if (p!=nullptr)*

*{*

*if ((0<=index)&&(index<size))*

*return p[index];*

*}*

*else*

*return 0.0;*

*}*

*void SetAi(int index, double value)*

*{*

*if (p!=nullptr)*

*{*

*if ((index>=0)&&(index<size))*

*p[index] = value;*

*}*

*}*

*// відобразити масив*

*void Show()*

*{*

*cout << "The Array:\n";*

*for (int i=0; i<size; i++)*

*cout << "p["<<i<<"] = " << p[i] << endl;*

*cout << "------------------------------" << endl;*

*}*

*// додати конструктор копіювання, щоб коректно виконувався деструктор та ініціалізація об'єкту*

*DoubleArray(const DoubleArray& A)*

*{*

*size = A.size;*

*// виділити пам'ять по своєму*

*p = new double[size];*

*// заповнити значеннями*

*for (int i=0; i<size; i++)*

*p[i] = A.p[i];*

*}*

*// Додати операторну функцію, що перевантажує оператор присвоєння =*

*// Необхідно для коректної роботи оператора присвоєння (уникнення побітового копіювання)*

*DoubleArray operator=(const DoubleArray& A)*

*{*

*size = A.size;*

*// виділити пам'ять для покажчиків*

*p = new double[size];*

*// заповнити значеннями кожен елемент масиву*

*for (int i=0; i<size; i++)*

*p[i] = A.p[i];*

*// повернути поточний об'єкт,*

*// викликається конструктор копіювання DoulbeArray(DoubleArray&)*

*return \*this;*

*}*

*// Операторна функція, що перевантажує оператор +.*

*// Поелементне додавання масивів, якщо один масив коротше за інший,*

*// тоді елементи меншого масиву вважаються рівними 0.*

*DoubleArray operator+(DoubleArray& A)*

*{*

*int min = size; // розмір коротшого масиву*

*int max = size; // розмір довшого масиву*

*if (min>A.size) min = A.size;*

*if (max<A.size) max = A.size;*

*DoubleArray objA;*

*objA.size = max;*

*objA.p = new double[objA.size]; // виділити пам'ять*

*// цикл поелементного додавання*

*for (int i=0; i<max; i++)*

*{*

*if (i<min)*

*objA.p[i] = p[i] + A.p[i];*

*else*

*{*

*if (size<max)*

*objA.p[i] = A.p[i];*

*else*

*objA.p[i] = p[i];*

*}*

*}*

*return objA;*

*}*

*};*

*void main()*

*{*

*// Демонстрація - клас DoubleArray, що використовує динамічний масив елементів*

*DoubleArray A(5);*

*A.FormArray();*

*A.Show();*

*DoubleArray A2=A;*

*// те саме стосується й оператора присвоєння*

*DoubleArray A3;*

*A.SetAi(2,55);*

*A3 = A; // виклик операторної функції operator=(const DoubleArray& A)*

*A3.Show();*

*// Оператор присвоювання у вигляді ланцюжка*

*DoubleArray A4;*

*A4 = A2 = A3; // працює коректно*

*cout<<"A4=A2=A3:"<<endl;*

*A4.Show();*

*cout<<"A2:"<<endl;*

*A2.Show();*

*cout<<"A3:"<<endl;*

*A3.Show();*

*// перевантаження оператора +*

*DoubleArray A5;*

*A5 = A2+A2+A3+A4;*

*A5.Show();*

*DoubleArray A6(3);*

*A6.SetAi(0, 1.5);*

*A6.SetAi(1, 2.2);*

*A6.SetAi(2, 1.8);*

*cout<<"A6:"<<endl;*

*A6.Show();*

*cout<<"A6=A5+A6:"<<endl;*

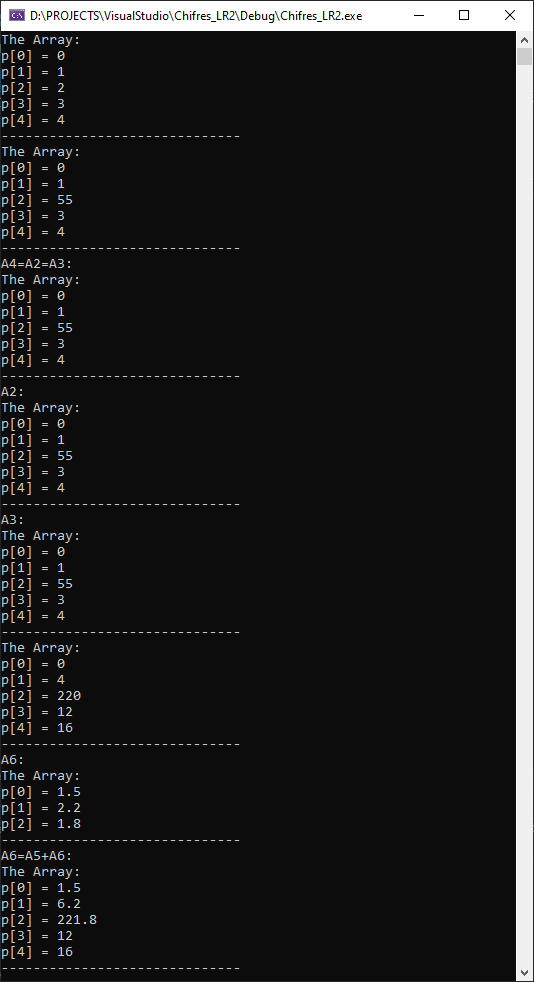
*A6 = A5+A6;*

*A6.Show();*

*system("pause");*

*}*

Результат роботи програми наведений нижче:



***Порядок виконання роботи:***

1. Розглянути приклади, наведені у теоретичних відомостях.

**Лабораторна робота №24**

**Перевантаження інкрементних та декрементних операторів ++, – –**

*Мета роботи*: набуття навичок роботи з перевантаженими операторами.

*Теоретичні основи*:

Інкрементні та декрементні оператори в мові C++ можуть бути перевантажені. Особливість перевантаження цих операторів полягає в тому, що потрібно перевантажувати як префіксну так і постфіксну форму цих операторів.

Як відомо, у мові C++ операції **++** та **– –** мають префіксну і постфіксну форми, як показано у прикладі:

++x; // префіксна форма оператора інкременту

x++; // постфіксна форма оператора

--x; // префіксна форма оператора декременту

x--; // постфіксна форма оператора декременту

Згідно синтаксису C++ оператори **++** та **—** потребують одного операнда.

Для того, щоб при реалізації в класі, відрізнити префіксну та постфіксну форми реалізації операторної функції **++** або **—**, потрібно дотримуватись наступних правил:

* якщо перевантажується префіксна форма оператора **++**, то в класі потрібно реалізувати операторну функцію operator++() без параметрів;
* якщо перевантажується префіксна форма оператора **– –**, то в класі потрібно реалізувати операторну функцію **operator- -()** без параметрів;
* якщо перевантажується постфіксн форма оператора **++**, то в класі потрібно реалізувати операторну функцію operator++(int d) з одним цілочисельним параметром. У цьому випадку параметр d не використовується у функції. Він вказується тільки для того, щоб вказати що це саме постфіксна реалізація оператора **++**. Ім’я d може бути замінене іншим іменем;
* якщо перевантажується постфіксна форма оператора **– –**, то в класі потрібно реалізувати операторну функцію operator–(int d) з одним цілочисельним параметром. Параметр d необхідний для вказання того, що перевантажується постфіксна реалізація оператора **– –**.

*Приклад 24.1.*

*class SomeClass*

*{*

*// ...*

*// операторна функція, яка переванажує префіксний оператор ++ (++X)*

*type operator++()*

*{*

*// тіло операторної функції*

*// ...*

*}*

*// операторна функція, яка переванажує постфіксний оператор ++ (X++)*

*type operator++(int d)*

*{*

*// тіло операторної функції*

*// ...*

*}*

*// операторна функція, яка переванажує префіксний оператор -- (--X)*

*type operator--()*

*{*

*// тіло операторної функції*

*// ...*

*}*

*// операторна функція, яка переванажує постфіксний оператор -- (X--)*

*type operator--(int d)*

*{*

*// тіло операторної функції*

*// ...*

*}*

*};*

*Приклад 24.2.* Демонстрється реалізація операторних функцій для класу Integer, який реалізує цлочисельне значення. Оголошується клас Integer, який містить:

* внутрішню змінну number цілого типу. Ця змінна зберігає цілочисельне значення;
* два конструктори, які ініціалізують змінну number;
* методи доступу до змінної number Get(), Set();
* дві операторні функції, які перевантажують оператор **++**;
* дві операторні функції, що перевантажують оператор **– –**.

Реалізація класу Integer наступна:

*// перевантаження інкрементних операторів для класу Integer*

*class Integer*

*{*

*private:*

*int number; // член даних*

*public:*

*// конструктори класу*

*// конструктор без параметрів*

*Integer() { number = 0; }*

*// конструктор з 1 параметром*

*Integer(int \_number) { number = \_number; }*

*// методи доступу*

*void Set(int \_number) { number = \_number; }*

*int Get(void) { return number; }*

*// операторна функція, яка перевантажує*

*// префіксну форму оператора ++ для класу Integer*

*Integer operator++(void)*

*{*

*number++;*

*return \*this; // повернути об'єкт даного класу*

*}*

*// операторна функція, яка перевантажує*

*// постфіксну форму оператора ++ для класу Integer*

*Integer operator++(int d) // параметр d не використовується*

*{*

*number++;*

*return \*this;*

*}*

*// перевантаження префіксного --*

*Integer operator--(void)*

*{*

*number--;*

*return \*this;*

*}*

*// перевантаження постфіксного --*

*Integer operator--(int d) // параметр d не використовується*

*{*

*number--;*

*return \*this;*

*}*

*};*

Нижче наведено використання класу Integer

*// демонстрація використання перевантажених інкрементних*

*// операторних функцій*

*Integer d1, d2(5); // оголошення об'єктів класу Integer*

*Integer d3;*

*int t1, t2;*

*// перевірка*

*t1 = d1.Get(); // t1 = 0*

*t2 = d2.Get(); // t2 = 5*

*// виклик перевантажених операторних функцій*

*++d1; // виклик префіксної операторної функції operator++()*

*t1 = d1.Get(); // t1 = 1*

*d2++; // виклик постфіксної операторної функції operator++(int)*

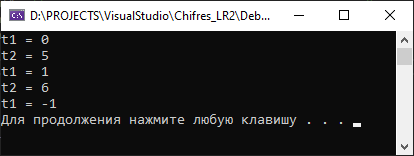
*t2 = d2.Get(); // t2 = 6*

*--d1; // префіксний --*

*d1--; // постфіксний --*

*t1 = d1.Get(); // t1 = -1*

Результат роботи програми наведений нижче:



*Приклад 24.3.* Оголошується клас ArrayInt, в якому продемонстровано перевантаження префіксних та постфіксних операторних функцій operator++() та operator- -(), які відповідно збільшують та зменшують на 5 кожен елемент масиву. Оголошення класу ArrayInt має вигляд:

*// клас, що реалізує масив цілих чисел*

*class ArrayInt*

*{*

*private:*

*int A[100]; // внутрішня змінна - масив цілих чисел*

*int size; // кількість елементів масиву*

*public:*

*// конструктори*

*ArrayInt() { size = 0; }*

*ArrayInt(int \_size)*

*{*

*if (\_size>100) \_size=100;*

*size = \_size;*

*for (int i=0; i<size; i++)*

*A[i] = 0;*

*}*

*// методи доступу*

*void SetAi(int index, int value)*

*{*

*if ((index>=0) && (index<=size))*

*A[index] = value;*

*}*

*int GetAi(int index) { return A[index]; }*

*int GetSize(void) { return size; }*

*// перевантажений оператор ++*

*// префіксна форма*

*ArrayInt operator++(void)*

*{*

*for (int i=0; i<size; i++)*

*A[i] = A[i] + 5;*

*return \*this;*

*}*

*// постфіксна форма оператора ++, параметр t не використовується*

*ArrayInt operator++(int t)*

*{*

*for (int i=0; i<size; i++)*

*A[i] = A[i] + 5;*

*return \*this;*

*}*

*// префіксна форма оператора --*

*ArrayInt operator--(void)*

*{*

*for (int i=0; i<size; i++)*

*A[i] = A[i] - 5;*

*return \*this;*

*}*

*// постфіксна форма оператора --, параметр t не використовується*

*ArrayInt operator--(int t)*

*{*

*for (int i=0; i<size; i++)*

*A[i] = A[i] - 5;*

*return \*this;*

*}*

*};*

Використання класу ArrayInt може бути таким

*// демонстрація використання перевантажених інкрементних*

*// операторних функцій*

*ArrayInt AA(10);*

*int t;*

*// ініціалізація масиву AA деякими значеннями*

*for (int i=0; i<AA.GetSize(); i++)*

*AA.SetAi(i, i+1);*

*// перевірка*

*t = AA.GetAi(2); // t = 3*

*// виклик перевантажених операторних функцій*

*AA++;*

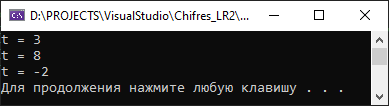
*t = AA.GetAi(2); // t = 3+5 = 8*

*--AA;*

*AA--;*

*t = AA.GetAi(2); // t = 8-5-5 = -2*

Результат роботи програми наведений нижче:



Якщо оператори **++** та **—** перевантажуються з допомого дружніх функцій до класу, то потрібно враховувати наступне:

* дружня до класу функція не отримує покажчика this класу;
* оскільки дружня функція не отримує покажчика this, то параметр у цю функцію має передаватись за посиланням а не за значенням. Це означає, що перед іменем параметру вказується символ посилання &.

Для того, щоб при реалізації дружньої функції до класу, відрізнити префіксну та постфіксну форми реалізації операторної функції **++** або —, потрібно дотримуватись наступних правил:

* якщо перевантажується префіксна форма оператора **++** (**++**x), то “дружня” операторна функція повинна отримувати один параметр. Цей параметр є посиланням на клас, для якого ця функція реалізована;
* якщо перевантажується префіксна форма оператора **– –** (**– –**x), то “дружня” операторна функція повинна отримувати один параметр, який є посиланням на клас в якому ця функція реалізована;
* якщо перевантажується постфіксна форма оператора **++** (x**++**), то “дружня” операторна функція отримує два параметри. Перший параметр є посиланням на дружній клас. Другий параметр є формальним параметром цілого типу, який служить індикатором того, що перевантажується саме постфіксна а не префіксна форма оператора **++**. Другий параметр у функції не використовується;
* якщо перевантажується постфіксна форма оператора **– –** (x**– –**), то дружня операторна функція отримує два параметри. Перший параметр є посиланням на дружній клас. Другий параметр не використовується, а служить для визначення того, що саме постфіксна форма оператора **– –** реалізована.

*Приклад 24.4.*

*// оголошення деякого класу*

*class SomeClass*

*{*

*// тіло класу*

*// ...*

*// оголошення дружніх функцій для класу SomeClass*

*friend SomeClass operator++(SomeClass & ref); // ++x*

*friend SomeClass operator--(SomeClass & ref); // --x*

*friend SomeClass operator++(SomeClass & ref, int d); // x++*

*friend SomeClass operator--(SomeClass & ref, int d); // x--*

*};*

*// оголошення операторної функції,*

*// яка перевантажує префіксний оператор ++ (++x)*

*SomeClass operator++(SomeClass & ref)*

*{*

*// ...*

*}*

*// оголошення операторної функції,*

*// яка перевантажує префіксний оператор -- (--x)*

*SomeClass operator++(SomeClass & ref)*

*{*

*// ...*

*}*

*// оголошення операторної функції,*

*// яка перевантажує постфіксний оператор ++ (x++)*

*SomeClass operator++(SomeClass & ref, int d)*

*{*

*// ...*

*}*

*// оголошення операторної функції,*

*// яка перевантажує постфіксний оператор -- (x--)*

*SomeClass operator++(SomeClass & ref, int d)*

*{*

*// ...*

*}*

Як видно з прикладу, “дружня” операторна функція отримує посилання на “дружній” клас.

*Приклад 24.5.* Перевантажуються оператори **++** та **– –** для класу Integer з допомогою “дружніх” операторних функцій. “Дружні” операторні функції реалізуються за межами класу.

*// клас Integer, що реалізує ціле число*

*class Integer*

*{*

*private:*

*int number;*

*public:*

*// конструктори класу*

*Integer() { number = 0; }*

*Integer(int \_number) { number = \_number; }*

*// методи доступу*

*int Get(void) { return number; }*

*void Set(int \_number) { number = \_number; }*

*// оголошення "дружніх" операторних функцій*

*friend Integer operator++(Integer & ref);*

*friend Integer operator--(Integer & ref);*

*friend Integer operator++(Integer & ref, int d);*

*friend Integer operator--(Integer & ref, int d);*

*};*

*// реалізація "дружніх" операторних функцій*

*Integer operator++(Integer & ref)*

*{*

*ref.number++; // доступ за посиланням*

*return ref;*

*}*

*Integer operator--(Integer & ref)*

*{*

*ref.number--;*

*return ref;*

*}*

*// постфіксна форма - x++*

*Integer operator++(Integer & ref, int d)*

*{*

*ref.number++;*

*return ref;*

*}*

*// постфіксна форма - x--*

*Integer operator--(Integer & ref, int d)*

*{*

*ref.number--;*

*return ref;*

*}*

Використання класу Integer в іншому методі може бути, наприклад, таким:

*// "дружні" операторні функції, перевантаження операторів ++, --*

*Integer d1, d2(8);*

*int t;*

*// перевірка*

*t = d1.Get(); // t = 0*

*t = d2.Get(); // t = 8*

*// виклик "дружніх" операторних функцій*

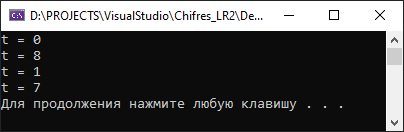
*++d1; // "дружня" префіксна операторна функція operator++()*

*d2--; // "дружня" постфіксна операторна функція operator--()*

*t = d1.Get(); // t = 1*

*t = d2.Get(); // t = 7*

Результат роботи програми наведений нижче:



*Приклад 24.6.* Перевантажуються оператори **++** та — для класу ArrayInt з допомогою “дружніх” операторних функцій. У класі ArrayInt оголошуються внутрішні дані, конструктори, методи доступу та “дружні” до класу операторні функції.

“Дружні” операторні функції перевантажують оператори **++** та **– –**. “Дружні” операторні функції збільшують або зменшують на 1 значення кожного елементу масиву, який оголошений в класі ArrayInt. Реалізація класу ArrayInt та “дружніх” операторних функцій наступна:

*// клас ArrayInt, який реалізує масив цілих чисел*

*class ArrayInt*

*{*

*private:*

*int \*A; // масив цілих чисел, розмір масиву довільний*

*int size; // розмір масиву*

*public:*

*// конструктори класу*

*ArrayInt()*

*{*

*size = 0;*

*A = NULL;*

*}*

*// конструктор з 1 параметром*

*ArrayInt(int \_size)*

*{*

*size = \_size;*

*A = new int[size];*

*for (int i=0; i<size; i++)*

*A[i] = 0;*

*}*

*// методи доступу*

*int GetSize(void)*

*{*

*return size;*

*}*

*void SetSize(int \_size)*

*{*

*// звільнити попередній фрагмент пам'яті*

*if (size>0)*

*delete[] A;*

*// встановити новий розмір масиву*

*size = \_size;*

*A = new int[size];*

*for (int i=0; i<size; i++)*

*A[i] = 0;*

*}*

*int GetAi(int index)*

*{*

*return A[index];*

*}*

*void SetAi(int index, int value)*

*{*

*A[index] = value;*

*}*

*// "дружні" операторні функції ++, --*

*friend ArrayInt operator++(ArrayInt & AI);*

*friend ArrayInt operator--(ArrayInt & AI);*

*friend ArrayInt operator++(ArrayInt & AI, int d);*

*friend ArrayInt operator--(ArrayInt & AI, int d);*

*};*

*// реалізація "дружніх" операторних функцій ++, --*

*// ++x*

*ArrayInt operator++(ArrayInt & AI)*

*{*

*// збільшити кожен елемент масиву на 1*

*for (int i=0; i<AI.size; i++)*

*AI.A[i] = AI.A[i] + 1; // доступ за посиланням*

*return AI;*

*}*

*// --x*

*ArrayInt operator--(ArrayInt & AI)*

*{*

*// зменшити кожен елемент масиву на 1*

*for (int i=0; i<AI.size; i++)*

*AI.A[i] = AI.A[i] - 1; // доступ за посиланням*

*return AI;*

*}*

*// x++*

*ArrayInt operator++(ArrayInt & AI, int d)*

*{*

*// збільшити кожен елемент масиву на 1*

*for (int i=0; i<AI.size; i++)*

*AI.A[i] = AI.A[i] + 1; // доступ за посиланням*

*return AI;*

*}*

*// x--*

*ArrayInt operator--(ArrayInt & AI, int d)*

*{*

*// зменшити кожен елемент масиву на 1*

*for (int i=0; i<AI.size; i++)*

*AI.A[i] = AI.A[i] - 1; // доступ за посиланням*

*return AI;*

*}*

Нижче продемонстровано використання класу ArrayInt та операторних функцій

*// "дружні" операторні функції, перевантаження операторів ++, --*

*ArrayInt A(10);*

*int t;*

*// встановлення довільних значень в масиві*

*for (int i=0; i<A.GetSize(); i++)*

*A.SetAi(i, i+1); // A.A[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 }*

*// перевірка*

*t = A.GetAi(3); // t = 4*

*t = A.GetAi(5); // t = 6*

*// використання перевантажених операторів ++, --*

*A++; // постфіксна функція operator++(ArrayInt, int)*

*t = A.GetAi(3); // t = 5*

*t = A.GetAi(5); // t = 7*

*--A;*

*--A;*

*t = A.GetAi(3); // t = 3*

*t = A.GetAi(5); // t = 5*

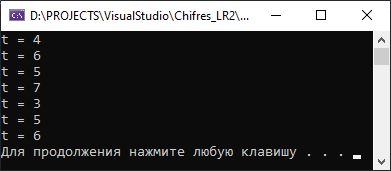
*++A;*

*++A;*

*++A;*

*t = A.GetAi(3); // t = 6*

Результат роботи програми наведений нижче:



***Порядок виконання роботи:***

1. Розглянути приклади, наведені у теоретичних відомостях.

**Лабораторна робота №25**

**Перевантаження скорочених операторів присвоювання +=, -=, \*=, /=, %=**

*Мета роботи*: набуття навичок роботи з перевантаженими операторами.

*Теоретичні основи*:

Скорочені оператори присвоювання +=, -=, \*=, /=, %= є бінарними, тобто вони вимагають двох операндів. Якщо операторна функція реалізована в класі, то вона отримує один операнд і має наступну загальну форму

*class ClassName*

*{*

*// ...*

*// операторна функція, яка перевантажує один зі скорочених операторів присвоювання*

*ClassName operator##(ClassName obj)*

*{*

*// ...*

*}*

*}*

де

* ClassName – ім’я класу, для якого реалізується операторна функція operator##();
* operator##() – деяка операторна функція. Символи ## замінюються символами +=, -=, \*=, /=, %=, тобто, операторна функція буде мати відповідно такі імена: operator+=(), operator-=(), operator\*=(), operator/=(), operator%=().
* Obj – екземпляр (об’єкт) класу ClassName.

Якщо операторна функція реалізована як “дружня”, то вона отримує два операнди. У цьому випадку загальна форма використання операторної функції має вигляд:

*class ClassName*

*{*

*// ...*

*// "дружня" операторна функція,*

*// яка перевантажує один зі скорочених операторів присвоювання*

*friend ClassName operator##(ClassName& obj1, ClassName& obj2)*

*{*

*// ...*

*}*

*}*

*// реалізація "дружньої" до класу ClassName операторної функції*

*ClassName operator##(ClassName& obj1, ClassName& obj2)*

*{*

*// ...*

*}*

тут obj1, obj2 – посилання на екземпляри класу ClassName, які передаються в “дружню” операторну функцію operator##().

*Приклад 25.1.* Задано клас Integer, який реалізує операції над цілочисельним значенням. У класі оголошуються:

* внутрішня змінна number цілого типу, яка зберігає значення;
* два конструктори;
* методи доступу до цілочисельної змінної number;
* операторна функція operator+=(), яка перевантажує скорочений оператор присвоювання **+=**.

Реалізація класу Integer

*// клас Integer - реалізує ціле число*

*class Integer*

*{*

*private:*

*int number;*

*public:*

*// конструктори класу*

*Integer() { number = 0; }*

*Integer(int \_number) { number = \_number; }*

*// методи доступу*

*int Get(void) { return number; }*

*void Set(int \_number) { number = \_number; }*

*// перевантаження оператора +=, операторна функція всередині класу*

*Integer operator+=(Integer obj)*

*{*

*number = number + obj.number; // доступ за посиланням*

*return \*this;*

*}*

*};*

Використання екземпляру (об’єкту) класу Integer в іншому програмному коді може бути, наприклад, таким:

*// перевантаження скорочених операторів присвоюванн +=, -= і т.д.*

*Integer d1(10), d2(18); // екземпляри класу Integer*

*int t;*

*// виклик операторної функції operator+=()*

*d1+=d2; // d1.number = d1.number + d2.number*

*t = d1.Get(); // t = 28*

За даним зразком можна перевантажувати інші скорочені оператори присвоювання -=, \*=, /=, %=.

*Приклад 25.2.* У класі Integer оголошується “дружня” (friend) операторна функція operator%=(), яка визначає об’єкт класу Integer, який є остачею від ділення об’єктів, що є параметрами функції.

*// клас Integer, що реалізує ціле число*

*class Integer*

*{*

*private:*

*int number;*

*public:*

*// конструктори класу*

*Integer() { number = 0; }*

*Integer(int \_number) { number = \_number; }*

*// методи доступу*

*int Get(void) { return number; }*

*void Set(int \_number) { number = \_number; }*

*// оголошення "дружньої" до класу Integer операторної функції*

*// яка перевантажує оператор %=*

*friend Integer operator%=(Integer& obj1, Integer& obj2);*

*};*

*// "дружня" операторна функція до класу Integer*

*// в операторну функцію передаються посилання на клас Integer*

*Integer operator%=(Integer& obj1, Integer& obj2)*

*{*

*obj1.number = obj1.number % obj2.number; // доступ за посиланням*

*return obj1;*

*}*

Використання операторної функції operator %=() може бути таким

*// перевантаження скорочених операторів присвоюванн +=, -= і т.д.*

*Integer d1(14), d2(6); // екземпляри класу Integer*

*int t;*

*// виклик "дружньої" операторної функції operator%=()*

*d1 %= d2; // d.number = d1.number + d2.number*

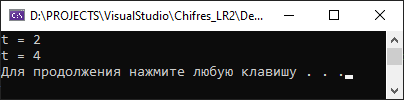
*t = d1.Get(); // t = 2*

*Integer d3(9), d4(5);*

*d3 %= d4;*

*t = d3.Get(); // t = 4*

Результат роботи програми наведений нижче:



*Приклад 25.3.* У класі Complex реалізується операторна функція, яка перевантажеує оператор \*=. Ця функція реалізує множення комплексних чисел за формулою:

(a+bj)\*(c+dj) = (a\*c-b\*d) + (a\*d+b\*c)j

Реалізація класу Complex

*class Complex*

*{*

*private:*

*double real; // дійсна частина*

*double imag; // уявна частина*

*public:*

*// конструктор класу*

*Complex(double \_real, double \_imag)*

*{*

*real = \_real;*

*imag = \_imag;*

*}*

*// методи доступу*

*double GetReal(void) { return real; }*

*double GetImag(void) { return imag; }*

*void Set(double \_real, double \_imag)*

*{*

*real = \_real;*

*imag = \_imag;*

*}*

*// операторна функція, яка перевантажує оператор \*=*

*// функція реалізує множення комплексних чисел*

*Complex operator\*=(Complex obj)*

*{*

*double r;*

*double i;*

*r = real\*obj.real - imag\*obj.imag;*

*i = real\*obj.imag + obj.real\*imag;*

*real = r;*

*imag = i;*

*return \*this;*

*}*

*};*

Демонстрація використання операторної функції

*// перевантаження скороченого оператора \*=*

*Complex c1(2,3); // 2+3j*

*Complex c2(-1,1); // -1+j*

*double t;*

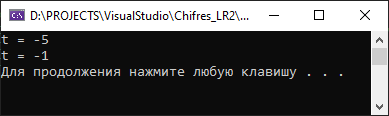
*// виклик операторної функції \*=*

*c1 \*= c2; // c1 = -5-1j*

*t = c1.GetReal(); // t = -5*

*t = c1.GetImag(); // t = -1*

Результат роботи програми наведений нижче:



*Приклад 25.4.* У прикладі реалізується операція множення комплексних чисел з допомогою “дружньої” (friend) операторної функції operator\*=(). “Дружня” операторна функція отримує 2 параметри.

*class Complex*

*{*

*private:*

*double real; // дійсна частина*

*double imag; // уявна частина*

*public:*

*// конструктор класу*

*Complex(double \_real, double \_imag)*

*{*

*real = \_real;*

*imag = \_imag;*

*}*

*// методи доступу*

*double GetReal(void) { return real; }*

*double GetImag(void) { return imag; }*

*void Set(double \_real, double \_imag)*

*{*

*real = \_real;*

*imag = \_imag;*

*}*

*// оголошення "дружньої" функції*

*friend Complex operator\*=(Complex& c1, Complex& c2);*

*};*

*// реалізація "дружньої" до класу Complex функції*

*Complex operator\*=(Complex& c1, Complex& c2)*

*{*

*double r;*

*double i;*

*r = c1.real \* c2.real - c1.imag \* c2.imag;*

*i = c1.real \* c2.imag + c1.imag \* c2.real;*

*c1.real = r;*

*c1.imag = i;*

*return c1;*

*}*

Використання класу Complex в іншому методі може бути таким:

*// перевантаження скороченого оператора \*=*

*Complex c1(3,-2); // 3-2j*

*Complex c2(1,-1); // 1-j*

*double t;*

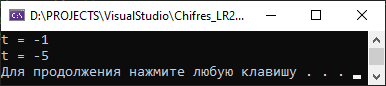
*// виклик операторної функції \*=*

*c1 \*= c2;*

*t = c1.GetReal(); // t = -1*

*t = c1.GetImag(); // t = -5*

Результат роботи програми наведений нижче:



*Приклад 25.5.* У класі ArrayDouble реалізується масив чисел типу double. Операторна функція operator/=() реалізує поелементне ділення двох масивів

*// фіксований масив типу double*

*class ArrayDouble*

*{*

*private:*

*double D[10]; // масив*

*public:*

*// конструктор*

*ArrayDouble()*

*{*

*for (int i=0; i<10; i++)*

*D[i] = 0;*

*}*

*// методи доступу*

*double Get(int index)*

*{*

*if ((index>=0)&&(index<10))*

*return D[index];*

*return 0.0;*

*}*

*void Set(int index, double value)*

*{*

*if ((index>=0)&&(index<10))*

*D[index] = value;*

*}*

*// операторна функція, поелементно ділить два масиви*

*ArrayDouble operator/=(ArrayDouble AD)*

*{*

*// поелементне ділення поточного масиву на масив AD*

*for (int i=0; i<10; i++)*

*D[i] = D[i] / AD.Get(i);*

*return \*this;*

*}*

*};*

Використання класу ArrayDoble може бути таким

*// перевантаження скороченого оператора \*=*

*ArrayDouble A1;*

*ArrayDouble A2;*

*double t;*

*// формування масивів*

*for (int i=0; i<10; i++)*

*A1.Set(i,10); // A1 = {10,10,10,10,10,10,10,10,10,10}*

*for (int i=0; i<10; i++) // A2 = {1,2,3,4,5,6,7,8,9,10}*

*A2.Set(i, i+1);*

*// виклик операторної функції*

*A1 /= A2;*

*// перевірка*

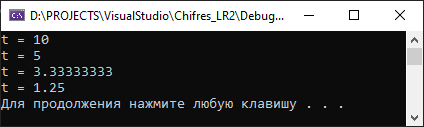
*t = A1.Get(0); // t = 10/1 = 10.0*

*t = A1.Get(1); // t = 10/2 = 5.0*

*t = A1.Get(2); // t = 10/3 = 3.33333333*

*t = A1.Get(7); // t = 10/8 = 1.25*

Результат роботи програми наведений нижче:



*Приклад 25.6.* У класі ArrayDouble оголошується “дружня” операторна функція, яка реалізує поелементне ділення масивів. Оголошення класу ArrayDouble та операторної функції наступне

*// фіксований масив типу double*

*class ArrayDouble*

*{*

*private:*

*double D[10]; // масив*

*public:*

*// конструктор*

*ArrayDouble()*

*{*

*for (int i=0; i<10; i++)*

*D[i] = 0;*

*}*

*// методи доступу*

*double Get(int index)*

*{*

*if ((index>=0)&&(index<10))*

*return D[index];*

*return 0.0;*

*}*

*void Set(int index, double value)*

*{*

*if ((index>=0)&&(index<10))*

*D[index] = value;*

*}*

*// оголошення "дружньої" операторної функції*

*friend ArrayDouble operator/=(ArrayDouble& AD1, ArrayDouble& AD2);*

*};*

*// реалізація "дружньої" до класу ArrayDouble операторної функції*

*ArrayDouble operator/=(ArrayDouble& AD1, ArrayDouble& AD2)*

*{*

*for (int i=0; i<10; i++)*

*AD1.D[i] = AD1.D[i] / AD2.D[i];*

*return AD1;*

*}*

Використання класу в іншому програмному коді

*// перевантаження скороченого оператора \*=*

*ArrayDouble A1;*

*ArrayDouble A2;*

*double t;*

*// формування масивів*

*for (int i=0; i<10; i++)*

*A1.Set(i,10); // A1 = {10,10,10,10,10,10,10,10,10,10}*

*for (int i=0; i<10; i++) // A2 = {1,2,3,4,5,6,7,8,9,10}*

*A2.Set(i, i+1);*

*// виклик "дружньої" операторної функції*

*A1 /= A2;*

*// перевірка*

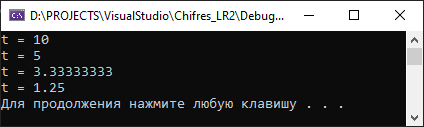
*t = A1.Get(0); // t = 10/1 = 10.0*

*t = A1.Get(1); // t = 10/2 = 5.0*

*t = A1.Get(2); // t = 10/3 = 3.33333333*

*t = A1.Get(7); // t = 10/8 = 1.25*

Результат роботи програми наведений нижче:



*Приклад 25.7.* Операторні функції в класі можуть бути перевантажені. Такі функції мають містити однакову кількість параметрів, але ці параметри мають мати різні типи.

*// Клас, що реалізує точку на координатній площині*

*// клас містить дві перевантажені операторні функції operator+=()*

*class Point*

*{*

*private:*

*int x, y; // координати точки*

*public:*

*// конструктори класу*

*Point()*

*{*

*x = y = 0;*

*}*

*Point(int nx, int ny)*

*{*

*x = nx;*

*y = ny;*

*}*

*// методи доступу до членів класу*

*int GetX(void) { return x; }*

*int GetY(void) { return y; }*

*void SetX(int nx) { x = nx; }*

*void SetY(int ny) { y = ny; }*

*// операторна функція '+='*

*// яка додає 2 точки*

*Point operator+=(Point pt)*

*{*

*x = x + pt.x;*

*y = y + pt.y;*

*return \*this;*

*}*

*// операторна функція '+='*

*// яка збільшує x та y на величину вхідного параметра d*

*Point operator+=(int d)*

*{*

*x = x + d;*

*y = y + d;*

*return \*this;*

*}*

*};*

У вищенаведеному коді реалізовано дві операторні функції operator+=(), які перевантажують оператор **+=**.

*// операторна функція '+='*

*// яка додає 2 точки*

*Point operator+=(Point pt)*

*{*

*// ...*

*}*

*// операторна функція '+='*

*// яка збільшує x та y на величину вхідного параметра d*

*Point operator+=(int d)*

*{*

*// ...*

*}*

Нижче продемонстровано використання операторних функцій класу Point.

*// перевантаження скороченого оператора +=*

*Point p1(2,3);*

*Point p2(4,5);*

*int x, y;*

*// виклик операторної функції додавання екземплярів типу Point*

*p1 += p2;*

*// перевірка*

*x = p1.GetX(); // x = 2+4 = 6*

*y = p1.GetY(); // y = 3+5 = 8*

*p2 += p1;*

*x = p2.GetX(); // x = 6+4 = 10*

*y = p2.GetY(); // y = 8+5 = 13*

*// виклик операторної функції додавання Point + int*

*p1 += 10;*

*x = p1.GetX(); // x = 6+10 = 16*

*y = p1.GetY(); // y = 8+10 = 18*

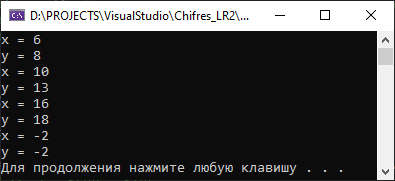
*Point p3(5,5);*

*p3+=-7;*

*x = p3.GetX(); // x = -2*

*y = p3.GetY(); // y = -2*

Результат роботи програми наведений нижче:



***Порядок виконання роботи:***

Розглянути приклади, наведені у теоретичних відомостях.

**Лабораторна робота №26**

**Перевантаження скорочених операторів присвоювання +=, -=, \*=, /=, %=**

*Мета роботи*: набуття навичок роботи з перевантаженими операторами.

*Теоретичні основи*:

Оператор new потрібно перевантажувати у випадках, коли пам’ять виділяється по особливому (не стандартним способом). Відповідно, оператор delete повинен звільняти цю пам’ять нестандартним способом. Як правило, якщо в класі перевантажується оператор new, то в цьому класі також перевантажується оператор delete.

Оператори new та delete можуть бути перевантажені двома способами. Ці способи відрізняються оголошенням операторної функції:

* перевантаження операторів new та delete з допомогою операторної функції, яка реалізована в межах класу. Це є спосіб перевантаження new та delete для конкретного класу. При виклику new та delete для цього класу викликаються перевантажені операторні функції, які виділяють/звільняють пам’ять специфічним чином. При виклику операторів new та delete для інших класів використовуються глобальні оператори new та delete;
* глобальне перевантаження операторів new та delete. У цьому випадку оператори new та delete перевантажуються для вбудованих типів (float, int і т.д.).

Загальна форма перевантаження оператора new має такий вигляд:

*return\_type \* operator new (size\_t size)*

*{*

*// Виділення пам'яті.*

*//*

*}*

тут

* return\_type – тип (клас), на який операторна функція повертає покажчик для якого виділена пам’ять специфічним способом;
* size – розмір пам’яті, який виділяється для типу return\_type. Кількість байт пам’яті, що виділяється, не обов’язково повинна співпадати зі значенням size, тому що в тілі операторної функції пам’ять можна виділяти по особливому.

Загальна форма перевантаження оператора delete має наступний вигляд:

*void operator delete(void \* pointer)  
{  
 // звільнення пам'яті, на яку вказує покажчик pointer  
 // ...  
}*

тут pointer – покажчик, який вказує на область пам’яті, яка була попередньо виділена оператором new.

Якщо у програмі перевантажуються оператори new та delete, то виникає два види цих операторів:

* перевантажені оператори new та delete. Ці перевантажені оператори застосовуються для конкретного класу (класів), в якому є операторні функції operator new() та operator delete();
* глобальні оператори new та delete (не перевантажені). Ці оператори застосовуються для класів, які не містять операторних функцій, що перевантажують оператори new та delete.

Для кожного класу компілятор визначає, який варіант операторів new та delete потрібно вибрати: глобальний чи для конкретного класу.

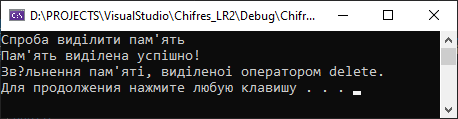
*Приклад 26.1.* У прикладі оголошується клас Complex, який реалізує комплексне число. Клас містить:

* внутрішні змінні real, imag, які є дійсною та уявною частиною комплексного числа;
* конструктор класу;
* методи доступу Get(), Set();
* перевантажені операторні функції operator new(), operator delete() які перевантажують відповідно оператори new та delete. Ці функції використовуються строго для класу Complex.

Реалізація класу Complex та функції main(), яка демонструє перевантаження операторів new та delete має наступний вигляд:

*#include "stdafx.h"  
#include <iostream>  
#include <cstdlib>  
#include <new>  
using namespace std;  
  
// клас, що містить перевантажені оператори new та delete  
class Complex  
{  
private:  
 double real;  
 double imag;  
  
public:  
 // конструктор  
 Complex(double \_real, double \_imag)  
 {  
 real = \_real;  
 imag = \_imag;  
 }  
  
 // методи доступу  
 void Get(double \* r, double \* i)  
 {  
 \*r = real;  
 \*i = imag;  
 return;  
 }  
  
 void Set(double r, double i)  
 {  
 real = r;  
 imag = i;  
 }  
  
 // перевантажений оператор new,  
 // виділяє пам'ять заданого розміру  
 void \* operator new(size\_t size)   
 {  
 void \* ptr;  
  
 cout<<"Спроба виділити пам'ять\n";  
 ptr = malloc(size); // спроба виділення пам'яті  
  
 if (!ptr) // якщо пам'ять не виділена, то згенерувати виключну ситуацію  
 {  
 bad\_alloc ba;  
 cout<<"Помилка виділення пам\'яті.\n";  
 throw ba;  
 }  
 else  
 {  
 //   
 cout<<"Пам'ять виділена успішно!\n";  
 return ptr;  
 }  
 }  
  
 // звільняє пам'ять, виділену перевантаженою   
 // операторною функцією operator new()  
 void operator delete(void \* ptr)  
 {  
 cout<<"Звільнення пам'яті, виділеної оператором delete.\n";  
 free(ptr);  
 }  
};  
  
int \_tmain(int argc, \_TCHAR\* argv[])  
{  
 // демонстрація використання операторів new та delete в класі Complex  
 Complex \* p1; // оголосити покажчик на Complex  
  
 try   
 {  
 // виклик операторної функції operator new()  
 p1 = new Complex(5,6); // виділити пам'ять  
 }  
 catch (bad\_alloc error)  
 {  
 return 1; // вийти з програми з кодом 1  
 }  
  
 // звільнити пам'ять  
 // виклик операторної функції operator delete()  
 delete p1;  
  
 return 0;  
}*

Результат роботи програми наведений нижче:



У результаті виконання вищенаведеного коду, на екран буде виведено наступний результат:

Спроба виділити пам'ять Пам'ять виділена успішно! Звільнення пам'яті, виділеної оператором delete.

*Приклад 26.2.* Оператори new та delete можуть бути перевантажені глобально. У цьому випадку використання глобальних перевантажених операторів new та delete може застосовуватись для будь-яких типів.

Нижче наведено приклад глобального перевантаження операторів new та delete

#include <iostream>

#include <cstdlib>

#include <math.h>

#include <cstdlib>

#include <new>

using namespace std;

// клас, що мiстить перевантаженi оператори new та delete

class Complex

{

private:

double real;

double imag;

public:

// конструктор

Complex(double \_real, double \_imag)

{

real = \_real;

imag = \_imag;

}

// методи доступу

void Get(double\* r, double\* i)

{

\*r = real;

\*i = imag;

return;

}

void Set(double r, double i)

{

real = r;

imag = i;

}

};

// глобальний оператор new

void\* operator new(size\_t size)

{

void\* ptr;

cout << "Глобальний оператор new" << ::endl;

cout << "1. Спроба видiлити пам'ять функцiєю malloc()\n";

ptr = malloc(size); // спроба видiлення пам'ятi

if (!ptr) // якщо пам'ять не видiлена, то згенерувати виключну ситуацiю

{

bad\_alloc ba;

cout << "2. Помилка видiлення пам'ятi.\n";

throw ba;

}

else

{

//

cout << "3. Пам'ять видiлена успiшно глобальним оператором new\n";

return ptr;

}

}

// глобальний оператор delete

void operator delete(void\* ptr)

{

cout << "4. Глобальний оператор delete.\n";

free(ptr);

}

int main()

{

setlocale(LC\_ALL, "Russian");

// демонстрацiя використання глобальних операторiв new та delete

Complex\* pC; // оголосити покажчик на Complex

int\* pI; // покажчик на int

try

{

// виклик операторної функцiї operator new()

pC = new Complex(5, 6); // видiлити пам'ять

}

catch (bad\_alloc error)

{

cout << "Помилка при видiленнi пам'ятi для об'єкту pC" << ::endl;

return 1; // вийти з програми з кодом 1

}

// звiльнити пам'ять

// виклик глобального оператора delete()

delete pC;

try

{

// виклик перевантаженої версiї оператора new

cout << "Спроба видiлення пам'ятi для об'єкту pI." << ::endl;

pI = new int;

}

catch (bad\_alloc error)

{

cout << "Помилка при видiленнi пам'ятi для об'єкту pI" << ::endl;

return 2; // вийти з програми з кодом 2

}

delete pI;

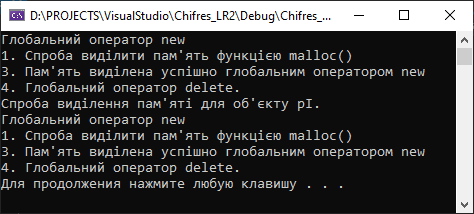
//cout << "x = 6\ny = 8\nx = 10\ny = 13\nx = 16\ny = 18\nx = -2\ny = -2\n";

system("pause");

return 0;

}

Результат роботи програми наведений нижче:



У результаті виконання вищенаведеного коду на екран буде виведено наступний результат:

Глобальний оператор new  
1. Спроба виділити пам'ять функцією malloc()  
3. Пам'ять виділена успішно глобальним оператором new  
4. Глобальний оператор delete.  
Спроба виділення пам'яті для об'єкту pI.  
1. Спроба виділити пам'ять функцією malloc()  
3. Пам'ять виділена успішно глобальним оператором new  
4. Глобальний оператор delete.

Як видно з результату, глобальні оператори new та delete працюють як для базових типів, так і для класів.

*Приклад 26.3.* Наведено приклад перевантаження операторів new та delete для масиву об’єктів типу Point. Клас Point визначає точку на координатній площині і містить наступні оголошення:

* змінні x, y які є координатами точки;
* конструктори;
* методи доступу Get(), Set();
* операторні функції operator new() та operator delete(), які реалізують виділення/звільнення пам’яті для одиночних об’єктів типу Point;
* операторні функції operator new[]() та operator delete[](), які реалізують виділення/звільнення пам’яті для масивів об’єктів типу Point.

Програмний код, що реалізує клас Point, має наступний вигляд:

#include <iostream>

#include <cstdlib>

#include <math.h>

#include <cstdlib>

#include <new>

using namespace std;

class Point

{

private:

double x, y;

public:

// конструктори

Point()

{

x = y = 0.0;

}

Point(double \_x, double \_y)

{

x = \_x;

y = \_y;

}

// методи доступу

void Get(double\* \_x, double\* \_y)

{

\*\_x = x;

\*\_y = y;

}

void Set(double \_x, double \_y)

{

x = \_x;

y = \_y;

}

// перевантажений оператор new для одиночного об'єкту класу Point

void\* operator new(size\_t size)

{

void\* ptr;

ptr = malloc(size);

if (!ptr)

{

bad\_alloc error;

throw error;

}

return ptr;

}

// перевантажений оператор delete для одиночного об'єкту класу Point

void operator delete(void\* ptr)

{

cout << "Звiльнення пам'ятi оператором delete" << ::endl;

free(ptr);

}

// перевантажений оператор new для масиву об'єктiв типу Point

void\* operator new[](size\_t size)

{

void\* ptr;

cout << "Застосування перевантаженого оператора new[]." << ::endl;

ptr = malloc(size);

if (!ptr)

{

bad\_alloc error;

throw error;

}

return ptr;

}

// перевантажений оператор delete для масиву об'єктiв типу Point

void operator delete[](void\* ptr)

{

cout << "Видалення масиву з пам'ятi оператором delete[]" << ::endl;

free(ptr);

}

};

int main()

{

setlocale(LC\_ALL, "Russian");

// перевантаження операторiв new та delete для масивiв об'єктiв

Point\* p1;

try

{

// виклик перевантаженої операторної функцiї operator new[]()

// класу Point

p1 = new Point[5]();

}

catch (bad\_alloc error)

{

cout << "Помилка при видiленнi пам'ятi для об'єкту p1." << ::endl;

return 1;

}

delete[] p1;

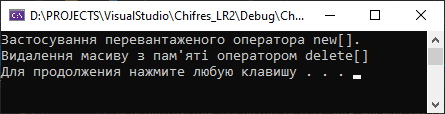
//cout << "x = 6\ny = 8\nx = 10\ny = 13\nx = 16\ny = 18\nx = -2\ny = -2\n";

system("pause");

return 0;

}

Результат роботи програми наведений нижче:



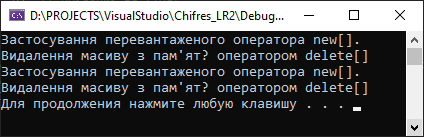
У результаті виконання вищенаведеного коду на екран буде виведено наступний результат

Застосування перевантаженого оператора new[].  
Видалення масиву з пам'яті оператором delete[]

*Приклад 26.4.* Нижче наведено приклад глобального перевантаження операторів new[] та delete[] для масивів класу Point

*#include "stdafx.h"  
#include <iostream>  
#include <cstdlib>  
using namespace std;  
  
class Point  
{  
private:  
 double x, y;  
  
public:  
 // конструктори  
 Point()  
 {  
 x = y = 0.0;  
 }  
  
 Point(double \_x, double \_y)  
 {  
 x = \_x;  
 y = \_y;  
 }  
  
 // методи доступу  
 void Get(double \* \_x, double \* \_y)  
 {  
 \*\_x = x;  
 \*\_y = y;  
 }  
  
 void Set(double \_x, double \_y)  
 {  
 x = \_x;  
 y = \_y;  
 }  
};  
  
// перевантажений оператор new для одиночного об'єкту класу Point  
void \* operator new(size\_t size)  
{  
 void \* ptr;  
 ptr = malloc(size);  
 if (!ptr)  
 {  
 bad\_alloc error;  
 throw error;  
 }  
 return ptr;  
}  
  
// перевантажений оператор delete для одиночного об'єкту класу Point  
void operator delete(void \* ptr)  
{  
 cout<<"Звільнення пам'яті оператором delete"<<::endl;  
 free(ptr);  
}  
  
// перевантажений оператор new для масиву об'єктів типу Point  
void \* operator new[](size\_t size)  
{  
 void \* ptr;  
 cout<<"Застосування перевантаженого оператора new[]."<<::endl;  
 ptr = malloc(size);  
 if (!ptr)  
 {  
 bad\_alloc error;  
 throw error;  
 }  
 return ptr;  
}  
  
// перевантажений оператор delete для масиву об'єктів типу Point  
void operator delete[](void \* ptr)  
{  
 cout << "Видалення масиву з пам'яті оператором delete[]"<<::endl;  
 free(ptr);  
}  
  
int \_tmain(int argc, \_TCHAR\* argv[])  
{  
 // глобальне перевантаження операторів new та delete для масивів об'єктів  
 Point \* p1; // покажчик на клас Point  
 float \* p2; // покажчик на float  
  
 try   
 {  
 // виклик перевантаженої глобальної операторної функції operator new[]()   
 // класу Point  
 p1 = new Point[5]();  
 }  
 catch (bad\_alloc error)  
 {  
 cout<<"Помилка при виділенні пам'яті для об'єкту p1."<<::endl;  
 return 1;  
 }  
 delete[] p1;  
  
 // виклик глобальної операторної функції operator new[]() для базового типу float  
 try  
 {  
 p2 = new float[10]; // виклик функції   
 }  
 catch (bad\_alloc error)  
 {  
 cout<<"Помилка при виділенні пам'яті для об'єкту p2."<<::endl;  
 return 2;  
 }  
 delete[] p2;  
 return 0;  
}*

Результат роботи програми наведений нижче:



У результаті виконання вищенаведеного коду буде виведено наступний результат:

Застосування перевантаженого оператора new[].  
Видалення масиву з пам'яті оператором delete[].

Як видно з результату, перевантажені глобальні операторні функції operator new[] та operator delete[] для масивів працюють однаково як з базовими типами (float) так і з типами-класами (Point).

***Порядок виконання роботи:***

Розглянути приклади, наведені у теоретичних відомостях.

**Лабораторна робота №27**

# **Перевантаження операторів new та delete**

*Мета роботи*: набуття навичок роботи з перевантаженими операторами.

*Теоретичні основи*:

Оператор new потрібно перевантажувати у випадках, коли пам’ять виділяється по особливому (не стандартним способом). Відповідно, оператор delete повинен звільняти цю пам’ять нестандартним способом. Як правило, якщо в класі перевантажується оператор new, то в цьому класі також перевантажується оператор delete.

Оператори new та delete можуть бути перевантажені двома способами. Ці способи відрізняються оголошенням операторної функції:

* перевантаження операторів new та delete з допомогою операторної функції, яка реалізована в межах класу. Це є спосіб перевантаження new та delete для конкретного класу. При виклику new та delete для цього класу викликаються перевантажені операторні функції, які виділяють/звільняють пам’ять специфічним чином. При виклику операторів new та delete для інших класів використовуються глобальні оператори new та delete;
* глобальне перевантаження операторів new та delete. У цьому випадку оператори new та delete перевантажуються для вбудованих типів (float, int і т.д.).

Загальна форма перевантаження оператора new має такий вигляд:

*return\_type \* operator new(size\_t size)  
{  
 // Виділення пам'яті.  
 // ...  
}*

тут

* return\_type – тип (клас), на який операторна функція повертає покажчик для якого виділена пам’ять специфічним способом;
* size – розмір пам’яті, який виділяється для типу return\_type. Кількість байт пам’яті, що виділяється, не обов’язково повинна співпадати зі значенням size, тому що в тілі операторної функції пам’ять можна виділяти по особливому.

Загальна форма перевантаження оператора delete має наступний вигляд:

*void operator delete(void \* pointer)  
{  
 // звільнення пам'яті, на яку вказує покажчик pointer  
 // ...  
}*

тут pointer – покажчик, який вказує на область пам’яті, яка була попередньо виділена оператором new.

Якщо у програмі перевантажуються оператори new та delete, то виникає два види цих операторів:

* перевантажені оператори new та delete. Ці перевантажені оператори застосовуються для конкретного класу (класів), в якому є операторні функції operator new() та operator delete();
* глобальні оператори new та delete (не перевантажені). Ці оператори застосовуються для класів, які не містять операторних функцій, що перевантажують оператори new та delete.

Для кожного класу компілятор визначає, який варіант операторів new та delete потрібно вибрати: глобальний чи для конкретного класу.

*Приклад 27.1.* У прикладі оголошується клас Complex, який реалізує комплексне число. Клас містить:

* внутрішні змінні real, imag, які є дійсною та уявною частиною комплексного числа;
* конструктор класу;
* методи доступу Get(), Set();
* перевантажені операторні функції operator new(), operator delete() які перевантажують відповідно оператори new та delete. Ці функції використовуються строго для класу Complex.

Реалізація класу Complex та функції main(), яка демонструє перевантаження операторів new та delete має наступний вигляд:

#include <iostream>

#include <cstdlib>

#include <math.h>

#include <cstdlib>

#include <new>

using namespace std;

// клас, що мiстить перевантаженi оператори new та delete

class Complex

{

private:

double real;

double imag;

public:

// конструктор

Complex(double \_real, double \_imag)

{

real = \_real;

imag = \_imag;

}

// методи доступу

void Get(double\* r, double\* i)

{

\*r = real;

\*i = imag;

return;

}

void Set(double r, double i)

{

real = r;

imag = i;

}

// перевантажений оператор new,

// видiляє пам'ять заданого розмiру

void\* operator new(size\_t size)

{

void\* ptr;

cout << "Спроба видiлити пам'ять\n";

ptr = malloc(size); // спроба видiлення пам'ятi

if (!ptr) // якщо пам'ять не видiлена, то згенерувати виключну ситуацiю

{

bad\_alloc ba;

cout << "Помилка видiлення пам\'ятi.\n";

throw ba;

}

else

{

//

cout << "Пам'ять видiлена успiшно!\n";

return ptr;

}

}

// звiльняє пам'ять, видiлену перевантаженою

// операторною функцiєю operator new()

void operator delete(void\* ptr)

{

cout << "Звiльнення пам'ятi, видiленої оператором delete.\n";

free(ptr);

}

};

int main()

{

setlocale(LC\_ALL, "Russian");

// демонстрацiя використання операторiв new та delete в класi Complex

Complex\* p1; // оголосити покажчик на Complex

try

{

// виклик операторної функцiї operator new()

p1 = new Complex(5, 6); // видiлити пам'ять

}

catch (bad\_alloc error)

{

return 1; // вийти з програми з кодом 1

}

// звiльнити пам'ять

// виклик операторної функцiї operator delete()

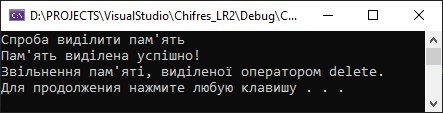
delete p1;

system("pause");

return 0;

}

Результат роботи програми наведений нижче:



У результаті виконання вищенаведеного коду, на екран буде виведено наступний результат:

Спроба виділити пам'ять.

Пам'ять виділена успішно!

Звільнення пам'яті, виділеної оператором delete.

*Приклад 27.2.* Оператори new та delete можуть бути перевантажені глобально. У цьому випадку використання глобальних перевантажених операторів new та delete може застосовуватись для будь-яких типів.

Нижче наведено приклад глобального перевантаження операторів new та delete

#include <iostream>

#include <cstdlib>

#include <math.h>

#include <cstdlib>

#include <new>

using namespace std;

// клас, що мiстить перевантаженi оператори new та delete

class Complex

{

private:

double real;

double imag;

public:

// конструктор

Complex(double \_real, double \_imag)

{

real = \_real;

imag = \_imag;

}

// методи доступу

void Get(double\* r, double\* i)

{

\*r = real;

\*i = imag;

return;

}

void Set(double r, double i)

{

real = r;

imag = i;

}

};

// глобальний оператор new

void\* operator new(size\_t size)

{

void\* ptr;

cout << "Глобальний оператор new" << ::endl;

cout << "1. Спроба видiлити пам'ять функцiєю malloc()\n";

ptr = malloc(size); // спроба видiлення пам'ятi

if (!ptr) // якщо пам'ять не видiлена, то згенерувати виключну ситуацiю

{

bad\_alloc ba;

cout << "2. Помилка видiлення пам'ятi.\n";

throw ba;

}

else

{

//

cout << "3. Пам'ять видiлена успiшно глобальним оператором new\n";

return ptr;

}

}

// глобальний оператор delete

void operator delete(void\* ptr)

{

cout << "4. Глобальний оператор delete.\n";

free(ptr);

}

int main()

{

setlocale(LC\_ALL, "Russian");

// демонстрацiя використання глобальних операторiв new та delete

Complex\* pC; // оголосити покажчик на Complex

int\* pI; // покажчик на int

try

{

// виклик операторної функцiї operator new()

pC = new Complex(5, 6); // видiлити пам'ять

}

catch (bad\_alloc error)

{

cout << "Помилка при видiленнi пам'ятi для об'єкту pC" << ::endl;

return 1; // вийти з програми з кодом 1

}

// звiльнити пам'ять

// виклик глобального оператора delete()

delete pC;

try

{

// виклик перевантаженої версiї оператора new

cout << "Спроба видiлення пам'ятi для об'єкту pI." << ::endl;

pI = new int;

}

catch (bad\_alloc error)

{

cout << "Помилка при видiленнi пам'ятi для об'єкту pI" << ::endl;

return 2; // вийти з програми з кодом 2

}

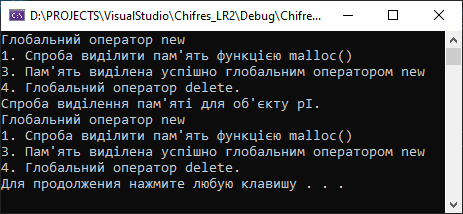
delete pI;

system("pause");

return 0;

}

Результат роботи програми наведений нижче:



У результаті виконання вищенаведеного коду на екран буде виведено наступний результат:

Глобальний оператор new  
1. Спроба виділити пам'ять функцією malloc()  
3. Пам'ять виділена успішно глобальним оператором new  
4. Глобальний оператор delete.  
Спроба виділення пам'яті для об'єкту pI.  
1. Спроба виділити пам'ять функцією malloc()  
3. Пам'ять виділена успішно глобальним оператором new  
4. Глобальний оператор delete.

Як видно з результату, глобальні оператори new та delete працюють як для базових типів, так і для класів.

*Приклад 27.3.* Наведено приклад перевантаження операторів new та delete для масиву об’єктів типу Point. Клас Point визначає точку на координатній площині і містить наступні оголошення:

* змінні x, y які є координатами точки;
* конструктори;
* методи доступу Get(), Set();
* операторні функції operator new() та operator delete(), які реалізують виділення/звільнення пам’яті для одиночних об’єктів типу Point;
* операторні функції operator new[]() та operator delete[](), які реалізують виділення/звільнення пам’яті для масивів об’єктів типу Point.

Програмний код, що реалізує клас Point, має наступний вигляд:

#include <iostream>

#include <cstdlib>

#include <math.h>

#include <cstdlib>

#include <new>

using namespace std;

class Point

{

private:

double x, y;

public:

// конструктори

Point()

{

x = y = 0.0;

}

Point(double \_x, double \_y)

{

x = \_x;

y = \_y;

}

// методи доступу

void Get(double\* \_x, double\* \_y)

{

\*\_x = x;

\*\_y = y;

}

void Set(double \_x, double \_y)

{

x = \_x;

y = \_y;

}

// перевантажений оператор new для одиночного об'єкту класу Point

void\* operator new(size\_t size)

{

void\* ptr;

ptr = malloc(size);

if (!ptr)

{

bad\_alloc error;

throw error;

}

return ptr;

}

// перевантажений оператор delete для одиночного об'єкту класу Point

void operator delete(void\* ptr)

{

cout << "Звiльнення пам'ятi оператором delete" << ::endl;

free(ptr);

}

// перевантажений оператор new для масиву об'єктiв типу Point

void\* operator new[](size\_t size)

{

void\* ptr;

cout << "Застосування перевантаженого оператора new[]." << ::endl;

ptr = malloc(size);

if (!ptr)

{

bad\_alloc error;

throw error;

}

return ptr;

}

// перевантажений оператор delete для масиву об'єктiв типу Point

void operator delete[](void\* ptr)

{

cout << "Видалення масиву з пам'ятi оператором delete[]" << ::endl;

free(ptr);

}

};

int main()

{

setlocale(LC\_ALL, "Russian");

// перевантаження операторiв new та delete для масивiв об'єктiв

Point\* p1;

try

{

// виклик перевантаженої операторної функцiї operator new[]()

// класу Point

p1 = new Point[5]();

}

catch (bad\_alloc error)

{

cout << "Помилка при видiленнi пам'ятi для об'єкту p1." << ::endl;

return 1;

}

delete[] p1;

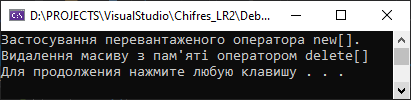
//cout << "x = 6\ny = 8\nx = 10\ny = 13\nx = 16\ny = 18\nx = -2\ny = -2\n";

system("pause");

return 0;

}

Результат роботи програми наведений нижче:



У результаті виконання вищенаведеного коду на екран буде виведено наступний результат.

Застосування перевантаженого оператора new[].  
Видалення масиву з пам'яті оператором delete[].

*Приклад 27.4.* Нижче наведено приклад глобального перевантаження операторів new[] та delete[] для масивів класу Point

#include <iostream>

#include <cstdlib>

#include <math.h>

#include <cstdlib>

#include <new>

using namespace std;

class Point

{

private:

double x, y;

public:

// конструктори

Point()

{

x = y = 0.0;

}

Point(double \_x, double \_y)

{

x = \_x;

y = \_y;

}

// методи доступу

void Get(double\* \_x, double\* \_y)

{

\*\_x = x;

\*\_y = y;

}

void Set(double \_x, double \_y)

{

x = \_x;

y = \_y;

}

};

// перевантажений оператор new для одиночного об'єкту класу Point

void\* operator new(size\_t size)

{

void\* ptr;

ptr = malloc(size);

if (!ptr)

{

bad\_alloc error;

throw error;

}

return ptr;

}

// перевантажений оператор delete для одиночного об'єкту класу Point

void operator delete(void\* ptr)

{

cout << "Звiльнення пам'ятi оператором delete" << ::endl;

free(ptr);

}

// перевантажений оператор new для масиву об'єктiв типу Point

void\* operator new[](size\_t size)

{

void\* ptr;

cout << "Застосування перевантаженого оператора new[]." << ::endl;

ptr = malloc(size);

if (!ptr)

{

bad\_alloc error;

throw error;

}

return ptr;

}

// перевантажений оператор delete для масиву об'єктiв типу Point

void operator delete[](void\* ptr)

{

cout << "Видалення масиву з пам'ятi оператором delete[]" << ::endl;

free(ptr);

}

int main()

{

setlocale(LC\_ALL, "Russian");

// глобальне перевантаження операторiв new та delete для масивiв об'єктiв

Point\* p1; // покажчик на клас Point

float\* p2; // покажчик на float

try

{

// виклик перевантаженої глобальної операторної функцiї operator new[]()

// класу Point

p1 = new Point[5]();

}

catch (bad\_alloc error)

{

cout << "Помилка при видiленнi пам'ятi для об'єкту p1." << ::endl;

return 1;

}

delete[] p1;

// виклик глобальної операторної функцiї operator new[]() для базового типу float

try

{

p2 = new float[10]; // виклик функцiї

}

catch (bad\_alloc error)

{

cout << "Помилка при видiленнi пам'ятi для об'єкту p2." << ::endl;

return 2;

}

delete[] p2;

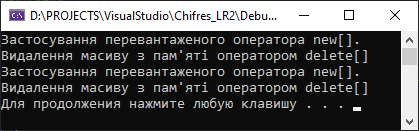
//cout << "x = 6\ny = 8\nx = 10\ny = 13\nx = 16\ny = 18\nx = -2\ny = -2\n";

system("pause");

return 0;

}

Результат роботи програми наведений нижче:



У результаті виконання вищенаведеного коду буде виведено наступний результат:

Застосування перевантаженого оператора new[].  
Видалення масиву з пам'яті оператором delete[].

Як видно з результату, перевантажені глобальні операторні функції operator new[] та operator delete[] для масивів працюють однаково як з базовими типами (float) так і з типами-класами (Point).

***Порядок виконання роботи:***

Розглянути приклади, наведені у теоретичних відомостях.

**Лабораторна робота №28**

# **Перевантаження оператору присвоювання =**

*Мета роботи*: набуття навичок роботи з перевантаженими операторами.

*Теоретичні основи*:

Якщо у програмі оголосити клас, в якому оператор присвоєння не перевантажується, то для цього класу компілятором буде створено оператор присвоювання за замовчуванням. Отже, оператор присвоювання за замовчуванням – це такий оператор присвоєння, який автоматично створюється компілятором для забезпечення копіювання одного екземпляру класу іншому екземпляру класу. При виклику оператора присвоювання за замовчуванням відбувається побітове копіювання одного екземпляру класу іншому.

Наприклад, для двох екземплярів класів mc1 та mc2.

*MyClass mc1;*

*MyClass mc2;*

якщо викликати

*mc1=mc2; // побітове копіювання mc2 в mc1*

то екземпляри класів будуть вказувати на одну й туж ділянку пам’яті. Це пояснюється тим, що відбудеться побітове копіювання ділянки пам’яті на яку посилається mc2 в ділянку пам’яті на яку посилається mc1. У результаті чого, всі зміни в mc2 будуть відображатись в mc1, і, навпаки, всі зміни в mc1 будуть відображатись в mc2.

Якщо потрібно присвоювати екземпляри класів один одному таким чином, щоб вони були не зв’язані між собою, то для цього в класі потрібно реалізувати операторну функцію operator=(), яка буде виконувати відповідні операції.

Оператор присвоєння = може бути перевантажений для конкретного класу. Загальна форма оператора присвоєння для класу з іменем CMyClass має вигляд:

*class CMyClass*

*{*

*// поля та методи класу CMyClass*

*// ...*

*// перевантажений оператор присвоєння =*

*// операторна функція operator=()*

*CMyClass operator=(CMyClass mc);*

*};*

*// реалізація операторної функції operator=()*

*CMyClass CMyClass::operator=(CMyClass mc)*

*{*

*// інструкції*

*// ...*

*}*

Можна реалізацію операторної функції operator=() розмістити всередині класу CMyClass. У цьому випадку загальна форма класу CMyClass матиме наступний вигляд:

*class CMyClass*

*{*

*// поля та методи класу CMyClass*

*// ...*

*// перевантажений оператор присвоєння =*

*// операторна функція operator=()*

*CMyClass operator=(CMyClass mc)*

*{*

*// реалізація операторної функції*

*// ...*

*}*

*};*

Для класу можна перевантажити оператор присвоєння таким чином, що при виклику операторної функції operator=() буде присвоюватись значення іншого типу. У цьому випадку загальна форма класу, що реалізує операторну функцію operator=(), яка отримує об’єкт іншого типу, має вигляд

*class CMyClass*

*{*

*// поля та методи класу CMyClass*

*// ...*

*// перевантажений оператор присвоєння =*

*// операторна функція operator=()*

*CMyClass operator=(type mc)*

*{*

*// реалізація операторної функції*

*// ...*

*}*

*};*

Тут type – деякий тип даних, який може бути базовим (int, float, double, …) чи типом класу, структури, зчислення.

*Приклад 28.1.* Оголошується клас Complex, що містить дві внутрішні приховані (private) змінні. Ці змінні формують дійсну та уявну складові комплексного числа. У класі Complex оголошується операторна функція

*Complex operator=(Complex cm)*

*{*

*// ...*

*}*

яка перевантажує оператор присвоєння **=** для цього класу. При виклику оператора присвоєння для екземплярів (об’єктів) класу Complex буде викликатись саме ця операторна функція.

#include <iostream>

#include <cstdlib>

#include <math.h>

#include <cstdlib>

#include <new>

using namespace std;

class ArrayFloat100

{

private:

float A[100]; // масив зі 100 чисел

int n; // розмір масиву

public:

// конструктори

ArrayFloat100()

{

n = 0;

for (int i = 0; i < 100; i++)

A[i] = 0.0f;

}

ArrayFloat100(int \_n, float \_A[])

{

for (int i = 0; i < \_n; i++)

A[i] = \_A[i];

n = \_n;

}

// методи доступу

void SetAi(int index, float value)

{

if ((index >= 0) && (index < 100)) A[index] = value;

return;

}

float GetAi(int index)

{

if ((index >= 0) && (index < 100))

return A[index];

return 0;

}

int GetN(void) { return n; }

ArrayFloat100 operator=(ArrayFloat100 af)

{

n = af.n;

for (int i = 0; i < n; i++)

A[i] = af.A[i];

return \*this;

}

};

int main()

{

setlocale(LC\_ALL, "Russian");

// перевантаження оператора = для класу ArrayFloat100

float t;

float A[] = { 5.1, 2.3, 4.1, 1.5, 2.6, 3.3 };

int n;

int d;

n = 6;

ArrayFloat100 af1(n, A);

ArrayFloat100 af2;

// перевірка

t = af1.GetAi(2); // t = 4.1

d = af1.GetN(); // d = 6

cout << "t=" << t << " d=" << d << endl;

t = af2.GetAi(2); // t = 0

d = af2.GetN(); // d = 0

cout << "t=" << t << " d=" << d << endl;

// виклик перевантаженого оператора присвоювання

af2 = af1; // відбувається копіювання масивів

t = af2.GetAi(2); // t = 4.1

d = af2.GetN(); // d = 6

cout << "t=" << t << " d=" << d << endl;

// af1 та af2 вказують на різні ділянки пам'яті - перевірка

af2.SetAi(2, 0.8);

t = af2.GetAi(2); // t = 0.8

cout << "t=" << t << " d=" << d << endl;

t = af1.GetAi(2); // t = 4.1

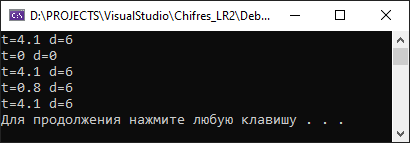
cout << "t=" << t << " d=" << d << endl;

system("pause");

return 0;

}

Результат роботи програми наведений нижче:



*Приклад 28.3.* Задано структуру BOOK, яка описує книгу. Також оголошується клас ArrayBooks, який реалізує динамічний масив типу BOOK. У класі реалізовано:

* внутрішні змінні, які описують масив книг BOOK та його розмір;
* конструктори;
* конструктор копіювання;
* методи доступу GetBook(), SetBook();
* операторна функція operator=(), яка реалізує присвоєння екземплярів класу типу BOOK;
* деструктор.

#include <iostream>

#include <cstdlib>

#include <math.h>

#include <cstdlib>

#include <new>

using namespace std;

struct BOOK

{

string name; // назва книги

string author; // автор книги

int year; // рiк випуску книги

float price; // вартiсть книги

};

// клас, що реалiзує масив книг типу BOOK

class ArrayBooks

{

private:

BOOK\* B; // динамiчний масив книг

int size; // розмiр масиву

public:

// конструктори класу

ArrayBooks()

{

size = 0;

B = NULL;

}

ArrayBooks(int \_size, BOOK \_B[])

{

int i;

size = \_size;

// видiлити пам'ять для масиву

B = new BOOK[size];

for (i = 0; i < \_size; i++)

B[i] = \_B[i];

}

ArrayBooks(int \_size)

{

size = \_size;

B = new BOOK[size];

for (int i = 0; i < size; i++)

{

B[i].author = "";

B[i].name = "";

B[i].price = 0.0f;

B[i].year = 0;

}

}

// конструктор копiювання

ArrayBooks(const ArrayBooks& \_B)

{

size = \_B.size;

B = new BOOK[size];

for (int i = 0; i < size; i++)

B[i] = \_B.B[i];

}

// методи доступу

BOOK GetBook(int index)

{

if ((index >= 0) && (index < size))

return B[index];

}

BOOK SetBook(int index, BOOK \_B)

{

B[index] = \_B;

}

// операторна функцiя operator=()

ArrayBooks& operator=(const ArrayBooks& A)

{

int i;

// звiльнити попередньо видiлену пам'ять

if (size > 0)

delete[] B;

size = A.size;

B = new BOOK[size];

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

{

B[i] = A.B[i];

}

return \*this;

}

// деструктор

~ArrayBooks()

{

if (size > 0)

delete[] B;

}

};

int main()

{

setlocale(LC\_ALL, "Russian");

// перевантаження оператора присвоєння '='

BOOK BB[] = {

{ "title-1", "author-1", 2012, 100.0f },

{ "title-2", "author-2", 2013, 150.0f },

{ "title-3", "author-3", 2014, 200.0f }

};

ArrayBooks AB1(3, BB); // екземпляр класу ArrayBooks

ArrayBooks AB2(4);

int year;

int price;

string author;

string name;

// перевiрка значень AB1

BOOK B;

B = AB1.GetBook(1);

year = B.year; // year = 2013

author = B.author; // author = "author-2"

price = B.price; // price = 150.0

name = B.name; // name = "title-2"

cout << "year = " << year << endl;

cout << "name = " << name << endl;

cout << "price = " << price << endl;

cout << "author = " << author << endl;

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

// перевiрка значень AB2

B = AB2.GetBook(2);

year = B.year; // year = 0

author = B.author; // author = ""

price = B.price; // price = 0

name = B.name; // name=""

cout << "year = " << year << endl;

cout << "name = " << name << endl;

cout << "price = " << price << endl;

cout << "author = " << author << endl;

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

// ВИКЛИК ПЕРЕВАНТАЖЕНОЇ ОПЕРАТОРНОЇ ФУНКЦiЇ operator=()

AB2 = AB1;

// перевiрка значень AB2

B = AB2.GetBook(2);

year = B.year; // year = 2014

author = B.author; // author = "author-3"

price = B.price; // price = 200.0

name = B.name; // name = "title-3"

cout << "year = " << year << endl;

cout << "name = " << name << endl;

cout << "price = " << price << endl;

cout << "author = " << author << endl;

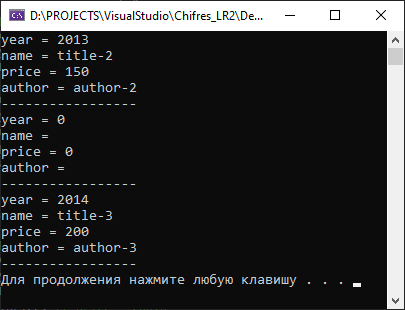
cout << "-----------------\n";

system("pause");

return 0;

}

Результат роботи програми наведений нижче:



Оператор присвоєння не може бути оголошений як “дружній” до класу. Якщо перевантажити оператор присвоєння як “дружній” до класу, то це означає, що буде перевантажений глобальний оператор присвоєння, який викликається для екземплярів класів автоматично. Це, в свою чергу, може призвести до плутанини в операціях присвоєння та збільшенні невидимих помилок. Тому, компілятори мови C++ не допускають перевантажувати оператор присвоєння як “дружній” до класу.

*Приклад 28.4.* У прикладі демонструється перевантаження оператора присвоювання для класу Value. Перевантажений оператор присвоювання реалізує присвоювання цілочисельного значення екземпляру класу Value. У цьому випадку операторна функція має вигляд:

*// операторна функція, яка отримує параметр типу int*

*Value operator=(int d)*

*{*

*// ...*

*}*

Оголошення класу Value та демонстрація його застосування у функції \_tmain() наступні:

#include <iostream>

#include <cstdlib>

#include <math.h>

#include <cstdlib>

#include <new>

using namespace std;

struct BOOK

{

string name; // назва книги

string author; // автор книги

int year; // рiк випуску книги

float price; // вартiсть книги

};

// клас, що реалiзує масив книг типу BOOK

class Value

{

private:

int d;

public:

// конструктори

Value() { d = 0; }

Value(int \_d) { d = \_d; }

int Get(void) { return d; }

void Set(int \_d) { d = \_d; }

// операторна функцiя, яка отримує параметр типу int

Value operator=(int d2)

{

d = d2;

return \*this;

}

};

int main()

{

setlocale(LC\_ALL, "Russian");

// перевантаження операторної функцiї operator=()

// яка отримує параметр базового типу

Value v1(5); // екземпляр класу Value

Value v2;

int t;

// перевiрка

t = v2.Get(); // t = 0

cout << "t = " << t << endl;

// ВИКЛИК ОПЕРАТОРНОЇ ФУНКЦiЇ operator=(int) КЛАСУ Value

v2 = 8;

// перевiрка

t = v2.Get(); // t = 8

cout << "t = " << t << endl;

// ВИКЛИК ОПЕРАТОРА ПРИСВОЮВАННЯ, ЯКИЙ ЗГЕНЕРОВАНИЙ ЗА ЗАМОВЧУВАННЯМ

v2 = v1; // ПОБАЙТОВЕ КОПiЮВАННЯ

// перевiрка

t = v2.Get(); // t = 5

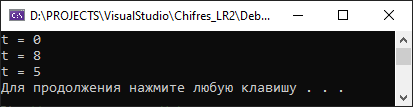
cout << "t = " << t << endl;

system("pause");

return 0;

}

Результат роботи програми наведений нижче:



У вищенаведеному коді, у функціїс\_tmain() оператор присвоєння для класу Value викликається два рази. Перший раз викликається перевантажена в класі операторна функція operator=() у рядку

v2 = 8.

Другий раз викликається оператор, що для класу Value генерується компілятором за замовчуванням

v2 = v1.

Цей оператор виконує побайтову операцію копіювання.

***Порядок виконання роботи:***

Розглянути приклади, наведені у теоретичних відомостях.

**Лабораторна робота №29**

# **Перевантаження оператору присвоювання =**

*Мета роботи*: набуття навичок роботи з перевантаженими операторами.

*Теоретичні основи*:

На перевантаження оператору [] накладаються наступні обмеження:

* до цього оператора заборонено застосовувати “дружні” функції;
* операторна функція, що перевантажують цей оператор, має бути нестатичною функцією-членом.

У мові програмування C++ є можливість перевантажувати оператор індексування елементів масиву **[ ]**. Цей оператор вважається унарним, тобто потребує одного параметра — індексу масиву. Отже, доцільно перевантажувати оператор **[ ]** в класах, де використовуються масиви.

Якщо у класі, що реалізує масив, перевантажено оператор **[ ]**, то об’єкт цього класу можна використовувати як звичайний масив (з використанням доступу за індексом), що є дуже зручно і природньо.

Нехай задано клас з іменем **A**. У класі перевантажено оператор індексування масиву **[ ]**. Тоді використання екземпляру класу **A** може бути таким:

*obj[d]*

де

* obj – екземпляр (об’єкт) класу **A**;
* d – індекс. Це є параметр, що передається операторній operator[]() функції класу **A**.

У вищенаведеному прикладі виклик

*obj[d]*

перетворюється на виклик

*obj.operator[](d)*

Для того, щоб перевантажити оператор індексування елементів масиву [], в класі повинна бути реалізована операторна функція operator[](). Існує 2 варіанти реалізації операторної функції operator[](). Ці варіанти відрізняються поверненням значення з операторної функції. Операторна функція отримує один параметр, який є індексом масиву.

***Варіант 1.*** Цей варіант застосовується, якщо оператор [] має бути використаний у правій (і тільки у правій) частині оператора присвоювання. Тобто значення елементу масиву в класі не змінюється.

У цьому випадку операторна функція operator[]() повертає значення деякого типу. Загальна форма функції при її реалізації в класі

*class ClassName*

*{*

*// ...*

*// операторна функція повертає значення типу type*

*type operator[](int d)*

*{*

*// ...*

*}*

*};*

Після цього, можна використовувати екземпляр класу ClassName наступним чином:

*x = obj[3];*

де

* x – деяка змінна типу type;
* obj – екземпляр (об’єкт) класу ClassName.

***Варіант 2.*** Цей варіант застосовується у випадку, коли потрібно щоб оператор **[]** розміщувався у лівій та правій частині оператора присвоювання. Наявність оператора індексування **[]** у лівій частині оператора присвоювання означає, що в об’єкті класу можна змінювати значення елементу масиву за індексом.

Відмінність від попереднього варіанту (Варіант 1) полягає у тому, що функція повертає посилання **&** на тип. Загальна форма оголошення операторної функції наступна

*class ClassName*

*{*

*// ...*

*// операторна функція повертає посилання (&) на тип type*

*type& operator[](int d)*

*{*

*// ...*

*}*

*};*

Після такого оголошення можна використовувати екземпляр класу ClassName наступним чином:

*// використання оператора [] для об’єкту класу*

*x = obj[3]; // у правій частині оператора присвоювання*

*obj[5] = y; // у лівій частині оператора присвоювання*

де

* x – деяка змінна типу type;
* obj – екземпляр (об’єкт) класу ClassName.

*Приклад 29.1.* Приклад перевантаження оператора **[]** у класі. Клас реалізує масив чисел типу int. Максимальна кількість елементів масиву рівна 10. Операторна функція класу повертає посилання на тип int. Це означає, що оператор **[]** можна використовувати у лівій частині оператора присвоювання.

У класі Array10 оголошуються:

* A – внутрішній масив 10 цілих чисел;
* n – кількість елементів у масиві;
* конструктори класу;
* метод GetN(). Цей метод призначений для читання значенн n;
* методи SetAi() та GetAi() для доступу до елементів масиву A;
* операторна функція operator[](). Ця функція перевантажує оператор доступу до елементу масиву за його індексом.

Текст програми наступний:

#include <iostream>

#include <cstdlib>

#include <math.h>

#include <cstdlib>

#include <new>

using namespace std;

// клас, що реалізує масив з 10 цілих чисел

class Array10

{

private:

int A[10]; // масив

int n; // к-сть елементів у масиві

public:

// конструктори класу

// конструктор без параметрів

Array10()

{

n = 0;

}

// конструктор з 1 параметром

Array10(int \_n)

{

if ((\_n >= 0) && (\_n <= 10))

{

n = \_n;

for (int i = 0; i < n; i++)

A[i] = 0; // занулити масив

}

else

n = 0;

}

// методи доступу до елементів масиву

// зчитати к-сть елементів масиву

int GetN()

{

return n;

}

// записати в елемент масиву значення

void SetAi(int index, int item)

{

if ((index >= 0) && (index < 10))

A[index] = item;

}

// зчитати значення по індексу

// внутрішній метод

int GetAi(int index)

{

return A[index];

}

// перевантаження оператора [] отримання елемента масиву

// операторна функція, реалізована всередині класу

int& operator[](int index)

{

return A[index];

}

};

void main()

{

setlocale(LC\_ALL, "Russian");

// оператор індексування масиву []

Array10 A1(10);

// сформувати масив A1 = [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ]

for (int i = 1; i <= 10; i++)

A1[i - 1] = i; // виклик операторної функції operator[]() класу Array10

// перевірка

int t;

t = A1[3]; // виклик операторної функції, t = 4

// Оператор [] у лівій частині оператора присвоювання

A1[5] = 205;

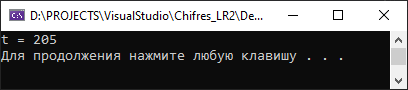
t = A1.GetAi(5); // t = 205

cout << "t = " << t << endl;

system("pause");

}

Результат роботи програми наведений нижче:



Як видно з прикладу, операторна функція повертає посилання на тип int

*// операторна функція - повертає int&*

*int& operator[](int index)*

*{*

*return A[index];*

*}*

Це означає, що оператор **[ ]** можна використовувати у лівій частині оператора присвоювання

*// Оператор [] у лівій частині оператора присвоювання*

*A1[5] = 205;*

Якщо операторну функцію реалізувати так, що вона повертає значення int (а не посилання int&)

*// операторна функція - повертає int*

*int operator[](int index)*

*{*

*return A[index];*

*}*

то використовувати оператор **[]** у лівій частині оператора присвоювання буде заборонено

*// Оператор [] у лівій частині оператора присвоювання*

*A1[5] = 205; // помилка, заборонено!*

*t = A1[3]; // так добре, можна*

*Приклад 29.2.* Перевантаження оператора індексування елементів масиву **[]** для масиву структур типу BOOK.

У прикладі оголошуються:

* структура типу BOOK, що містить інформацію про книгу;
* клас ArrayBooks, який реалізує динамічний масив книг типу BOOK.

У класі ArrayBooks реалізовано:

* внутрішню змінну-покажчик на тип BOOK;
* внутрішню змінну n, яка визначає кількість книг у масиві;
* два конструктори;
* метод GetN(), який повертає кількість книг n у масиві;
* операторна функція operator[](). З допомогою цієї функції здійснюється доступ до заданого елементу масиву. Функція перевантажує оператор доступу за індексом [].

Текст програми, створеної за шаблоном Console Application наступний:

#include <iostream>

#include <cstdlib>

#include <math.h>

#include <cstdlib>

#include <new>

using namespace std;

// структура, що описує книгу

struct BOOK

{

string title; // назва книги

int year; // рік видання

float price; // вартість книги

};

// клас, що реалізує масив книг типу BOOK

// містить перевантажену операторну функцію operator[]()

class ArrayBooks

{

private:

BOOK\* B; // пам'ять для масиву виділяється динамічно

int n; // кількість книг

public:

// конструктори класу

ArrayBooks()

{

n = 0;

B = NULL;

}

ArrayBooks(int \_n)

{

n = \_n;

// виділити пам'ять для покажчиків на BOOK

B = (BOOK\*) new BOOK[n];

// заповнити кожну книгу пустими значеннями

for (int i = 0; i < n; i++)

{

B[i].title = "";

B[i].price = 0.00;

B[i].year = 0;

}

}

// деструктор

~ArrayBooks()

{

// звільнити пам'ять, виділену під масив структур

if (n > 0)

delete[] B;

}

// метод, що повертає значення n

int GetN(void) { return n; }

// операторна функція, яка перевантажує оператор індексування []

// функція повертає BOOK&, для того щоб оператор [] можна було використовувати

// в лівій частині оператора присвоювання

BOOK& operator[](int index)

{

if ((index >= 0) && (index < n))

return B[index];

else

{

// повернути пусту структуру

BOOK BB;

BB.title = "";

BB.price = 0.0;

BB.year = 0;

return BB;

}

}

};

void main()

{

setlocale(LC\_ALL, "Russian");

// демонстрація перевантаження оператора []

ArrayBooks AB(3); // створити об'єкт класу ArrayBooks, у масиві 3 елементи

// виклик операторної функції operator[](), сформувати значення

// книга - 1

AB[0].title = "This is a first book";

AB[0].price = 99.99;

AB[0].year = 2999;

// книга - 2

AB[1].title = "This is a second book";

AB[1].price = 125.95;

AB[1].year = 2020;

// книга - 3

AB[2].title = "Third book";

AB[2].price = 777.77;

AB[2].year = 2000;

// перевірка

string title;

double price;

int year;

title = AB[1].title; // title = "This is a second book"

price = AB[1].price; // price = 125.95

year = AB[1].year; // year = 2020

// вивести на екран

cout << "Title = " << title.c\_str() << ::endl;

cout << "Price = " << price << ::endl;

cout << "Year = " << year << ::endl;

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

title = AB[0].title;

price = AB[0].price;

year = AB[0].year;

// вивести на екран

cout << "Title = " << title.c\_str() << ::endl;

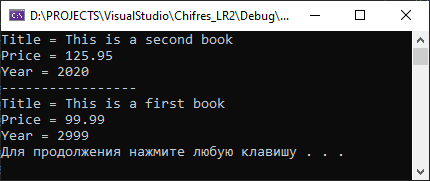
cout << "Price = " << price << ::endl;

cout << "Year = " << year << ::endl;

system("pause");

}

Результат роботи програми наведений нижче:



Результат виконання програми

*Title = This is a second book*

*Price = 125.95*

*Year = 2020*

Операторна функція, яка перевантажує оператор індексування елементів масиву, отримує параметр, який є індексом в масиві. Тип параметру не обов’язково має бути int. Допускається інший тип параметру в індексі масиву (наприклад, char).

*Приклад 29.3.* У прикладі реалізовано клас CharIndex, який містить масив з 26 елементів. У класі оголошується операторна функція operator[](), яка в якості індексу отримує значення типу char.

У програмі підраховується кількість входжень символів латинського алфавіту ‘a’..’z’ в заданому тексті.

Текст програми, створеної за шаблоном Console Application, наступний

#include <iostream>

#include <cstdlib>

#include <math.h>

#include <cstdlib>

#include <new>

using namespace std;

class CharIndex

{

private:

int A[26]; // масив частоти входжень символу в текстi

public:

CharIndex()

{

for (int i = 0; i < 26; i++)

A[i] = 0;

}

// операторна функцiя - доступ за типом char

int& operator[](char c)

{

int position; // позицiя в масивi A

position = (int)c - int('a');

return A[position];

}

};

void main()

{

setlocale(LC\_ALL, "Russian");

// демонстрацiя твердження,

// що тип iндексу не обов'язково має бути int

CharIndex cI; // об'єкт класу CharIndex

string text; // заданий текст

int i;

char sym;

// задано деякий текст

text = "C++ is the best language in the world!";

// обчислити кiлькiсть входжень символiв 'a'..'z' у текстi

for (i = 0; i < text.length(); i++)

cI[text[i]]++; // виклик операторної функцiї

// вивiд, виводяться тiльки ненульовi значення

for (i = 0; i < 26; i++)

{

sym = 'a' + i; // взяти iндекс

if (cI[sym] > 0)

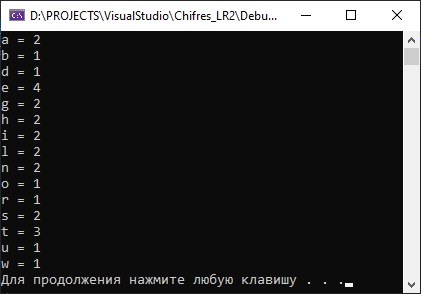
cout << sym << " = " << cI[sym] << ::endl;

}

system("pause");

}

Результат виконання програми



***Порядок виконання роботи:***

Розглянути приклади, наведені у теоретичних відомостях.

**Лабораторна робота №30**

# **Перевантаження оператору () виклику функції**

*Мета роботи*: набуття навичок роботи з перевантаженими операторами.

*Теоретичні основи*:

Оператор виклику функції може бути перевантажений. У цьому випадку у класі створюється операторна функція operator()(). Ця функція може приймати довільну кількість параметрів.

Загальна форма реалізації операторної функції, що перевантажує оператор operator()() наступна. Якщо у деякому класі з іменем ClassName перевантажена операторна функція operator()().

*class ClassName*

*{*

*// ...*

*return\_type operator()(*parameters*)*

*{*

*// ...*

*}*

*};*

то при виклику в іншому коді об’єкту з іменем obj класу ClassName

*ClassName obj;*

*obj(parameters);*

цей виклик буде перетворено в наступний виклик операторної функції

*obj.operator()(parameters);*

тут

* ClassName – ім’я класу, в якому реалізована операторна функція operator()();
* return\_type – тип, який повертає операторна функція operator()();
* parameters – параметри, які отримує операторна функція operator()();
* obj – екземпляр (об’єкт) класу ClassName.

Операторна функція operator()() може приймати будь-яку кількість параметрів. Операторна функція operator()() може бути без параметрів.

*Приклад 30.1.*Задано клас SqEqual, що містить усі дані та методи для розв‘язку квадратного рівняння.

У класі SqEqual реалізовано:

* внутрішні приховані (private) дані a, b, c, x1, x2 для рішення задачі;
* конструктор класу;
* внутрішня прихована (private) функція Solution(), в якій реалізовано алгоритм розв’язку квадратного рівняння. Ця функція викликається з інших методів та операторних функцій класу;
* загальнодоступні (public) методи GetSolution() та GetSolutionP(), які викликають функцію Solution() для розв’язку квадратного рівняння. Ці методи можна викликати з екземпляру класу. Методи реалізовані з метою демонстрації передачі параметрів за посиланням та за покажчиком;
* дві перевантажені загальнодоступні операторні функції operator()(), які можна викликати для отримання розв’язку квадратного рівняння. Функції отримують різну кількість параметрів, тому вони сприймаються компілятором як різні функції. Функція operator()() з двома параметрами в якості коефіцієнтів рівняння використовує значення внутрішніх прихованих (private) змінних класу a, b, c. Функція operator()() з п’ятьма параметрами для розв’язку використовує вхідні параметри a, b, c.

Текст програми, що містить реалізацію класу SqEqual, створеної за шаблоном Console Application наступний

*#include <iostream>*

*using namespace std;*

*// клас, що реалізує дані та методи розв'язку квадратного рівняння*

class SqEqual

{

private:

// коефіцієнти квадратного рівняння

double a;

double b;

double c;

// корені - розв'язки

double x1;

double x2;

// внутрішній метод розв'язку квадратного рівняння

bool Solution(void)

{

double d; // дискримінант

// чи взагалі є рівняння?

if (a == 0)

return false;

d = b \* b - 4 \* a \* c;

if (d >= 0)

{

// є розв'язок рівняння

x1 = (-b - sqrt(d)) / (2 \* a);

x2 = (-b + sqrt(d)) / (2 \* a);

}

else

return false; // рівняння не має розв'язку

return true; // є розв'язок в змінних x1, x2

}

public:

// конструктор класу

SqEqual(double a, double b, double c)

{

this->a = a;

this->b = b;

this->c = c;

}

// загальнодоступні методи розв'язку квадратного рівняння

bool GetSolution(double& x1, double& x2)

{

if (Solution()) // викликати внутрішній метод розв'язку

{

x1 = this->x1;

x2 = this->x2;

return true; // є розв'язок

}

return false; // немає розв'язку

}

bool GetSolutionP(double\* x1, double\* x2)

{

if (Solution())

{

\*x1 = this->x1;

\*x2 = this->x2;

return true; // є розв'язок

}

return false; // немає розв'язку

}

// операторна функція operator()(), що розв'язує квадратне рівняння

// операторна функція отримує 5 параметрів

// функція повертає true, якщо рівняння має розв'язок,

// x1, x2 - розв'язок рівняння, передаються за покажчиком

bool operator()(double a, double b, double c, double\* x1, double\* x2)

{

this->a = a;

this->b = b;

this->c = c;

if (Solution())

{

\*x1 = this->x1;

\*x2 = this->x2;

return true;

}

return false;

}

// варіант 2 - операторна функція отримує тільки 2 параметри, що є розв'язком

// x1, x2 - передаються як посилання

bool operator()(double& x1, double& x2)

{

// коефіцієнти a, b, c отримуються з поточного об'єкту

if (Solution())

{

x1 = this->x1;

x2 = this->x2;

return true; // є рішення

}

return false; // рівняння немає розв'язку

}

};

void main(void)

{

// перевантаження оператора () виклику функції

double a = 2, b = -3, c = 1;

SqEqual e(a, b, c); // створити екземпляр

double x1, x2;

bool f;

// виклик операторної функції operator()() з двома параметрами

f = e(x1, x2);

if (f)

{

// якщо є рішення рівняння 2\*x^2 - 3\*x + 1 = 0, то вивести результат

cout << a << "\* x^2 + (" << b << ") \* x + (" << c << ") = 0" << endl;

cout << "x1 = " << x1 << endl;

cout << "x2 = " << x2 << endl << endl;

}

// виклик операторної функції з 5 параметрами

f = e(5, -7, 2, &x1, &x2);

if (f)

{

// якщо є рішення рівняння, то вивести результат

cout << "5 \* x^2 - 7\*x + 2 = 0" << endl;

cout << "x1 = " << x1 << endl;

cout << "x2 = " << x2 << endl << endl;

}

}

Як видно з вищенаведеного коду, у функції main() виклик операторної функції operator()() з двома параметрами наступний

*f = e(x1, x2).*

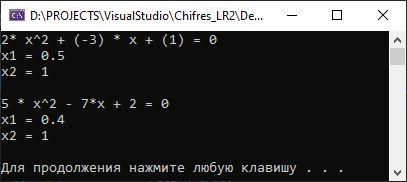
Якщо рівняння має розв’язок, то змінні x1 та x2 будуть мати відповідні значення. У цій операторній функції передача значень у змінні x1, x2 відбувається за посиланням.

Виклик операторної функції operator()() з п’ятьма параметрами наступний

*f = e(5, -7, 2, &x1, &x2)*.

У цьому випадку у функцію передаються адреси змінних x1, x2. У функції доступ до змінних відбувається за покажчиком.

Результат виконання програми



*Приклад 30.2.* Нехай задано клас Point, що реалізує точку на координатній площині.

У класі Point оголошуються:

* внутрішні приховані (private) змінні x, y;
* конструктор класу, який ініціалізує нульовими значеннями значення змінних x, y;
* методи доступу до внутрішніх змінних SetP(), GetX(), GetY();
* перевантажена операторна функція operator()() без параметрів. Ця функція повертає поточний об’єкт (екземпляр) класу;
* перевантажена операторна функція operator()() з двома параметрами. Ця функція повертає поточний екземпляр класу, значення внутрішніх змінних x, y якого рівні значенню відповідних параметрів \_x, \_y.

Реалізація класу Point наступна:

*// клас, що реалізує точку*

class Point

{

private:

int x, y;

public:

Point()

{

x = y = 0;

}

// методи доступу

void SetP(int \_x, int \_y)

{

x = \_x;

y = \_y;

}

int GetX() { return x; }

int GetY() { return y; }

// перевантаження операторної функцiї operator()() без параметрiв

Point operator()()

{

return \*this; // повернути поточний об'єкт

}

// перевантаження операторної функцiї operator()() з двома параметрами

Point operator()(int \_x, int \_y)

{

x = \_x;

y = \_y;

return \*this;

}

};

void main()

{

setlocale(LC\_ALL, "Russian");

// перевантаження оператора () виклику функцiї

Point p1; // створити об'єкт

int t;

p1.SetP(3, 5); // встановити значення

Point p2; // новий об'єкт

// виклик операторної функцiї без параметрiв

p2 = p1();

// перевiрка

t = p2.GetX(); // t = 3

cout << "t=" << t << endl;

t = p2.GetY(); // t = 5

cout << "t=" << t << endl;

// виклик операторної функцiї operator()() з двома параметрами

p2 = p1(7, 8);

// перевiрка

t = p2.GetX(); // t = 7

cout << "t=" << t << endl;

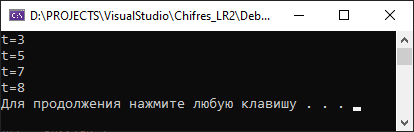
t = p2.GetY(); // t = 8

cout << "t=" << t << endl;

system("pause");

}

Результат роботи програми наведений нижче:



Як видно з вищенаведеного прикладу, операторна функція operator()() може мати декілька реалізацій в класі. Ці реалізації повинні відрізнятися кількістю або типами параметрів, які передаються в операторну функцію.

*Приклад 30.3.* У прикладі оголошується клас Complex, в якому перевантажуються:

* оператор виклику функції **( )**;
* оператор додавання **+**, який здійснює додавання комплексних чисел.

У класі Complex реалізовано наступні елементи:

* внутрішні приховані (private) змінні real та imag. Ці змінні визначають дійсну та уявну частину комплексного числа;
* два конструктори;
* методи GetI() та GetR() для читання значень внутрішніх змінних real та imag;
* операторна функція operator+(), яка переваантажує оператор **+**. Функція сумує два екземпляри (об’єкти) класу Complex;
* операторна функція operator()(), яка перевантажує оператор виклику функції (). Функція створює екземпляр класу Complex на основі вхідних параметрів.

Текст програми типу Console Application, що демонструє реалізацію класу Complex наступний:

#include <iostream>

#include <cstdlib>

#include <math.h>

#include <cstdlib>

#include <new>

using namespace std;

// клас, що реалiзує комплексне число

class Complex

{

private:

double real; // дiйсна частина комплексного числа

double imag; // уявна частина

public:

// конструктори

Complex()

{

real = imag = 0;

}

Complex(double real, double imag)

{

this->real = real;

this->imag = imag;

}

// методи доступу

double GetI() { return imag; }

double GetR() { return real; }

// перевантажений оператор додавання +

Complex operator+(Complex c)

{

// додавання комплексних чисел

return Complex(real + c.real, imag + c.imag);

}

// перевантажений оператор виклику функцiї

Complex operator()(double real, double imag)

{

this->real = real;

this->imag = imag;

return \*this;

}

};

void main()

{

setlocale(LC\_ALL, "Russian");

Complex c1(3, 4);

Complex c2;

Complex c3;

double r, i;

// виклик операторних функцiй operator+() та operator()()

c3 = c1 + c2(5, 6); // c3 = c1.operator+(c2.operator(5,6))

// перевiрка

r = c3.GetR(); // r = 3 + 5 = 8

i = c3.GetI(); // i = 4 + 6 = 10

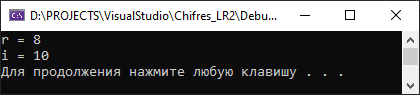
cout << "r = " << r << endl;

cout << "i = " << i << endl;

system("pause");

}

Результат роботи програми наведений нижче:



***Порядок виконання роботи:***

Розглянути приклади, наведені у теоретичних відомостях.

**Лабораторна робота №31**

# **Перевантаження оператору () виклику функції**

*Мета роботи*: набуття навичок роботи з перевантаженими операторами.

*Теоретичні основи*:

У C++ оператор доступу до члена об’єкта **->** можна перевантажувати. Якщо оператор **->** перевантажений, то виклик елемента класу має такий загальний вигляд

*obj->item*

де

* obj – об’єкт класу;
* item – деякий елемент класу (внутрішня змінна, метод). Цей елемент повинен бути членом класу, який є доступний всередині об’єкту.

При перевантаженні слід враховувати наступні особливості:

* оператор **->** вважається унарним;
* операторна функція operator->() повинна повертати покажчик на об’єкт класу, для якого він визначений;
* операторна функція operator->() повинна бути членом класу, для якого вона реалізується.

Загальна форма класу, в якому перевантажений оператор ->, наступна

*class ClassName*

*{*

*// ...*

*// операторна функція*

*ClassName\* operator->()*

*{*

*// тіло операторної функції*

*// ...*

*return this; // повернути покажчик на об'єкт класу*

*}*

*};*

*Приклад 31.1.* Оголошується клас Double, що містить внутрішню змінну d типу double. У класі реалізована перевантажена функція operator->(), яка перевантажує оператор ->.

#include <iostream>

#include <cstdlib>

#include <math.h>

#include <cstdlib>

#include <new>

using namespace std;

class Double

{

public:

double d; // внутрішня змінна

// операторна функція, що перевантажує

// оператор ->

Double\* operator->()

{

return this; // повернути покажчик на клас

}

};

void main()

{

setlocale(LC\_ALL, "Russian");

// перевантаження оператора доступу за покажчиком ->

Double D; // екземпляр класу D

double x;

D.d = 3.85;

// виклик операторної функції operator->()

x = D->d;

// інший спосіб доступу

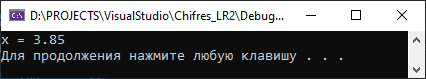
x = D.d; // x = 3.85

cout << "x = " << x << endl;

system("pause");

}

Результат роботи програми наведений нижче:



*Приклад 31.2.* У прикладі демонструється перевантаження оператору **->** для класу, в якому реалізовано динамічний масив об’єктів (екземплярів) класу.

Задано клас Point, що описує точку на площині. У класі Point реалізовано такі елементи:

* внутрішні змінні x, y – координати точки;
* конструктори;
* методи доступу GetX(), GetY(), SetXY().

Також задано клас Polygon, що реалізує масив точок. У класі оголошуються:

* внутрішня прихована (private) змінна-покажчик на тип Point. Ця змінна містить покажчик на динамічний масив точок типу Point;
* внутрішня прихована (private) змінна n, що визначає кількість елементів масиву;
* конструктор;
* метод GetN(), який призначений для отримання значення n;
* метод Add(), який додає нову точку типу Point до масиву. Нова точка отримується як параметр методу;
* метод Delete(), який видаляє точку з заданої позиції index.
* Позиція index є вхідним параметром методу;
* метод GetPoint(), який повертає значення точки типу Point в заданій позиції index;
* метод Show(), що відображає на екрані значення масиву точок в класі Polygon;
* операторна функція operator->(), яка перевантажує оператор **->**.

Текст програми типу Console Application, що містить реалізації класів Point та Polygon, наступний:

#include <iostream>

#include <cstdlib>

#include <math.h>

#include <cstdlib>

#include <new>

using namespace std;

// клас, що описує точку

class Point

{

private:

double x, y; // внутрiшнi змiннi

public:

// конструктори класу

Point()

{

x = y = 0;

}

Point(double x, double y)

{

this->x = x;

this->y = y;

}

// методи доступу

void SetXY(double x, double y)

{

this->x = x;

this->y = y;

}

double GetX(void) { return x; }

double GetY(void) { return y; }

};

// клас багатокутник

class Polygon

{

private:

Point\* Pt; // масив точок

int n; // кiлькiсть точок

public:

// конструктор

Polygon()

{

n = 0;

Pt = NULL;

}

// методи доступу

// зчитати кiлькiсть точок

int GetN(void) { return n; }

// додати нову точку

void Add(Point p)

{

Point\* Pt2;

// видiлити пам'ять на новий масив типу Point - на 1 елемент бiльше

Pt2 = new Point[n + 1];

// скопiювати Pt=>Pt2

for (int i = 0; i < n; i++)

Pt2[i].SetXY(Pt[i].GetX(), Pt[i].GetY());

// додати зайвий елемент

Pt2[n].SetXY(p.GetX(), p.GetY());

// знищити пам'ять, видiлену пiд старий масив

if (n > 0)

delete[] Pt;

// присвоїти внутрiшнiй змiннiй новий масив

Pt = Pt2;

n++;

}

// видалити точку в позицiї pos

void Delete(int pos)

{

if (n < 0) return;

if (pos > (n - 1)) return;

if (pos < 0) return;

// якщо один елемент, то видалити його

if (n == 1)

{

n = 0;

delete[] Pt;

return;

}

// n>1

Point\* Pt2; // новий масив

double tx, ty; // додатковi змiннi

Pt2 = new Point[n - 1]; // видiлити пам'ять пiд новий масив - на 1 елемент менше

// скопiювати Pt в Pt2 в обхiд позицiї pos

// до позицiї pos

for (int i = 0; i < pos; i++)

{

tx = Pt[i].GetX();

ty = Pt[i].GetY();

Pt2[i].SetXY(tx, ty);

}

// пiсля позицiї pos

for (int i = pos + 1; i < n; i++)

{

tx = Pt[i].GetX();

ty = Pt[i].GetY();

Pt2[i - 1].SetXY(tx, ty); // Pt2[i-1] - важливо

}

// звiльнити пам'ять, видiлену для Pt ранiше

delete[] Pt;

// зменшити к-сть елементiв на 1

n--;

// встановити нове значення Pt

// перенаправити Pt на Pt2

Pt = Pt2;

}

// взяти точку з позицiї index

Point GetPoint(int index)

{

if ((index >= 0) && (index < n))

return Point(Pt[index].GetX(), Pt[index].GetY());

else

return Point(0.0, 0.0);

}

// метод, що виводить масив

void Show(void)

{

double x, y;

int i;

for (i = 0; i < n; i++)

{

x = Pt[i].GetX();

y = Pt[i].GetY();

cout << "x" << i + 1 << " = " << x << ", ";

cout << "y" << i + 1 << " = " << y;

cout << endl;

}

}

// операторна функцiя, що перевантажує оператор ->

Polygon\* operator->()

{

return this;

}

};

void main()

{

setlocale(LC\_ALL, "Russian");

// перевантаження оператора доступу за покажчиком ->

Polygon Pol; // екземпляр класу Polygon

double x, y;

int i;

Point Pt;

// сформувати довiльний масив з 5 точок

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

{

x = (double)(i \* 2);

y = (double)(i + 3);

Pt.SetXY(x, y); // сформувати точку

// виклик операторної функцiї operator->

Pol->Add(Pt); // додати точку

}

cout << "Array of points: \n";

// вивести масив через звертання до операторної функцiї operator->()

Pol->Show();

// видалити точку в позицiї 2

Pol->Delete(2);

cout << "Modified array: \n";

cout << "n = " << Pol->GetN() << endl;

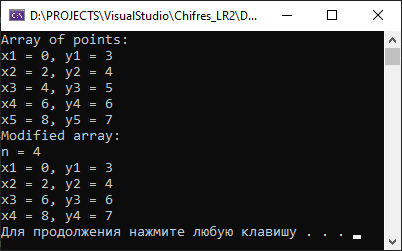
// знову вивести масив

Pol->Show();

system("pause");

}

Результат роботи програми наведений нижче:



У мові C++ оператор **‘ , ‘** може бути перевантажений. При перевантаженні оператора ‘ , ‘ у класі має бути оголошена операторна функція operator,(). В тіло операторної функції можна помістити будь-який код. Тобто, оператор ‘ , ‘ при бажанні може виконувати будь-які нестандартні операції над об’єктами класу.

У стандартному випадку оператор ‘ , ‘ використовується в операції присвоєння за зразком

*obj1 = (obj2, obj3, ..., objN);*

де obj1, …, objN – екземпляри деякого класу.

У випадку стандартного використання оператора ‘ , ‘ потрібно врахувати наступні особливості:

* оператор ‘ , ‘ вважається бінарним. Тому операторна функція operator,() отримує один параметр;
* при використанні перевантаженого оператора ‘ , ‘ в операції присвоєння приймається до уваги останній аргумент (цей аргумент є результатом оператора). Усі інші аргументи ігноруються.

В загальному при перевантаженні оператора ‘ , ‘ клас має такий вигляд:

*class ClassName*

*{*

*// ...*

*// операторна функція, яка перевантажує оператор ','*

*ClassName operator,(ClassName obj)*

*{*

*// ...*

*}*

*};*

де

* ClassName – ім’я класу, в якому перевантажується оператор ‘ , ‘;
* obj – ім’я екземпляру класу, що передається як параметр в операторну функцію operator,().

*Приклад 31.3.* У прикладі перевантажується оператор ‘,’ в класі Coords3D, що реалізує координати в просторі. У класі оголошуються:

* три внутрішні приховані (private) змінні з іменами x, y, z;
* два конструктори класу;
* метод доступу Get(), призначений для отримання значень x, y, z;
* операторна функція operator,(), яка перевантажує оператор ‘ , ‘.

Вигляд програми для додатку типу Console Application наступний:

#include <iostream>

#include <cstdlib>

#include <math.h>

#include <cstdlib>

#include <new>

using namespace std;

class Coords3D

{

private:

double x, y, z;

public:

Coords3D()

{

x = y = z = 0;

}

Coords3D(double x, double y, double z)

{

this->x = x;

this->y = y;

this->z = z;

}

// метод читання x, y, z

void Get(double& x, double& y, double& z)

{

x = this->x;

y = this->y;

z = this->z;

}

// перевантажений оператор ,

Coords3D operator,(Coords3D obj)

{

Coords3D tmp;

tmp.x = obj.x;

tmp.y = obj.y;

tmp.z = obj.z;

return tmp;

}

};

void main()

{

setlocale(LC\_ALL, "Russian");

double x, y, z;

Coords3D c1(1, 3, 5); // екземпляри класу Coords3D

Coords3D c2(2, 4, 6);

Coords3D c3;

// виклик операторної функцiї c3.operator,(c2)

c3 = (c1, c2); // у c3 записується c2

// перевiрка

c3.Get(x, y, z); // x = 2, y = 4, z = 6

cout << "x = " << x << endl;

cout << "y = " << y << endl;

cout << "z = " << z << endl;

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

//створити iнший екземпляр

Coords3D c4(10, 15, 20);

c3 = (c2, c1, c4); // c3 <= c4

// перевiрка

c3.Get(x, y, z); // x = 10, y = 15, z = 20

cout << endl;

cout << "x = " << x << endl;

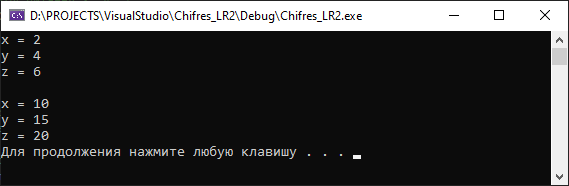
cout << "y = " << y << endl;

cout << "z = " << z << endl;

system("pause");

}

Результат роботи програми наведений нижче:



У вищенаведеному коді в рядку

*c3 = (c1, c2);*

викликається операторна функція c3.operator,(c2). Отже, до уваги приймається останній екземпляр c2. Екземпляр з іменем c1 ігнорується. Це стосується і рядка

*c3 = (c2, c1, c4);*

де об’єкту c3 присвоюються значення внутрішніх змінних об’єкта c4.

Результат роботи програми

*x = 2*

*y = 4*

*z = 6*

*x = 10*

*y = 15*

*z = 20*

***Порядок виконання роботи:***

Розглянути приклади, наведені у теоретичних відомостях.

**Лабораторна робота №32**

### **Типи відношень між класами: is-a, has-a, uses**

*Мета роботи*: набуття навичок роботи з перевантаженими операторами.

*Теоретичні основи*:

Між класами можливі два типи відношень:

1. Відношення типу is-a (є, являється) при якому один клас є підвидом іншого класу. При такому співвідношенні один клас розширює (деталізує) можливості іншого класу. Розширення можливостей класу здійснюється завдяки використанню спадковості.

2. Відношення, при якому існує взаємозв’язок між двома класами. Тут виділяють два підвиди взаємозв’язку між класами:

2.1. Відношення типу has-a (клас містить інший клас). У цьому випадку в класі оголошується один або декілька екземплярів іншого класу. При даному відношенні можливі два випадки взаємодії. Перший випадок, це коли об’єкт (екземпляр), що оголошений в класі, не є складовою частиною класу (агрегація) і його використання не впливає на функціональну роботу класу. Другий випадок, коли об’єкт, що оголошений в класі, є складовою частиною цього класу (композиція).

2.2. Відношення типу uses (клас “використовує інший клас). У цьому випадку клас містить програмний код іншого вкладеного класу, до якого він має доступ.

*Приклад 32.1.* Суть відношення типу is-a полягає в тому, що клас є підвидом іншого класу. У даному прикладі базовий клас Circle розширюється класом CircleColor. Клас CircleColor є підвидом класу Circle і додає до нього поле кольору.

У класі Circle реалізовано наступні елементи:

* внутрішні приховані (private) поля x, y, r;
* конструктор Circle() з 3 параметрами, які заповнюють значення внутрішніх полів;
* методи доступу GetXYR(), SetXYR() до полів класу;
* функція Area() обчислення площі круга.

У похідному класі CircleColor реалізовано поля та методи які доповнюють клас Color:

* внутрішнє приховане поле класу color;
* конструктор CircleColor() з 4 параметрами, який звертається до конструктора класу Color;
* методи GetColor(), SetColor() для доступу до прихованої змінної color.

Демонстраційний текст програми на C++ типу ConsoleApplication наступний:

#include <iostream>

#include <cstdlib>

#include <math.h>

#include <cstdlib>

#include <new>

using namespace std;

class Circle

{

private:

double x, y; // Координати центру кола

double r; // радіус кола

public:

Circle(double x, double y, double r)

{

this->x = x;

this->y = y;

if (r > 0) this->r = r;

else

this->r = 1.0;

}

void GetXYR(double& x, double& y, double& r)

{

x = this->x;

y = this->y;

r = this->r;

}

void SetXYR(double x, double y, double r)

{

this->x = x;

this->y = y;

if (r > 0)

this->r = r;

else

this->r = 1.0;

}

double Area()

{

const double Pi = 3.141592;

return Pi \* r \* r;

}

};

class CircleColor : public Circle

{

private:

unsigned int color = 0; // колiр кола

public:

CircleColor(double x, double y, double r, unsigned int color) :Circle(x, y, r)

{

this->color = color;

}

unsigned int GetColor()

{

return color;

}

void SetColor(unsigned int color)

{

this->color = color;

}

};

void main()

{

setlocale(LC\_ALL, "Russian");

Circle cr(1, 3, 2);

double area = cr.Area();

cout << "The instance of class Color:" << endl;

cout << "area = " << area << endl;

double x, y, r;

cr.GetXYR(x, y, r);

cout << "x = " << x << ", y = " << y << ", radius = " << r << endl;

CircleColor crCol(2, 4, 6, 1);

unsigned int col = crCol.GetColor();

crCol.GetXYR(x, y, r);

cout << "The instance of class ColorCircle: " << endl;

cout << "color = " << col << endl;

cout << "x = " << x << endl;

cout << "y = " << y << endl;

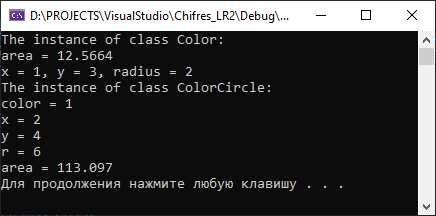
cout << "r = " << r << endl;

cout << "area = " << crCol.Area() << endl;

system("pause");

}

Результат роботи програми наведений нижче:



При відношенні has-a клас містить один або декілька об’єктів (екземплярів) іншого класу. Існує два види відношення has-a:

* агрегація. Це випадок, коли один або декілька вкладених об’єктів не є частиною класу, тобто клас може містити будь-яку кількість таких об’єктів (навіть 0). Для кращого вивчення агрегації дивіться наведені нижче приклади;
* композиція. У цьому випадку один або декілька вкладених об’єктів є частиною класу, тобто без цих об’єктів неможливе логічне існування самого класу.

У випадку агрегації клас містить множини (один або декілька) об’єктів інших класів, які не є складовою частиною цього класу (не містять код, що доповнює роботу самого класу).

*Приклад 32.2.* У даному скороченому коді демонструється агрегація на прикладі класів Triangle, Circle, Figures. Клас Figures може містити різну кількість різних фігур (навіть 0). При можливості до класу Figures може бути додано масиви інших фігур, наприклад, Rectangle (прямокутник). У будь-якому випадку клас Figures буде повноцінним функціонально, отже, це є агрегація.

*// Клас трикутник*

*class Triangle*

*{*

*// Методи та поля класу Triangle*

*// ...*

*};*

*// Клас, що реалізує коло*

*class Circle*

*{*

*// Методи та поля класу Circle*

*// ...*

*};*

*// Клас, що реалізує різні геометричні фігури.*

*// Використовується тип відношення - агрегація.*

*class Figures*

*{*

*Triangle tr[10]; // масив трикутників*

*unsigned int n\_tr; // к-сть трикутників у масиві tr*

*Circle cr[10]; // масив кіл*

*unsigned int n\_cr; // к-сть кіл у масиві cr*

*// Інші поля та методи класу*

*// ...*

*};*

*Приклад 32.3.* У прикладі клас BusStation (Автостанція) містить масиви екземплярів класів Bus (Автобус), Car (Автомобіль). Кількість елементів у масивах класів Bus та Car може змінюватись. Навіть, якщо на автостанції в даний момент часу не буде жодного автобуса чи автомобіля, автостанція буде функціонувати. Це є агрегація.

#include <iostream>

#include <cstdlib>

#include <math.h>

#include <cstdlib>

#include <new>

using namespace std;

const int MAX\_VEHICLES = 10;

class Car

{

private:

string model; // марка автомобiля

public:

Car(string \_model) :model(\_model)

{

}

Car() : Car("")

{

}

string GetModel() { return model; }

void SetModel(string \_model)

{

model = \_model;

}

void Print()

{

cout << "model = " << model << endl;

}

};

class Bus

{

private:

string model; // Марка автобуса

unsigned int seats; // кiлькiсть посадкових мiсць

public:

Bus(string \_model, int \_seats) : model(model), seats(\_seats)

{

}

Bus() : Bus("", 0) {}

void Get(string& \_model, int& \_seats)

{

\_model = model;

\_seats = seats;

}

void Set(string \_model, int \_seats)

{

model = \_model;

seats = \_seats;

}

void Print()

{

cout << "model = " << model << ", seats = " << seats << endl;

}

};

class BusStation

{

private:

Bus B[MAX\_VEHICLES]; // масив об'єктiв класу Bus - вiдношення has-a

unsigned int nBus; // поточна к-сть автобусiв

Car C[MAX\_VEHICLES]; // масив об'єктiв класу Car

unsigned int nCar; // поточна к-сть автомобiлiв

public:

BusStation()

{

nBus = nCar = 0;

}

Bus GetBus(unsigned int number)

{

if (number < nBus)

return B[number];

else

{

cout << "Error. Incorrect number of a bus." << endl;

return Bus("", 0);

}

}

void AddBus(string model, unsigned int seats)

{

if ((nCar + nBus) < MAX\_VEHICLES)

{

nBus++; // збiльшити к-сть автобусiв на 1

B[nBus - 1].Set(model, seats); // викликати метод класу Bus

cout << "A new bus is added!" << endl;

}

else

{

cout << "Cannot add a new bus. Sorry" << endl;

return;

}

}

void DelBus(unsigned int number)

{

if (number < nBus)

{

for (int i = number; i < nBus - 1; i++)

B[i] = B[i + 1];

nBus--;

}

}

Car GetCar(unsigned int number)

{

if (number < nCar)

return C[number];

else

{

cout << "Error. Incorrect number of a car." << endl;

return Car("");

}

}

void AddCar(string model)

{

if ((nCar + nBus) < MAX\_VEHICLES)

{

nCar++; // збiльшити к-сть автомобiлiв

C[nCar - 1].SetModel(model); // додати данi автомобiля

cout << "A new car is added!" << endl;

}

else

{

cout << "Cannot add a new car. Sorry" << endl;

return;

}

}

void DelCar(unsigned int number)

{

if (number < nCar)

{

for (int i = number; i < nCar - 1; i++)

C[i] = C[i + 1];

nCar--;

}

}

void Print()

{

cout << "Info about bus station:" << endl;

cout << "nBus = " << nBus << endl;

if (nBus > 0)

{

cout << "Info about buses:" << endl;

for (int i = 0; i < nBus; i++)

{

B[i].Print();

}

}

cout << "nCar = " << nCar << endl;

if (nCar > 0)

{

cout << "Info about cars:" << endl;

for (int i = 0; i < nCar; i++)

{

C[i].Print();

}

}

}

};

void main()

{

setlocale(LC\_ALL, "Russian");

BusStation bs;

bs.AddBus("Mersedes Sprinter", 28);

bs.AddBus("Renault Trafic", 29);

bs.AddCar("Bentley");

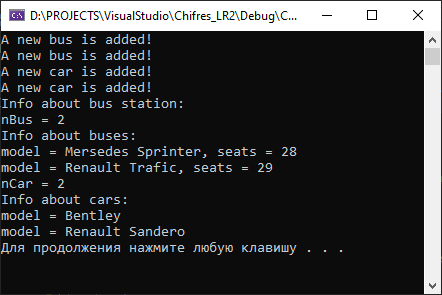
bs.AddCar("Renault Sandero");

bs.Print();

system("pause");

}

Результат роботи програми наведений нижче:



*Приклад 32.3.* У прикладі продемонстровано композицію для класів Vehicle, Wheel, Car. В автомобіль (Car) входять двигун (клас Vehicle) та колесо (Wheel), які є його складовою частиною – це є композиція (об’єднання).

*// Клас Двигун*

*class Vehicle*

*{*

*// Поля та методи класу*

*// ...*

*};*

*// Клас Колесо*

*class Wheel*

*{*

*// Поля та методи класу*

*// ...*

*};*

*// Клас Автомобіль - містить обов'язкові елементи,*

*// які є складовими Автомобіля*

*class Car*

*{*

*// Екземпляри обов'язкових класів,*

*// що є частиною даного класу - це є композиція*

*private:*

*Vehicle veh; // автомобіль містить двигун (обов'язково)*

*Wheel whl[4]; // колеса - 4 штуки (обов'язково)*

*// Поля та методи класу*

*// ...*

*};*

*Приклад 32.4.* Ще один приклад композиції. Клас Bike (Велосипед) містить екземпляри класів Wheel (Колесо) та Saddle (сідло), які є його складовою частиною. Велосипед не може повноцінно функціонувати без коліс чи сідла, тому це є композиція.

*// Клас Колесо*

*class Wheel*

*{*

*// Поля та методи класу*

*// ...*

*};*

*// Клас Велосипед, містить колеса, сідло*

*// Клас сідло*

*class Saddle*

*{*

*// Поля та методи класу*

*// ...*

*};*

*// Клас Велосипед, містить об'єкти класів,*

*// які є частиною Велосипеду (сідло, колеса) - це є композиція*

*class Bike*

*{*

*Saddle sd; // одне сідло - частина, яка доповнює велосипед*

*Wheel whl[2]; // два колеса - частина велосипеду*

*// Поля та методи класу Bike*

*// ...*

*};*

***Порядок виконання роботи:***

Розглянути приклади, наведені у теоретичних відомостях.

**Лабораторна робота №33**

## **Поняття виключної ситуації. Блок try…catch. Оператор throw**

*Мета роботи*: набуття навичок роботи з перевантаженими операторами.

*Теоретичні основи*:

У програмах на C++ можуть виникати помилки. Розрізняють три типи помилок, які можуть виникати у програмах:

* синтаксичні. Це помилки в синтаксисі мови C++. Вони можуть зустрічатись в іменах операторів, функцій, розділювачів і т.д. У цьому випадку компілятор визначає наявність синтаксичної помилки і видає відповідне повідомлення. У результаті виконавчий (\*.exe) файл не створюється і програма не виконується;
* логічні. Це помилки програміста, які важко виявити на етапі розробки програми. Ці помилки виявляються на етапі виконання під час тестування роботи програми. Логічні помилки можна виявити тільки за результатами роботи програми. Прикладом логічних помилок може бути неправильна робота з покажчиками у випадках виділення/звільнення пам’яті;
* помилки часу виконання. Такі помилки виникають під час роботи програми. Помилки часу виконання можуть бути логічними помилками програміста, помилками зовнішніх подій (наприклад, нехватка оперативної пам’яті), невірним введенням даних користувачем тощо. У результаті виникнення помилки часу виконання, програма призупиняє свою роботу. Тому, важливо перехопити цю помилку і правильно обробити її для того, щоб програма продовжила свою роботу без зупинки.

Дана тема висвітлює застосування механізму перехоплення помилок часу виконання.

Виключна ситуація – це подія, що призвела до збою в роботі програми. У результаті виникнення виключної ситуації програма не може коректно продовжити своє виконання.

Приклади дій у програмі, що можуть призвести до виникнення виключних ситуацій:

* ділення на нуль;
* нехватка оперативної пам’яті при застосуванні оператора new для її виділення (або іншої функції);
* доступ до елементу масиву за його межами (помилковий індекс);
* переповнення значення для деякого типу;
* взяття кореня з від’ємного числа;
* інші ситуації.

У мові C++ виключення – це спеціальний об’єкт класу або значення базового типу, що описує (визначає) конкретну виключну ситуацію і відповідним чином обробляється.

При написанні програми система опису виключних ситуацій вибирається програмістом на власний розсуд. Можна створити свою кваліфікацію помилок, які можуть виникати у програмі. Наприклад, програміст може кваліфікувати різні типи помилок числовим (цілочисельним) значенням або побудувати власну ієрархію класів що описують виключні ситуації. Крім того, можна використовувати можливості класів C++, які є похідними від класу exception.

Мова програмування C++ дає можливість перехоплювати виключні ситуації та відповідним чином їх обробляти.

Механізм перехоплення виключень C++ дозволяє генерувати виключення у тому місці, в якому воно виникає – це є дуже зручно. Не потрібно “видумувати” власні способи обробки виключень, які виникли у функціях нижчого рівня, для того щоб передати їх у функції вищого рівня.

Для перехоплення та обробки виключних ситуацій у мові C++ введено конструкцію try…catch, яка має наступну загальну форму:

*try {*

*// тіло блоку try*

*// ...*

*// генерування виключення оператором throw*

*}*

*catch(*type1 argument1*)*

*{*

*// тіло блоку catch*

*}*

*catch(*type2 argument2*)*

*{*

*// тіло блоку catch*

*}*

*...*

*catch(*typeN *argumentN)*

*{*

*// тіло блоку catch*

*}*

де type1, type2, …, typeN – відповідно тип аргументів argument1, argument2, …, argumentN.

Код, який потрібно проконтролювати, повинен виконуватись всередниі блоку try. Виключні ситуації перехоплюються оператором catch, який слідує безпосередньо за блоком try в якому вони виникли.

У блоці try можуть бути розміщені оператори і функції. Якщо у блоці try генерується відповідна виключна ситуація, то вона перехоплюється відповідним блоком catch. Вибір того чи іншого блоку catch здійснюється в залежності від типу виключної ситуації. Після виникнення виключної ситуації певного типу, викликається блок catch з таким самим типом аргументу. Аргумент приймає деяке значення, яке відповідним чином обробляється (виводиться на екран повідомлення про помилку тощо).

Якщо у блоці try виникне виключна ситуація, яка не передбачена блоком catch, то викликається стандартна функція terminate(), яка за замовчуванням викликає функцію abort(). Ця стандартна функція припиняє виконання програми.

Щоб у блоці try згенерувати виключну ситуацію, потрібно використати оператор throw. Оператор throw може бути викликаний всередині блоку try або всередині функції, яка викликається з блоку try.

Загальна форма оператора throw наступна

throw виключення;

У результаті виконання оператора throw генерується виключення деякого типу. Це виключення повинно бути оброблене в блоці catch.

*Приклад 33.1.* Демонструється використання блоку try…catch для обробки виразу:

У даному виразі у трьох випадках може виникнути виключна ситуація:

* корінь з від’ємного числа **a**, якщо **a**<0;
* корінь з від’ємного числа **b**, якщо **b**<0;
* ділення на 0, якщо **b**=0.

Тому, у блоці try…catch потрібно обробити ці три випадки.

Текст програми типу Console Application наступний:

*#include <iostream>*

*using namespace std;*

*void main()*

*{*

*// обробка виразу sqrt(a)/sqrt(b)*

*double a, b;*

*cout << "a = ";*

*cin >> a;*

*cout << "b = ";*

*cin >> b;*

*double c;*

*try {*

*// початок блоку try*

*if (b == 0)*

*throw 1;*

*if (b < 0)*

*throw 2;*

*if (a < 0)*

*throw 2;*

*c = sqrt(a) / sqrt(b);*

*cout << "c = " << c << endl;*

*}*

*catch (int e) // перехоплення помилки*

*{*

*if (e == 1)*

*cout << "Division by 0." << endl;*

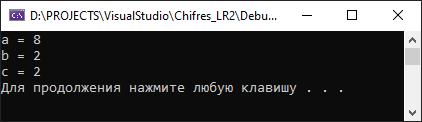
*if (e == 2)*

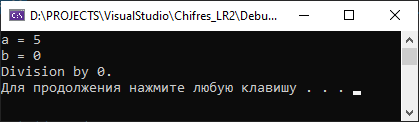
*cout << "Negative root." << endl;*

*}*

*}*

Результати роботи програм наведений нижче:





Після застосування блоку try…catch робота програми не припиняється.

*Приклад 33.2.* Інший варіант обробки виразу з прикладу 1. Тут блок try…catch містить два оператори catch.

*#include <iostream>*

*using namespace std;*

void main()

{

setlocale(LC\_ALL, "Russian");

// обробка виразу sqrt(a)/sqrt(b) - варіант 2

double a, b;

cout << "a = ";

cin >> a;

cout << "b = ";

cin >> b;

double c;

string s;

try { // початок блоку try

if (b == 0)

throw "Division by 0.";

if (b < 0)

throw "Negative root.";

if (a < 0)

throw "Negative root.";

// якщо виключних ситуацій немає, то продовжити обчислення

c = sqrt(a) / sqrt(b);

cout << "c = " << c << endl;

}

catch (int e) // перехоплення помилки типу int

{

if (e == 1)

cout << "Division by 0." << endl;

if (e == 2)

cout << "Negative root." << endl;

}

catch (const char\* e) // перехоплення помилки типу const char\*

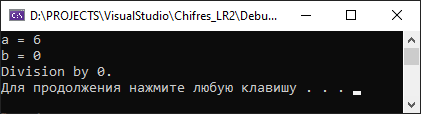
{

cout << e << endl;

}

}

Результат роботи програми наведений нижче:



*Приклад 33.3.* Написати функцію обчислення значення за заданим рядком символів, що є записом цього числа в десятковій системі числення. Передбачити випадок виходу за межі діапазону, що визначається типом int. Необхідно використати механізм виключень.

Текст програми для додатку типу Console Application наступний

using namespace std;

int StrToInt(const char\* str)

{

char s[20];

int t, i;

long res = 0; // результат повернення з функції

int len = strlen(str);

try {

t = 1;

if (str[0] == '-') t = -1; // перевірка, чи перший символ '-'

// цикл конвертування рядка в число типу int

i = len - 1;

while (i >= 0)

{

if (str[i] == '-')

{

if (i == 0) break; // якщо перший символ, то все добре

else throw "Bad position of minus.";

}

// якщо в рядку недопустимі символи, то згенерувати виключення

if (str[i] < '0') throw "Bad symbols";

if (str[i] > '9') throw "Bad symbols";

res = res + (str[i] - '0') \* t;

t \*= 10;

i--;

}

// якщо результат виходить за межі діапазону для 32-розрядних

// цілочисельних значень, то згенерувати відповідне виключення

if (res > INT32\_MAX)

throw "Out of range.";

if (res < INT32\_MIN)

throw "Out of range.";

return res;

}

catch (const char\* e)

{

cout << e << endl;

return 0;

}

}

void main()

{

setlocale(LC\_ALL, "Russian");

int d;

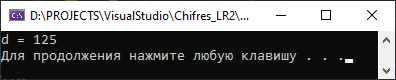
d = StrToInt("125");

cout << "d = " << d<<endl;

system("pause");

}

Результат роботи програми наведений нижче:



Вищенаведена програма може бути переписана так, що блок try…catch розміщується у функції main(), як показано нижче

*// Функція обчислення значення за заданним рядком символів*

#include <iostream>

#include <cstdlib>

#include <math.h>

#include <cstdlib>

#include <new>

using namespace std;

int StrToInt2(const char\* str)

{

char s[20];

int t, i;

long res = 0; // результат роботи функції

int len = strlen(str);

t = 1;

if (str[0] == '-') t = -1;

i = len - 1;

while (i >= 0)

{

if (str[i] == '-')

{

if (i == 0) break; // якщо перший символ, то все добре

else throw "Bad position of minus."; // інакше, згенерувати виключення

}

if (str[i] < '0') throw "Bad symbols";

if (str[i] > '9') throw "Bad symbols";

res = res + (str[i] - '0') \* t;

t \*= 10;

i--;

}

// якщо результат виходить за межі діапазону для 32-розрядних

// цілочисельних значень, то згенерувати відповідне виключення

if (res > INT32\_MAX)

throw "Out of range.";

if (res < INT32\_MIN)

throw "Out of range.";

return res;

}

void main()

{

setlocale(LC\_ALL, "Russian");

int d;

// блок try...catch розміщується у функції main() вищого рівня,

// а виключення генерується у функції StrToInt2() нижчого рівня

try {

d = StrToInt2("19125");

cout << "d = " << d<<endl;

}

catch (const char\* e)

{

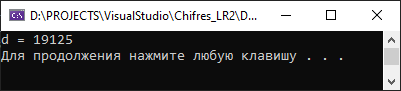
cout << e << endl;

}

system("pause");

}

Результат роботи програми наведений нижче:



Як видно з виденаведеного коду, генерувати виключення оператором throw можна в іншій функції, виклик якої включений у блок try. Отже, функція у своєму тілі може генерувати виключення.

Результат виконання програми

d = 19125

Якщо виклик функції StrToInt2() перенести за межі оператора try

*void main()*

*{*

*int d;*

*try {*

*//d = StrToInt2("19125");*

*//cout << "d = " << d;*

*}*

*catch (const char\* e)*

*{*

*cout << e << endl;*

*}*

*// виклик функції за межами оператора try*

*d = StrToInt2("у19125");*

*}*

то виключні ситуації у функції StrToInt2() оброблятись не будуть. При виникненні виключної ситуації у функції StrToInt2() компілятор згенерує власну помилку

Exception Unhandled

що означає, що виключення необроблене

*Приклад 33.4.* У прикладі, у функції нижнього рівня GenerateException() генерується виключення типу const char\*. Функція перевіряє допустимі межі вхідного параметру index.

У функції верхнього рівня ProcessException() відбувається виклик функції GenerateException(). Цей виклик взятий в блок try.

Текст програми наступний:

*// Приклад Програми, яка генерує виключення в одній функції, а перехоплює його в іншій*

*// Функція згенерує виключення "Out of index",*

*// якщо значення index знаходиться за межами діапазону 0..9*

void GenerateException(int index)

{

if ((index < 0) || (index > 9))

throw "Out of index";

}

// Функція, яка перехоплює виключення "Out of index"

void ProcessException()

{

int index;

cout << "index = ";

cin >> index;

// 1. Викликати виключну ситуацію без обробки,

// компілятор видасть повідомлення "Exception unhandled"

// GenerateException(-3);

// 2. Викликати виключну ситуацію з обробкою блоком try...catch

try {

GenerateException(index); // виклик функції, яка генерує виключення

cout << "OK!" << endl; // якщо index в межах 0..9, то OK!

}

catch (const char\* e)

{

cout << "Error: " << e << endl;

}

}

void main()

{

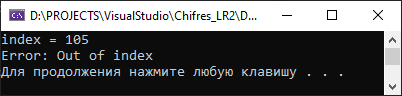
setlocale(LC\_ALL, "Russian");

ProcessException();

system("pause");

}

Результат роботи програми наведений нижче:



*Приклад 33.5.* Бувають випадки, коли потрібно перехватити усі виключні ситуації підряд. Для цього, в C++ використовується блок catch(…), який має таку загальну форму

*catch(...)*

*{*

*// Обробка усіх виключних ситуацій*

*// ...*

*}*

У прикладі демонструється використання блоку catch(…) для обробки ситуацій будь-якого типу.

У програмі реалізовано:

* функція DivNumbers(), яка повертає результат ділення двох чисел, введених з клавіатури. У функції генерується виключення типу int, якщо значення дільника рівне 0;
* функція SqRoot(), яка повертає корінь з від’ємного числа. У функції генерується виключення типу const char\*, якщо значення параметру number від’ємне;
* функція ProcessException(). Ця функція демонструє роботу функцій DivNumbers() та SqRoot(). У функції використовується інструкція try…catch().

*// Приклад. Демонстрація використання блоку catch*

double SqRoot(double number)

{

if (number < 0) throw "Negative number";

return sqrt(number);

}

double DivNumbers(double a, double b)

{

if (b == 0) throw 1;

return a / b;

}

void ProcessException()

{

double v;

while (1)

{

cout << "Input a function to call (1-2, 3-exit): " << endl;

cout << "1-DivNumbers. 2-SqRoot" << endl;

cout << ">>";

cin >> v;

try {

if (v == 1) // функція DivNumbers

{

double a, b;

cout << "DivNumbers(double a, double b)" << endl;

// ввести a, b

cout << "a = "; cin >> a;

cout << "b = "; cin >> b;

// Викликати функцію DivNumbers()

double c = DivNumbers(a, b);

cout << "c = " << c << endl;

}

if (v == 2)

{

double x, num;

cout << "SqRoot(double num)" << endl;

cout << "num = "; cin >> num;

// Викликати функцію SqRoot()

x = SqRoot(num);

cout << "x = " << x << endl;

}

if (v == 3) break;

}

catch (const char\* e)

{

cout << "Error. Text = " << e << endl;

}

catch (...) // усі інші типи виключень

{

cout << "Error in block catch(...)." << endl;

}

}

}

void main()

{

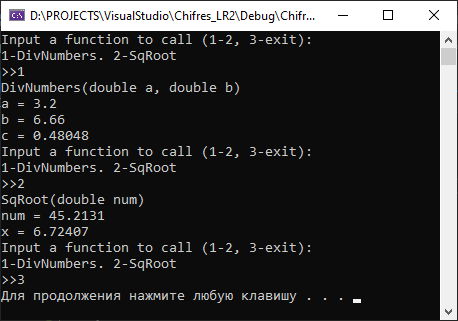
setlocale(LC\_ALL, "Russian");

ProcessException();

system("pause");

}

Результат роботи програми наведений нижче:



Як видно з тексту функції ProcessException() виклик функцій DivNumbers() та SqRoot() взято в блок try…catch

*// Викликати різні варіанти функцій*

*try {*

*...*

*}*

*catch (const char\* e)*

*{*

*cout << "Error. Text = " << e << endl;*

*}*

*catch (...) // усі інші типи виключень*

*{*

*cout << "Error in block catch(...)." << endl;*

*}*

У блоці try…catch обробляються

* виключення типу const char\*;
* усі інші види виключень. У цьому випадку використовується інструкція catch(…).

Результат роботи програми

Input a function to call (1-2, 3-exit):

1-DivNumbers.   2-SqRoot

>>2

SqRoot(double num)

num = -4

Error. Text = Negative number

Input a function to call (1-2, 3-exit):

1-DivNumbers.   2-SqRoot

>>1

DivNumbers(double a, double b)

a = 3

b = 0

Error in block catch(...).

Input a function to call (1-2, 3-exit):

1-DivNumbers.   2-SqRoot

>>1

DivNumbers(double a, double b)

a = 2

b = 5

c = 0.4

Input a function to call (1-2, 3-exit):

1-DivNumbers.   2-SqRoot

>>3

***Порядок виконання роботи:***

1Розглянути приклади, наведені у теоретичних відомостях.

2Додати обробку виключень при написанні наступних функцій:

* + 1. функції обчислюють периметр: трикутника, прямокутника, квадрату, кругу.
    2. функція переводить години і хвилини у секунди.

**Лабораторна робота №34**

## **Шаблонні функції (template functions)**

*Мета роботи*: набуття навичок роботи з шаблоними функціями.

*Теоретичні основи*:

Поняття шаблону у функції має на увазі використання узагальненого типу даних в якості вхідних та вихідних параметрів функції. Замість конкретного типу даних у своїй роботі функція використовує деякий узагальнений тип, який має узагальнене ім’я, наприклад T.

*Приклад 34.1.* Загальна форма узагальненої функції має вигляд:

*template <class T>* return*\_type* ***FunName*** *(*list\_of\_parameters*)*

*{*

*// ...*

*}*

де

* return\_type – тип, що повертає функція;
* Fun\_Name – ім’я шаблонної функції;
* list\_of\_parameters – параметри шаблонної функції, що мають узагальнений тип T.

При оголошенні шаблонної функції, замість ключового слова class можна застосовувати ключове слово typename. У цьому випадку загальна форма узагальненої функції матиме вигляд:

*template <typename T>* return*\_type* ***FunName****(*list\_of\_parameters*)*

*{*

*// ...*

*}*

*Приклад 34.2.* Оголошення функції, що отримує вхідним параметром змінну узагальненого типу T та повертає значення узагальненого типу T має вигляд:

*template <class T>*

*T* **FunName**(*T* parameterName*)*

*{*

*// тіло функції*

*// ...*

*}*

де

* FunName – ім’я узагальненої функції;
* T – ім’я узагальненого типу. При оголошенні шаблонної функції, це ім’я може бути будь-яким, наприклад TYPE;
* parameterName – ім’я параметру узагальненого типу T, який використовується у функції FunName.

*Приклад 34.3.* У прикладі реалізовано шаблонну функцію Add(), що додає два параметри узагальненого типу TT. У функції main() продемонстровано використання функції Add()

*#include "stdafx.h"*

*#include <iostream>*

*using namespace std;*

*// Шаблонна функція, що повертає суму двох величин типу TT*

*template <class TT>*

*TT Add(TT t1, TT t2)*

*{*

*return t1 + t2;*

*}*

*int main()*

*{*

*// Використання шаблонної функції Add*

*// 1. Для типу double*

*double d1 = 3.85, d2 = 2.50;*

*double d3;*

*d3 = Add(d1, d2); // d3 = 6.35*

*// 2. Для типу int*

*int i1 = 25;*

*int i2 = 13;*

*int i3;*

*i3 = Add(i1, i2); // i3 = 38*

*// 3. Для типу string*

*string s1 = "abc";*

*string s2 = "def";*

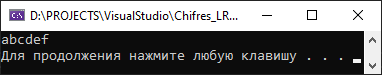
*string s3 = Add(s1, s2); // s3 = "abcdef"*

*cout << s3.c\_str() << endl;*

*return 0;*

*}*

Результат роботи програми наведений нижче:



Як видно з прикладу, шаблонна функція

*template <class TT>*

*TT Add(TT t1, TT t2)*

*{*

*return t1 + t2;*

*}*

використовується для додавання параметрів типів double, int, string. У цій функції узагальнений тип має ім’я TT.

*Приклад 34.4.* Реалізовано шаблонну функцію Equal(), яка порівнює два вхідні параметри узагальненого типу X. Функція повертає true, якщо значення параметрів однакові. В іншому випадку функція повертає false.

*#include "stdafx.h"*

*#include <iostream>*

*using namespace std;*

*// функція, яка порівнює два вхідні параметри типу X*

*template <typename X>*

*bool Equal(X a, X b)*

*{*

*return a==b;*

*}*

*int main()*

*{*

*// демонстрація шаблонної функції Equal()*

*bool res;*

*res = Equal(5, 6); // res = 0*

*res = Equal("abcd", "abcd"); // res = 1*

*res = Equal(5.5, 5.5); // res = 1*

*res = Equal('A', 'B'); // res = 0*

*return 0;*

*}*

Як видно з прикладу, виклик функції Equal() здійснюється для різних типів: int, string, double, char.

Якщо компілятор зустрічає шаблонну функцію, він генерує версії функції для кожного варіанту її застосування з конкретним типом. Тобто, якщо шаблонна функція викликається для типів double, int, string, то компілятор генерує 3 версії функції для кожного типу.

*Приклад 34.5.* Нехай задано функцію, яка додає два параметри узагальненого типу T:

*T Add(T t1, T t2)*

*{*

*return t1 + t2;*

*}*

При наступному виклику такої функції

*int a = Add(3, 8);*

*double d = Add(3.8, 2.9);*

*string s = Add("Hello ", "world!");*

компілятор згенерує такі реалізації функції Add():

*int Add(int t1, int t2)*

*{*

*return t1 + t2;*

*}*

*double Add(double t1, double t2)*

*{*

*return t1 + t2;*

*}*

*string Add(string t1, string t2)*

*{*

*return t1 + t2;*

*}*

Компілятор генерує стільки варіантів функції, скільки існує способів її викликів у програмі.

Якщо потрібно, в шаблонній функції може бути визначено будь-яку кількість узагальнених типів (два, три і т.д.). У цьому випадку узагальнені типи перераховуються через кому з допомогою ключового слова class або typename.

*Приклад 34.6.* Для двох типів з іменами T1, T2 шаблонна функція матиме такий загальний вигляд

*template <class T1, class T2>*

return*\_type* ***FunName****(*list\_of\_parameters*)*

*{*

*// ...*

*}*

де

* return\_type – тип, що повертає функція;
* Fun\_Name – ім’я шаблонної функції;
* list\_of\_parameters – параметри шаблонної функції, що оперує узагальненими типами T1, T2.

*Приклад 34.7.* Нижче наведено приклад шаблонної функції, яка використовує два узагальнені типи T1, T2. Функція отримує два параметри типів T1, T2 і виводить значення цих параметрів на екран. Для оголошення типів використовується ключове слово typename. Можна також використовувати ключове слово class.

#include <iostream>

#include <cstdlib>

#include <math.h>

#include <cstdlib>

#include <new>

using namespace std;

template <typename T1, typename T2>

void Display(T1 t1, T2 t2)

{

cout << "t1 = " << t1 << "; t2 = " << t2 << endl;

}

int main()

{

int a = 30;

double d = 9.83;

char c = 'X';

bool b = true;

Display(a, d); // t1 = 30; t2 = 9.83

Display(b, c); // t1 = 1; t2 = X

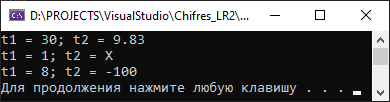
Display(8, -100); // t1 = 8; t2 = -100

system("pause");

return 0;

}

Результат роботи програми наведений нижче:



Явне “перевантаження” (explicit specialization) шаблонної функції – це оголошення ще однієї функції з таким самим іменем, але вже для конкретного типу (наприклад, int, double). Таким чином, “перевантажена” функція заміщає шаблонну (узагальнену) функцію для деякого конкретного випадку (типу даних). Для інших випадків викликається узагальнений варіант шаблонної функції.

*Приклад 34.8.* У прикладі шаблонна функція Display() “перевантажена” для типу string. Таке “перевантаження” є логічним, оскільки в операторі cout вивести тип string звичайним рядком

cout << "t = " << t << endl;

не вдасться. У цьому випадку вийде помилка компіляції, оскільки для типу string потрібно перетворення t.c\_str() у тип const char \*. Таке перетворення реалізоване у “перевантаженій” функції Display(), що отримує параметром змінну t типу string.

#include <iostream>

#include <cstdlib>

#include <math.h>

#include <cstdlib>

#include <new>

using namespace std;

template <class T>

void Display(T t)

{

cout << "t = " << t << endl;

}

// явно "перевантажена" функція Display() для типу string

void Display(string t)

{

// вивід типу string виділений окремою функцією

cout << "t = " << t.c\_str() << endl;

}

int main()

{

int a = 10;

double d = 3.55;

string s = "Hello";

// виклик шаблонної функції

Display(a); // t = 10

Display(d); // t = 3.55

// виклик перевантаженої функції для типу string

Display("abcd"); // t = abcd

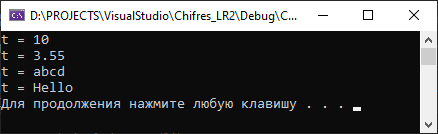
Display(s); // t = Hello

system("pause");

return 0;

}

Результат роботи програми наведений нижче:



“Перевантаження” шаблонної функції – це використання того самого імені для шаблонної функції, але з іншою кількістю параметрів.

*Приклад 34.9.* Нижче наведено “перевантаження” шаблонної функції Max(), що знаходить максимальне значення між двома, трьома та чотирма величинами узагальненого типу T.

#include <iostream>

#include <cstdlib>

#include <math.h>

#include <cstdlib>

#include <new>

using namespace std;

template <typename T>

T Max(T value1, T value2)

{

if (value1 > value2) return value1;

return value2;

}

// "перевантажена" шаблонна функція Max()

// знаходить максимум між трьома значеннями value1, value2, value3

template <class T>

T Max(T value1, T value2, T value3)

{

T max = value1;

if (max < value2) max = value2;

if (max < value3) max = value3;

return max;

}

// "перевантажена" шаблонна функція Max()

// максимум між чотирма значеннями

template <typename T>

T Max(T value1, T value2, T value3, T value4)

{

T max = value1;

if (max < value2) max = value2;

if (max < value3) max = value3;

if (max < value4) max = value4;

return max;

}

int main()

{

// демонстрація використання "перевантаженої" шаблонної функції Max()

int i;

double d;

bool b;

char c;

// для типу int

i = Max(8, 11); // i = 11

cout << i << endl;

i = Max(9, 13, 14); // i = 14

cout << i << endl;

i = Max(8, 10, 4, -3); // i = 10

cout << i << endl;

// для типу double

d = Max(3.5, 1.2); // d = 3.5

cout << d << endl;

d = Max(1.3, -9.20, 100.33); // d = 100.33

cout << d << endl;

d = Max(-39.5, 8.82, 1.23); // d = 8.82

cout << d << endl;

// для типу bool

b = Max(true, false); // b = 1

cout << b << endl;

// для типу char

c = Max('C', 'F'); // c = 'F'

cout << c << endl;

c = Max('F', 'H', 'G'); // c = 'H'

cout << c << endl;

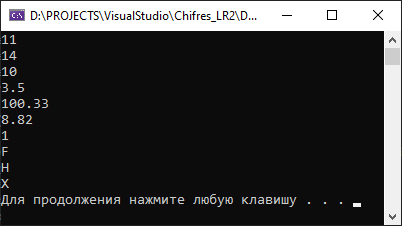
c = Max('M', 'A', 'X', 'W'); // c = 'X'

cout << c << endl;

system("pause");

return 0; }

Результат роботи програми наведений нижче:



У шаблонній функції дозволяється використовувати стандартні параметри. Ці параметри оголошуються так само, як і для звичайної нешаблонної функції.

*Приклад 34.10.* Реалізується функція Mult(), яка додає задану кількість разів значення узагальненого параметру типу T. Кількість додавань задається цілим числом count. Функція повертає результат, що є сумою параметрів типу T.

У прикладі демонструється використання функції для типів int та double. Для інших типів потрібно визначати “перевантажені” реалізації функції Mult().

#include <iostream>

#include <cstdlib>

#include <math.h>

#include <cstdlib>

#include <new>

using namespace std;

template <class T>

T Mult(T value, int count)

{

T res = value;

for (int i = 1; i < count; i++)

{

res = res + value;

}

return res;

}

int main()

{

// демонстрація шаблонної функції Mult()

int i;

double d;

string s;

// для типу int

i = Mult(5, 2); // i = 10

cout << i << endl;

i = Mult(6, 4); // i = 24

cout << i << endl;

// для типу double

d = Mult(3.5, 3); // d = 10.5

cout << d << endl;

d = Mult(6.2, 8); // d = 49.6

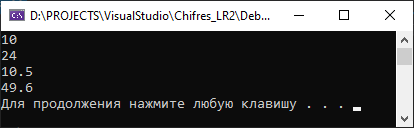
cout << d << endl;

system("pause");

return 0;

}

Результат роботи програми наведений нижче:



*Приклад 34.11.* У прикладі реалізована шаблонна функція Sort(), що сортує масив елементів деякого узагальненого типу T. Функція отримує параметром посилання на масив та кількість елементів масиву. У функції main() демонструється застосування шаблонної функції Sort().

#include <iostream>

#include <cstdlib>

#include <math.h>

#include <cstdlib>

#include <new>

using namespace std;

template <class T>

void Sort(T A[], int count)

{

int i, j;

T tmp;

for (i = 0; i < count - 1; i++)

for (j = i; j >= 0; j--)

if (A[j] < A[j + 1])

{

// обміняти значеннями A[i] та A[i+1]

tmp = A[j];

A[j] = A[j + 1];

A[j + 1] = tmp;

}

}

int main()

{

// демонстрація шаблонної функції Sort()

int M[] = { 3, 5, -1, 2, -9, 7, 8, 3 };

double D[] = { 3.3, 2.8, 1.5, 4.8, 6.1, 1.3 };

// для типу int

Sort(M, 8); // M = { }

for (int i = 0; i < 8; i++)

cout << M[i] << ", ";

cout << endl << endl;

// для типу double

Sort(D, 6);

for (int i = 0; i < 6; i++)

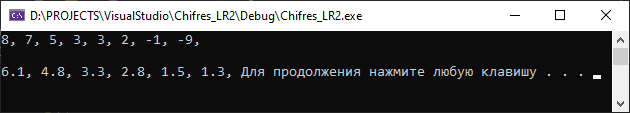
cout << D[i] << ", ";

system("pause");

return 0;

}

Результат роботи програми наведений нижче:



***Порядок виконання роботи:***

Розглянути приклади, наведені у теоретичних відомостях.

**Лабораторна робота №35**

## **Поняття шаблону класу. Ключове слово template. Переваги використання шаблонів. Аргументи в шаблонах**

*Мета роботи*: набуття навичок роботи з шаблоними класами.

*Теоретичні основи*:

Часто, при розробці класів для різних типів даних, програмісту доводиться писати програмний код для кожного типу окремо. Методи та операції над даними різних типів можуть містити той самий повторюваний код. Щоб уникнути повторюваності написання коду для різних типів даних, у мові C++ використовуються так звані шаблони.

Шаблон класу дозволяє оперувати даними різних типів у загальному. Тобто, немає прив’язки до деякого конкретного типу даних (int, float, …). Вся робота виконується над деяким узагальненим типом даних, наприклад типом з іменем T.

Фактично, оголошення шаблону класу є тільки описом. Створення реального класу з заданим типом даних здійснюється компілятором в момент компіляції, коли оголошується об’єкт класу.

У найпростішому випадку загальна форма оголошення шаблону класу без аргументів має такий вигляд:

*template <class T> class ClassName*

*{*

*// тіло класу*

*// ...*

*}*

де

* T – узагальнене ім’я типу, що використовується методами класу;
* ClassName – ім’я класу, що містить методи оперування узагальненим типом класу.

Загальна форма оголошення об’єкту шаблонного класу має такий вигляд:

*ClassName <***type***>* objName*;*

де

* ClassName – ім’я шаблонного класу;
* Type – конкретний тип даних у програмі;
* objName – ім’я об’єкту (екземпляру) класу.

Ключове слово class може бути замінене ключовим словом typename. У цьому випадку загальна форма оголошення шаблонного класу матиме такий вигляд:

*template <typename T>*

*class ClassName*

*{*

*// тіло класу*

*// ...*

*}*

У даному випадку немає різниці між використанням слів class та typename.

Оголошення шаблону класу дає такі переваги:

* уникнення повторюваності написання програмного коду для різних типів даних. Програмний код (методи, функції) пишеться для деякого узагальненого типу T;
* зменшення текстової частини програмного коду, і, як наслідок, підвищення читабельності програм;
* забезпечення зручного механізму передачі аргументів у шаблон класу з метою їх обробки методами класу.

*Приклад 35.1.* Нехай потрібно оголосити шаблон класу, що буде обробляти деяке число. Число може бути будь-якого типу, що дозволяє виконувати над ним арифметичні операції.

У прикладі оголошується шаблон класу, що містить методи, які виконують наступні операції над деяким числом:

* множення числа на 2;
* ділення одного числа на інше. Для цілих типів виконується ділення націло;
* взяття квадрату числа (степінь 2).

Оголошення шаблону має вигляд

#include <iostream>

#include <cstdlib>

#include <math.h>

#include <cstdlib>

#include <new>

using namespace std;

template <class T>

class MyNumber

{

public:

// конструктор

MyNumber(void) { }

// метод, що множить на 2 число

void Mult2(T\* t);

// метод, що повертає квадрат числа для деякого типу T

T MySquare(T);

// метод, що ділить два числа типу T і повертає результат типу T

T DivNumbers(T, T);

};

// реалізація методу, що множить число на 2

template <class T> void MyNumber<T>::Mult2(T\* t)

{

\*t = (\*t) \* 2;

}

// реалізація методу, що повертає квадрат числа

template <class T> T MyNumber<T>::MySquare(T number)

{

return (T)(number \* number);

}

// метод, що ділить 2 числа і повертає результат від ділення

template <class T> T MyNumber<T>::DivNumbers(T t1, T t2)

{

return (T)(t1 / t2);

}

int main()

{

MyNumber <int> mi; // об'єкт mi класу працює з типом int

MyNumber <float> mf; // об'єкт mf працює з типом float

int d = 8;

float x = 9.3f;

// множення числа на 2

mi.Mult2(&d); // d = 16

cout << d << "\n";

mf.Mult2(&x); // x = 18.6

cout << x << "\n";

// взяття квадрату з числа

int dd;

dd = mi.MySquare(9); // dd = 81 - ціле число

cout << dd << "\n";

double z;

z = mf.MySquare(1.1); // z = 1.21000... - дійсне число

cout << z << "\n";

// ділення чисел

long int t;

float f;

t = mi.DivNumbers(5, 2); // t = 2 - ділення цілих чисел

cout << t << "\n";

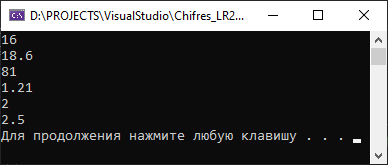
f = mf.DivNumbers(5, 2); // f = 2.5 - ділення дійсних чисел

cout << f << "\n";

system("pause");

return 0; }

Результат роботи програми наведений нижче:



Бувають випадки, коли в шаблоні класу потрібно використовувати деякі аргументи. Ці аргументи можуть використовуватись методами, що описуються в шаблоні класу.

Загальна форма шаблону класу, що містить аргументи, наступна:

*template <class T,* **type1**var1, **type2** var2, ..., ***typeN*** varN*> class ClassName*

*{*

*// тіло шаблону класу*

*// ...*

*}*

де

* T – деякий узагальнений тип даних;
* type1, type2, …,typeN – конкретні типи аргументів з іменами var1, var2, …, varN;
* var1, var2, …, varN – імена аргументів, що використовуються у шаблоні класу.

Загальна форма оголошення об’єкту шаблонного класу, який містить один аргумент:

ClassName <**type**, arg> objName;

де

* ClassName – ім’я шаблонного класу;
* Type – конкретний тип даних, для якого формується реальний клас;
* Arg – значення аргументу, яке використовується у шаблоні класу;
* objName – ім’я об’єкту шаблонного класу.

*Приклад 35.2.* У прикладі реалізується шаблон класу CMyArray, що містить методи обробки масиву чисел. Тип елементів масиву може бути дійсним або цілим.

Шаблон класу отримує два цілих числа:

* count – кількість елементів масиву. Використовується при ініціалізації класу з допомогою конструктора з 1 параметром;
* num – число, що служить для проведення операцій над масивом.

Ці числа використовуються у методах для виконання операцій над масивом. Шаблон класу містить такі дані та методи:

* кількість елементів масиву n;
* масив елементів (чисел) A заданої розмірності (10);
* метод Power(), що здійснює піднесення елементів масиву A у ступінь num, що є вхідним параметром (аргументом);
* метод CalcNum(), що здійснює підрахунок кількості елементів, що є більше заданого параметру num.

Текст шаблону класу наступний:

*// шаблон класу, що отримує 2 параметри*

#include <iostream>

#include <cstdlib>

#include <math.h>

#include <cstdlib>

#include <new>

using namespace std;

template <class TT, int count, int num> class CMyArray

{

private:

int n; // кількість елементів масиву

TT A[10]; // масив елементів

public:

// конструктор класу без параметрів

CMyArray()

{

// кількість елементів беремо з вхідного параметра count

n = count;

// заповнити масив довільними значеннями

for (int i = 0; i < n; i++)

A[i] = (TT)(i \* 2);

}

// конструктор класу з 1 параметром

CMyArray(int cnt)

{

if (cnt <= 10) n = cnt;

else n = 0;

// заповнення масиву довільними значеннями

for (int i = 0; i < n; i++)

A[i] = (TT)(i \* 2);

}

// методи доступу

int GetN(void) { return n; }

void SetN(int n)

{

if (n <= 10) this->n = n;

else n = 0;

for (int i = 0; i < n; i++)

A[i] = (TT)(i \* 2);

}

// метод, що читає комірку масиву з заданим індексом

TT GetItem(int index) { return (TT)A[index]; }

// методи, що виконують операції над масивом A

// піднесення елементів масиву до степеня num

void Power(void);

// підрахунок кількості елементів, що є більше заданого аргументу

int CalcNum(void);

};

// метод, що підносить елементи масиву до степеня num

template <class TT, int count, int num>

void CMyArray<TT, count, num>::Power(void)

{

if (n < 0) return;

for (int i = 0; i < n; i++)

A[i] = System::Math::Pow(A[i], num);

}

// метод, що визначає кількість елементів масиву,

// які більше заданого числа num (num - вхідний параметр)

template <class TT, int count, int num>

int CMyArray<TT, count, num>::CalcNum(void)

{

int k = 0;

// цикл підрахунку

for (int i = 0; i < n; i++)

if (A[i] > num)

k++;

return k;

}

int main()

{

// використання шаблону класу CMyArray

// масив цілих чисел, параметри: count=7, num=2

CMyArray <int, 7, 2> ai1;

// масив цілих чисел, виклик конструктора з 1 параметром

CMyArray <int, 8, -3> ai2(6); // к-сть елементів count = 6, num=-3

// масив дійсних чисел типу double, виклик конструктора без параметрів

CMyArray <double, 4, 5> ad1;

// перевірка

int n, t;

double x;

n = ai1.GetN(); // n = 7

cout << n << "\n";

n = ai2.GetN(); // n = 6

cout << n << "\n";

n = ad1.GetN(); // n = 4

cout << n << "\n";

// перевірка масиву

t = ai1.GetItem(3); // t = 6

cout << t << "\n";

t = ai2.GetItem(0); // t = 0

cout << t << "\n";

x = ad1.GetItem(2); // x = 4.0

cout << x << "\n";

// виклик методів обробки масиву та перевірка результату

ai1.Power(); // піднесення кожного ел-ту масиву до степеня num=2

t = ai1.GetItem(3); // t = 6^2 = 36

cout << t << "\n";

// підрахунок к-сті елементів, які більше за -3

// всього у масиві класу ai2 6 елементів

t = ai2.CalcNum(); // t = 6

cout << t << "\n";

// робота з класом, що обробляє тип double

x = ad1.GetItem(3); // x = 6.0

cout << x << "\n";

ad1.Power(); // піднести усі числа масиву до степеня num = 5

x = ad1.GetItem(3); // x = 6.0^5 = 7776

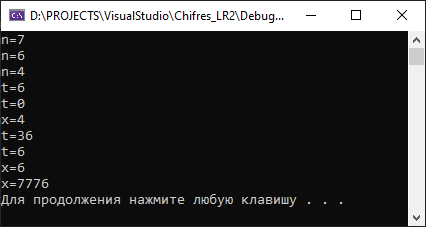
cout << x << "\n";

system("pause");

return 0;

}

Результат роботи програми наведений нижче:



***Порядок виконання роботи:***

Розглянути приклади, наведені у теоретичних відомостях.

**Лабораторна робота №36**

## **Віртуальні методи. Абстрактні класи**

*Мета роботи*: набуття навичок роботи з віртуальними методами.

*Теоретичні основи*:

Віртуальні методи існують для того, щоб «спадкоємець» поводився відмінно від «предка», зберігаючи при цьому властивість сумісності з ним.

***Віртуальний метод*** - це метод, який, будучи описаний у нащадках, заміщає собою відповідний метод скрізь, навіть в методах, описаних для предка, якщо він викликається для нащадка.

Адреса віртуального методу відома тільки у момент виконання програми. Коли відбувається виклик віртуального методу, його адреса береться з таблиці віртуальних методів свого класу. Таким чином викликається те, що треба.

Перевага застосування віртуальних методів полягає в тому, що при цьому використовується саме механізм пізнього зв'язування, який допускає обробку об'єктів, тип яких невідомий під час компіляції.

*Приклад 36.1.*

#include <iostream>

#include <cstdlib>

#include <math.h>

#include <cstdlib>

#include <new>

using namespace std;

class vehicle // клас "транспортний засiб"

{

int wheels;

float weight;

public: // початок публiчного(вiдкритого) роздiлу класу

virtual void message(void) { cout << "Транспортний засiб\n"; } // опис

// вiртуальної функцiї message класу vehicle i реалiзацiя цiй функцiї.

// При виклику функцiї message класу vehicle на екран монiтора

// буде виведений рядок "Транспортний засiб"

};

class car : public vehicle // клас "легкова машина", успадкований з класу

// "транспортний засiб"

{

int passenger\_load;

public: // початок публiчного(вiдкритого) роздiлу класу

void message(void) { cout << "Легкова машина\n"; } // опис вiртуальної функцiї

// message класу car i реалiзацiя цiй

// функцiї. При виклику функцiї message класу car на екран монiтора

// буде виведений рядок " Легкова машина "

};

class truck : public vehicle // клас "вантажна машина", успадкований з

// класу "транспортний засiб"

{

int passenger\_load;

float payload;

public: // початок публiчного(вiдкритого) роздiлу класу

int passengers(void) { return passenger\_load; }

};

class boat : public vehicle // клас "човен", успадкований класу

// "транспортний засiб"

{

int passenger\_load;

public: // початок публiчного(вiдкритого) роздiлу класу

int passengers(void) { return passenger\_load; }

void message(void) { cout << "Човен\n"; }// опис вiртуальної функцiї

// message класу boat i реалiзацiя цiй функцiї.

// При виклику функцiї message класу boat на екран монiтора

// буде виведений рядок "Човен"

};

void main() // основний виконуваний блок програми

{

setlocale(LC\_ALL, "Russian");

vehicle\* unicycle; // описуємо змiннiй unicycle як покажчик на

// об'єкт класу vehicle

unicycle = new vehicle; // Створюємо об'єкт класу vehicle,

// покажчик unicycle вказує на цей об'єкт

unicycle->message(); // викликаємо метод message об'єкту

delete unicycle; // видаляємо об'єкт unicycle

// Усi подальшi блоки по 3 рядки абсолютно iдентичнi першому

// блоку з тiєю лише рiзницею, що змiнюється клас створюваного об'єкту

// на car, truck, boat

unicycle = new car;

unicycle->message();

delete unicycle;

unicycle = new truck;

unicycle->message();

delete unicycle;

unicycle = new boat;

unicycle->message();

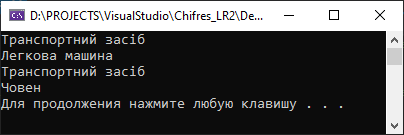
delete unicycle;

//cout << "x=" << x << "\n";

system("pause");

}

Результат роботи програми наведений нижче:



Результати роботи програми(висновок на екран) :

Транспортний засіб

Легкова машина

Транспортний засіб

Човен

Дуже часто клас, що містить віртуальний метод називають поліморфним класом. Найголовніше відмінність полягає в тому, що поліморфні класи допускають обробку об'єктів, тип яких невідомий під час компіляції. Функції, описані у базовому класі як віртуальні, можуть бути модифіковані в похідних класах, причому зв'язування станеться не на етапі компіляції (те, що називається раннім зв'язуванням), а у момент звернення до цього методу (пізнє зв'язування).

Віртуальні методи описуються за допомогою ключового слова **virtual** у базовому класі. Це означає, що в похідному класі цей метод може бути заміщений методом, більше відповідним для цього похідного класу. Оголошений віртуальним у базовому класі, метод залишиться віртуальним для усіх похідних класів. Якщо в похідному класі віртуальний метод не буде перевизначений, то при виклику буде знайдений метод з таким ім'ям вгору за ієрархією класів (тобто у базовому класі).

Дуже часто у базовому класі визначається віртуальна функція, яка не виконує яких-небудь значимих дій. Як я вже згадував при поясненні поняття об'єкту, чим нижче знаходиться клас в сходах ієрархії, тим він конкретніший. Т.е. базовий клас зазвичай не визначає закінчений тип. Дуже часто виходить, що базовий клас не має якої-небудь практичної цінності для його безпосереднього застосування в програмі. Проте, він має цінність для формування похідних класів. Розглянемо наш приклад. Ми використали в програмі створення об'єкту класу **vehicle** тільки для надання прикладу більшої наочності. У реальній же програмі (також як і світі) нам навряд чи згодиться об'єкт класу **vehicle** (транспортний засіб). Нас більше цікавить конкретна реалізація можливостей цього класу, яку нам дають похідні з нього класи **car**, **truck**, **boat**. Розглядаючи роботу функції **message** класу **vehicle**, ми помітимо, що при виконанні нашої програми тільки один раз (якщо не враховувати перший виклик процедури **message**, коли був визначений об'єкт класу **vehicle**) з трьох викликалася функція **message** класу **vehicle**. Якщо ж в клас **truck** додати свою функцію **message**, те реалізація функції **message** у класі **vehicle** не представлятиме ніякої цінності.

Таким чином, ми можемо зробити два дуже важливих на цьому етапі висновки:

1. Клас **vehicle** нам цінний тільки як базовий клас
2. Функція message класу **vehicle** стала потрібна тільки для того, щоб її перевизначили в похідних класах.

Чисто віртуальним методом називають такий метод, який не визначається у базовому класі. У нього немає тіла, а є тільки декларація про його існування.

*Приклад 36.2.* Якщо ми хочемо представити клас **vehicle** у вигляді абстрактного класу, нам необхідно зробити дві речі: додати метод **message** у клас **truck** і прибрати тіло функції у базовому класі **vehicle**. Тоді опис класу **vehicle** виглядатиме таким чином:

*class vehicle*

*{*

*int wheels;*

*float weight;*

*public:*

*virtual void message(void) = 0;*

*};*

Клас, що містить хоч би один чистий віртуальний метод, називається абстрактним класом.

Абстрактні класи не бувають ізольованими, тобто завжди абстрактний клас має бути успадковним. Оскільки у чисто віртуального методу немає тіла, то створити об'єкт абстрактного класу неможливо. Крім того, щоб уникнути появи помилки при виклику чистого віртуального методу, похідний клас повинен містити декларування і тіло чистого віртуального методу.

Абстрактним класом можна назвати клас, спеціально визначений для забезпечення спадкоємства характеристик породженими класами.

***Порядок виконання роботи:***

Розглянути приклади, наведені у теоретичних відомостях.