Міністерство освіти і науки України
Промислово-економічний коледж
Національного авіаційного університету

В. В. Левченко

ОБ'ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ

методичні вказівки до виконання лабораторних робіт
для молодших спеціалістів денної форми навчання
спеціальності 5.05010301 "Розробка програмного забезпечення"

Модуль І – Об'єктно-орієнтоване програмування в С++

м. Київ
2010

Міністерство освіти і науки України

Промислово-економічний коледж

Національного авіаційного університету

ЗАТВЕРДЖУЮ

Заступник директора ПЕК НАУ

з навчальної роботи

_____Л. В. Тандир

“___”_____2010 р.

ОБ’ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ

методичні вказівки до виконання лабораторних робіт

для молодших спеціалістів денної форми навчання

спеціальності 5.05010301 “Розробка програмного забезпечення”

Модуль І

Розглянуто на засіданні ЦМК

“Програмування та обчислювальної техніки”

Протокол №___ від “___”_____2010 р.

Склав викладач:

_____В. В. Левченко

“___”_____2010 р.

Голова ЦМК

_____В. В. Левченко

“___”_____2010 р.

м. Київ

2010

Левченко В. В.

Об’єктно-орієнтоване програмування: методичні вказівки до виконання лабораторних робіт для молодших спеціалістів денної форми навчання спеціальності 5.05010301 “Розробка програмного забезпечення”. Модуль І. – К.: ПЕК НАУ, 2010. – 82 с.

Методичні вказівки містять базові відомості з теорії та практики об’єктно-орієнтованого програмування мовою C++ в середовищах CodeGear RAD Studio 2009/2010 та Microsoft Visual Studio 2008/2010, рекомендації щодо виконання лабораторних робіт, завдання та питання для самоперевірки. Призначені для молодших спеціалістів денної форми навчання спеціальності 5.05010301 “Розробка програмного забезпечення”, які вивчають дисципліну “Об’єктно-орієнтоване програмування”.

© В. В. Левченко, 2010.

ЗМІСТ

Вітаю!

Завершився другий навчальний курс, а це означає, що ви повністю оволоділи основами програмування. Не вірите? Давайте згадаємо, що саме ви дізналися в результаті вивчення основ програмування мовою C/C++: структуру програми, типи даних, оголошення змінних і констант, правила побудови арифметичних і логічних виразів, засоби вводу і виводу інформації, керуючі конструкції (умовні і циклові оператори), засоби динамічного розподілу пам'яті, засоби для роботи з файлами, складні типи даних (масиви, структури тощо) і багато іншого. Але головне, що за рік ви навчились *створювати та реалізовувати алгоритми розв'язання різних класів задач*. Саме це вміння, а не енциклопедичні знання складових елементів мови програмування визначає рівень професіоналізму програміста!

Якщо навіть ви не згадали чогось з переліченого – не лякайтесь, а візьміть свою "настільну" книгу по C++ і ліквідуйте всі "білі плями". Адже вивчення дисциплін, які чекають на вас попереду, потребує відмінного знання алгоритмічного та програмного базису, який закладався дисципліною "Основи програмування".

"Почекайте, – скажете ви, – але що вивчати далі? Хіба що іншу мову програмування?" Справа в тому, що для *професійного* програмування недостатньо знати одну або навіть декілька мов програмування. Надзвичайно важливе (а може й найважливіше!) значення мають підходи, методи, технології, які використовуються при розробці програм незалежно від мови програмування. Тому, для вас прийшов час оволодіти методологією, яка є фундаментом сучасної технології розробки програмного забезпечення: *об'єктно-орієнтованим програмуванням*.

Успіхів!

Левченко Віктор Володимирович

Лабораторна робота № 1

ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ КЛАСУ

Мета роботи:

- розвинути об’єктно-орієнтоване мислення в процесі аналізу та розв’язання поставленої задачі;
- вивчити протокол оголошення та реалізації класу в мові програмування С++;
- закріпити на практиці базові поняття і принципи об’єктно-орієнтованого програмування.

Теоретичні відомості

1. Основні поняття об’єктно-орієнтованого програмування

Об’єктно-орієнтоване програмування (ООП) – це метод програмування, який базується на представленні програми у вигляді сукупності об’єктів, що взаємодіють, кожен з яких є екземпляром певного класу, а класи є членами певної ієрархії успадкування.

Клас – це множина об’єктів, які мають спільну структуру, поведінку і семантику.

З точки зору програмування клас є складним типом, який визначає програміст. Кожен клас містить *дані* (*властивості*) та *функції* (*методи*), які маніпулюють цими даними.

Окремий *об’єкт* є *екземпляром класу*, тобто *змінною класового типу*.

Інкапсуляція – це властивість об’єкту, яка дозволяє йому об’єднувати властивості і методи, приховуючи таким чином реалізацію об’єкту від користувача.

2. Оголошення і реалізація класів у С++

Оголошення одиночного класу в мові програмування С++ здійснюється за наступним форматом:

```
class <назва класу>
{
private:
    <перелік приватних елементів класу>
public:
    <перелік відкритих (загальнодоступних) елементів класу>
};
```

Мітки “private” і “public” називаються *специфікаторами доступу до членів класу*.

Мітка “private” відкриває розділ класу, в якому розміщуються оголошення його закритих (приватних) елементів. Закриті елементи класу доступні лише для методів класу.

Мітка “public” відкриває розділ класу, в якому розміщуються оголошення його загальнодоступних елементів.

За замовчуванням (без явної вказівки мітки) всі елементи класу є приватними, тому мітку “private” завчасто не використовують і починають опис структури класу з переліку приватних елементів.

Приклад:

```
class Sphere
{
    double Radius;
public:
    void SetRadius (double);
    double GetVolume ();
};
```

Реалізація методів класу здійснюється двома способами. Перший спосіб полягає в тому, що тіло метода розміщується відразу після його оголошення (прототипу) в тілі класу. Такий метод називається *inline-методом*. Згідно другого способу (використовується переважно) реалізація метода здійснюється поза межами класу. При цьому для вказівки приналежності метода до певного

класу використовується *оператор дозволу області видимості* – "::".

Приклад:

```
class Sphere
{
    double Radius;
public:
    // inline-метод
    void SetRadius (double NewRadius)
    {
         if (Radius > 0.0) Radius = NewRadius;
         else Radius = 0.0;
    }
    double GetVolume ();
};

// реалізація метода GetVolume
double Sphere::GetVolume ()
{
    return 4.0 * M_PI * Radius * Radius * Radius / 3.0;
}
```

3. Оголошення і використання об'єктів

Як і змінна будь-якого типу, об'єкт класу може оголошуватись (створюватись) глобально і локально, статично і динамічно.

Доступ до відкритих елементів об'єкта, оголошеного статично, здійснюється за допомогою оператора ".". Якщо ж використовується покажчик на об'єкт класу, то доступ здійснюється за допомогою оператора "->".

Приклад:

```
...
// глобальне статичне оголошення об'єкта
Sphere A;
// глобальне оголошення покажчика
Sphere *B;
void main ()
{
```

```
    // локальне статичне оголошення об'єкта
    Sphere C;
    // динамічне створення об'єкта
    B = new Sphere;

    // виклик метода для об'єкта, створеного статично
    A.SetRadius (1.0);
    C.SetRadius (25.0);
    // виклик метода для об'єкта, створеного динамічно
    // (через покажчик)
    B->SetRadius (5.0);

    printf ("VA = %lf\n", A.GetVolume ());
    printf ("VB = %lf\n", B->GetVolume ());
    printf ("VC = %lf\n", C.GetVolume ());

    // знищення об'єкта створеного динамічно
    delete B;

    getch ();
}
```

Методичні вказівки

Завданням даної лабораторної роботи є створення, компіляція, відладка та виконання програми, в якій реалізується і тестується певний клас. *Результатом* виконання лабораторної роботи має бути демонстрація коректної логіки функціонування методів реалізованого класу в програмі.

В процесі виконання роботи рекомендується дотримуватись наступної послідовності дій:

- сформувати орієнтовний список властивостей і методів класу, необхідних для моделювання його роботи згідно завдання;
- визначити доступність кожної властивості класу для читання і запису; в разі необхідності доповнити перелік методів класу методами читання і запису властивостей;
- визначити, як виконання кожного метода впливає на стан об'єкта класу, тобто на його властивості;

- визначити, які методи заданого класу є приватними, а які – відкритими;
- реалізувати клас в програмі;
- визначити взаємодію методів класу між собою;
- реалізувати методи класу в програмі;
- реалізувати програму, яка дозволяє ініціювати виконання відкритих методів об’єкта класу.

При виконанні лабораторної роботи слід дотримуватись наступних рекомендацій:

- назви класу, об’єктів та їх складових елементів повинні відображати відповідний зміст або функціональне призначення; зокрема бажано, щоб назви методів, призначених для запису (ініціалізації) і читання (отримання доступу) даних класу, починались з префіксів Set і Get відповідно;
- програма, яка виконує тестування реалізованого класу, повинна забезпечувати можливість ручного (з клавіатури) запуску будь-якого відкритого метода класу та виводу поточного стану об’єкта (всіх його властивостей) на дисплей;
- взаємодія програми з користувачем (ввід з клавіатури, вивід на дисплей) повинна здійснюватись незалежно від методів класу. Іншими словами, не рекомендується здійснювати виклик функцій вводу/виводу методами класу: це призведе до втрати останніми таких властивостей, як можливість повторного використання і переносимість;
- рекомендується розміщувати реалізований код у файлах за наступною схемою: оголошення класу (прототип) – у заголовочному файлі (*.h або *.hpp); реалізація класу – у файлі *.cpp (з такою ж назвою), до якого відповідний заголовочний файл підключається за допомогою директиви #include; основна програма – у файлі *.cpp, до якого також підключається

заголовочний файл класу. Всі файли при цьому слід розміщувати в одній директорії.

Приклад

Реалізувати та протестувати клас, який моделює струменевий принтер, з наступними вихідними даними:

- *живлення принтера може бути увімкнено та вимкнено;*
- *принтер може перебувати в одному з наступних станів: "готовий", "відсутні чорнила", "відсутній папір", "аварія" (наприклад, якщо застряг папір); існує фіксована ймовірність виникнення аварії – 0,05 (5 %); вихід з усіх неробочих станів можливий лише за рахунок тимчасового вимикання принтера;*
- *принтер характеризується, також, кількістю паперу, завантаженого в його лоток (0 – 100 аркушів), та кількістю наявних чорнил у картриджі (0 – 50 мл);*
- *над принтером можна виконувати наступні операції: вмикати/вимикати, завантажувати та вивантажувати папір, заправляти чорнилами (лише у вимкненому стані), запускати друк заданої кількості сторінок (друк завершується автоматично);*
- *контроль кількості аркушів та чорнил здійснюється лише в процесі друку; для здійснення контролю використовується коефіцієнт середнього заповнення аркуша, який змінюється від нуля до ста процентів: за цим коефіцієнтом визначається кількість чорнил (мл), необхідна для друку одного аркуша:*

$$p = 0{,}02k,$$

де k *– коефіцієнт середнього заповнення аркуша, %;*

p *– кількість чорнил, необхідна для друку одного аркуша, мл.*

Розв'язання:

Проведемо аналіз заданого класу – струменевого принтера. Визначимо

та проіменуємо *властивості* принтера, а також виберемо типи даних C++, які найбільш доцільно використовувати для програмного представлення кожної властивості:

Зміст властивості	Пропонована назва	Тип даних C++
Живлення	Power	bool
Стан	State	enum PrinterState {OFF, READY, NOINK, NOPAPER, ERROR};
Зміст властивості	Пропонована назва	Тип даних C++
Кількість аркушів паперу в лотку	Paper	int
Кількість чорнил в картриджі	Ink	int

Для представлення стану принтера пропонується використовувати спеціальне перелічення з назвою PrinterState.

Відповідно до сучасних вимог професійного програмування всі перелічені властивості планується оголосити приватними, а для доступу до них реалізувати відповідні методи.

Визначимо та створимо прототипи *методів*, які складатимуть *інтерфейс* принтера, тобто ту його частину, яка є доступною для користувача:

Зміст метода	Прототип	Зміст аргументів
Увімкнення	void TurnOn (void);	-
Вимкнення	void TurnOff (void);	-
Визначення стану живлення	bool GetPower (void);	-
Завантаження паперу	int LoadPaper (int);	Кількість аркушів
Вивантаження паперу	int UnloadPaper (int);	Кількість аркушів
Визначення кількості аркушів паперу	int GetPaper (void);	-
Заправка чорнилами	int LoadInk (int);	Кількість чорнил
Визначення кількості чорнил	int GetInk (void);	-

Запуск друку	int Print (int);	Кількість аркушів
Зміст метода	Прототип	Зміст аргументів
Визначення стану принтера	int GetState (void);	-

Методи GetPower, GetPaper і GetInk повертають значення відповідно властивостей Power, Paper та Ink. Методи LoadPaper і UnloadPaper повертають кількість аркушів паперу, яку вдалось завантажити або вивантажити відповідно. Аналогічно, метод LoadInk повертає кількість чорнил, яку вдалось використати для заправлення картриджу. Метод Print повертає кількість сторінок, яку вдалось роздрукувати з числа заданих з урахуванням доступної кількості аркушів, чорнил та можливої аварійної ситуації. Метод GetState повертає поточний стан принтера.

Перелічені методи є відкритими.

Стан принтера можуть змінювати методи TurnOn та Print.

Лістинг програми мовою C++ (IDE BC, BDS/ECRS):

```
// вміст файлу printer.hpp
// ---------------------------------------------------------------------------
// константи, які задають найбільшу кількість паперу і чорнил
const int MAXPAPER = 100;
const int MAXINK = 50;
// ймовірність виникнення помилки друку (5 %)
const float ERRORPROB = 0.05;
// перелічення можливих станів принтера
enum PrinterState {OFF, READY, NOINK, NOPAPER, ERROR};
// масив констант з назвами станів
const char PrinterStateName[5][10] = {"OFF", "READY", "NOINK", "NOPAPER", "ERROR"};
// оголошення класу принтера
class Printer
{
      bool Power;
      int State;
      int Paper;
      float Ink;
      // приватний метод, що встановлює заданий стан
      // метод призначено для внутрішнього використання іншими методами
      // (користувач не може самовільно змінити стан принтера:
      // для цього необхідно використовувати доступні методи)
      void SetState (int);
public:
      // ініціалізація змінних класу початковими значеннями
      void Reset (void);
      // управління живленням
      void TurnOn (void);
      void TurnOff (void);
```

```
        // inline-реалізація простого метода
        bool GetPower (void) {return Power;}
        // управління папером
        int LoadPaper (int);
        int UnloadPaper (int);
        int GetPaper (void) {return Paper;}
        // управління чорнилами
        int LoadInk (int);
        int GetInk (void) {return Ink;}
        // друк
        int Print (int, int);
        // визначення коду і назви стану
        int GetState (void);
        const char* const GetStateName () {return PrinterStateName[State];}
};
// вміст файлу printer.cpp
// ----------------------------------------------------------------------------
#include <stdlib.h>
#include "printer.hpp"
// реалізація класу принтера
void Printer::Reset (void)
{
        Power = false;
        SetState (OFF);
        Paper = 0;
        Ink = 50;
}
void Printer::TurnOn (void)
{
        if (!Power)
        {
                Power = true;
                SetState (READY);
        }
        // для імітації подальшого випадкового виникнення помилок при друку
        randomize ();
}
void Printer::TurnOff (void)
{
        if (Power)
        {
                Power = false;
                SetState (OFF);
        }
}
int Printer::LoadPaper (int Sheets)
{
        if (Sheets < 0) return 0;
        int NewPaper = Paper + Sheets;
        // перевірка, чи не перевищує кількість паперу найбільшу можливу
        if (NewPaper > MAXPAPER)
        {
                Paper = MAXPAPER;
                return MAXPAPER - Sheets;
        }
        Paper = NewPaper;
        return Sheets;
}
int Printer::UnloadPaper (int Sheets)
{
        if (Sheets < 0) return 0;
        int OldPaper = Paper;
```

```
	int NewPaper = Paper - Sheets;
	// перевірка, чи не перевищує кількість аркушів наявну
	if (NewPaper < 0)
	{
		Paper = 0;
		return OldPaper;
	}
	Paper = NewPaper;
	return Sheets;
}
int Printer::LoadInk (int Drops)
{
	if (Power) return 0;
	if (Drops < 0) return 0;
	int NewInk = Ink + Drops;
	// перевірка, чи не перевищує кількість чорнил найбільшу можливу
	if (NewInk > MAXINK)
	{
		Ink = MAXINK;
		return MAXINK - Drops;
	}
	Ink = NewInk;
	return Drops;
}
void Printer::SetState (int NewState)
{
	State = NewState;
}
int Printer::Print (int Sheets, int Fill)
{
	// перевірка працездатності принтера та коректності заданих параметрів
	if (!Power || State != READY) return 0;
	if (Sheets < 0 || Fill < 0 || Fill > 100) return 0;
	int LuckSheets;
	// імітація виникнення випадкової помилки із заданою ймовірністю
	if (random (100) < ERRORPROB * 100) LuckSheets = random (Sheets);
	else LuckSheets = Sheets;
	// обчислення кількості сторінок, на друк яких вистачить наявних чорнил
	int PosSheetsByInk = Ink * 50 / Fill, PosPaper;
	// не вистачить чорнил
	if (PosSheetsByInk < LuckSheets)
	{
		PosPaper = PosSheetsByInk;
		SetState (NOINK);
	}
	// не вистачить паперу
	else if (Paper < LuckSheets)
	{
		PosPaper = Paper;
		SetState (NOPAPER);
	}
	// вистачить всього
	else
	{
		PosPaper = LuckSheets;
		SetState (READY);
	}
	// зменшення кількості чорнил і паперу:
	// обнулення "Ink = 0" потрібне для усунення похибки, яка
	// може виникати при обчисленні потрібної кількості чорнил
	if (State == NOINK) Ink = 0;
	else Ink -= PosPaper * Fill / 50;
```

```
	UnloadPaper (PosPaper);
	if (LuckSheets < Sheets) SetState (ERROR);
	return PosPaper;
}
int Printer::GetState (void)
{
	return State;
}
// вміст файлу testprn.cpp
// ---------------------------------------------------------------------------------------
#include <conio.h>
#include <stdio.h>
#include "printer.hpp"
// основна програма
void main ()
{
	// оголошення об'єкта класу Printer
	Printer Canon;
	Canon.Reset ();
	// бескінечний цикл обробки команд з клавіатури
	do
	{
		// вивід запрошення
		clrscr ();
		printf ("--------Object-oriented printer model--------\n");
		printf ("----------------------------------------------\n");
		printf ("----Select one of these options [e - exit]---\n");
		printf ("[0 - Turn off] [1 - Turn on] [2 - Load paper]\n");
		printf ("[3 - Unload paper] [4 - Load ink] [5 - Print]\n");
		printf ("----------------------------------------------\n");
		// вивід стану принтера
		gotoxy (1, 8);
		printf ("State of printer:\n");
		printf ("Power: %d State: %s ", Canon.GetPower (), Canon.GetStateName());
		printf ("Paper: %d Ink: %d", Canon.GetPaper(), Canon.GetInk());
		// прийом команди з клавіатури
		int key = getch ();
		gotoxy (1, 7);
		switch (key)
		{
			case '0': Canon.TurnOff (); break;
			case '1': Canon.TurnOn (); break;
			case '2':
			{
				int Sheets;
				printf ("Input the number of sheets to load: ");
				scanf ("%d", &Sheets);
				Canon.LoadPaper (Sheets);
				break;
			}
			case '3':
			{
				int Sheets;
				printf ("Input the number of sheets to unload: ");
				scanf ("%d", &Sheets);
				Canon.UnloadPaper (Sheets);
				break;
			}
			case '4':
			{
				int Drops;
				printf ("Input the volume of ink to load: ");
```

```
                        scanf ("%d", &Drops);
                        Canon.LoadInk (Drops);
                        break;
                }
                case '5':
                {
                        int Sheets, Fill;
                        printf ("Input the number of sheets and fill: ");
                        scanf ("%d%d", &Sheets, &Fill);
                        Canon.Print (Sheets, Fill);
                        break;
                }
                case 'e':
                {
                        return;
                }
                default:
                        printf ("Invalid option! Press any key to continue...");
                        getch ();
        }
    } while (true);
}
```

Зверніть увагу на те, що для зручності роботи і наочності в клас було додано закритий (приватний) метод SetState для зміни поточного стану принтера, а також відкритий метод GetStateName, який повертає *константний покажчик на константу* – рядок з назвою відповідного стану з глобального масиву PrinterStateName.

Завдання

1) вивчити теоретичні відомості та методичні вказівки до лабораторної роботи;
2) вибрати завдання для виконання згідно з варіантом;
3) реалізувати клас та тестуючу програму;
4) відкомпілювати та відладити програму;
5) відповісти на контрольні запитання;
6) зробити висновки.

Загальне завдання: реалізувати та протестувати клас, який моделює заданий об’єкт реального світу згідно з варіантом.

Варіанти завдань

<table>
<tr><th>Варіант</th><th colspan="2">Клас, його властивості та логіка роботи методів</th></tr>
<tr><td rowspan="8">1</td><td colspan="2">Радіоприймач з електронним регулюванням</td></tr>
<tr><td colspan="2">Властивості:
• стан (увімкн./вимкн.);
• частота налаштування, МГц (88 … 108);
• назва активної станції;
• рівень гучності (0 … 10).</td></tr>
<tr><td>Метод</td><td>Правила роботи метода</td></tr>
<tr><td>Вмикання</td><td>Встановлює середню гучність і найменшу частоту налаштування в діапазоні.</td></tr>
<tr><td>Вимикання</td><td>Скидає гучність до нуля.</td></tr>
<tr><td>Перемикання на станцію із заданою назвою</td><td>Встановлює відповідну частоту налаштування.</td></tr>
<tr><td>Запуск збільшення/зменшення частоти налаштування</td><td>Скидає гучність до нуля; відповідно збільшує або зменшує частоту налаштування з кроком 0,5 МГц.</td></tr>
<tr><td>Завершення збільшення/зменшення частоти налаштування</td><td>При налаштуванні на станцію висвітлює її назву; відновлює попередній рівень гучності.</td></tr>
<tr><td>Збільшення/зменшення гучності</td><td>Відповідно збільшує або зменшує рівень гучності з кроком 1.</td></tr>
<tr><td rowspan="7">2</td><td colspan="2">Холодильник</td></tr>
<tr><td colspan="2">Властивості:
• стан живлення (увімкн./вимкн.);
• поточна температура (–25 ... 0) °C;
• стан дверці (відчинена/зачинена);
• сигнал аварійного стану (увімкн./вимкн.).</td></tr>
<tr><td>Метод</td><td>Правила роботи метода</td></tr>
<tr><td>Вмикання</td><td>Якщо дверця зачинена знижує поточну температуру до рівня заданої за час, який розраховується згідно виразу:
$t\ [хв] = 24 \cdot \ln (t_{нс} - t_з)$,
де $t_{нс}$ – температура навколишнього середовища, °C;
$t_з$ – задана (цільова) температура, °C.
Якщо дверця відчинена, прирівнює поточну температуру до температури навколишнього середовища і вмикає сигнал аварійного стану.</td></tr>
<tr><td>Вимикання</td><td>Вимикає сигнал аварійного стану (якщо він працював); підвищує поточну температуру до температури навколишнього середовища.</td></tr>
<tr><td>Встановлення температури</td><td>Якщо дверця зачинена знижує поточну температуру до рівня заданої за час, який розраховується згідно виразу:
$t\ [хв] = 24 \cdot \ln (t_п - t_з)$,
де $t_{нс}$ – поточна температура, °C;
$t_з$ – задана (цільова) температура, °C.
Якщо дверця відчинена прирівнює поточну температуру до температури навколишнього середовища і вмикає сигнал аварійного стану.</td></tr>
<tr><td>Відчинення дверці</td><td>Якщо поточна температура менша за температуру навколишнього середовища підвищує поточну температуру на 5 % від їх різниці.</td></tr>
<tr><td>Зачинення дверці</td><td>Вимикає сигнал аварійного стану (якщо він працював); знижує поточну температуру до рівня заданої за час, який розраховується аналогічно до метода “Вмикання”.</td></tr>
<tr><td rowspan="5">3</td><td colspan="2">Стаціонарний телефон</td></tr>
<tr><td colspan="2">Властивості:
• стан лінії (сигнал присутній/відсутній);
• стан трубки (на базі/піднята);
• рівень гучності (0 … 3);
• наявність гудка (відсутній/безперервний/очікування/зайнято).</td></tr>
<tr><td>Метод</td><td>Правила роботи метода</td></tr>
<tr><td>Підключення до лінії</td><td>Встановлює відповідний стан лінії; вмикає неперервний гудок, якщо трубка піднята.</td></tr>
<tr><td>Відключення від лінії</td><td>Встановлює відповідний стан лінії; вимикає гудок.</td></tr>
</table>

	Підняття трубки	Встановлює відповідний стан трубки; вмикає гудок, якщо лінія активна (є сигнал).
	Опускання трубки	Встановлює відповідний стан трубки; вимикає гудок.
	Встановлення рівня гучності	Встановлює заданий рівень гучності.
	Набір номеру	Якщо є сигнал лінії і трубка піднята встановлює сигнал “очікування” або “зайнято” з ймовірністю ½.
4	**Автоматична пральна машина**	
	Властивості: • стан (увімкн./вимкн.); • режим (прання/швидке прання/віджим); • кількість обертів двигуна (400/600/1000 відповідно до режиму); • рівень води у баку (0 … 40 л); • стан двигуна (увімкн./вимкн.); • стан помпи набору/зливу води (вимкн./набір/злив).	
	Метод	**Правила роботи метода**
	Вмикання	Встановлює режим “прання”.
	Вимикання	Вимикає двигун і помпу зливу.
	Вибір режиму	Встановлює заданий режим і відповідну кількість обертів двигуна.
	Запуск встановлення рівня води у баку	Вмикає помпу у режим наповнення або зливу.
	Завершення встановлення рівня води у баку	Встановлює заданий рівень води у баку; вимикає помпу.
	Запуск роботи	Якщо рівень води в баку в межах норми і вибрано режим прання вмикає лише двигун, інакше додатково вмикає помпу у відповідний режим. Якщо вибрано віджим, вмикає лише помпу в режимі зливу.
	Зупинка	Якщо вибрано режим прання, встановлює рівень води у баку 75 % від найбільшого, інакше – 0 %; вимикає двигун і/або помпу.
5	**Ліфт**	
	Властивості: • загальний стан (простій/виклик/перевезення); • стан двигуна (вимкн./підйом/спуск); • стан дверчат (відчинені/зачинені); • поточний номер поверха (1 … 24); • навантаження на підлогу (0 … 250 кг).	
	Метод	**Правила роботи метода**
	Старт виклику	Зачиняє дверчата; за заданим цільовим номером поверху встановлює відповідний режим двигуна.
	Кінець виклику	Зупиняє двигун; встановлює поточний номер поверху; відчиняє дверчата.
	Завантаження кабіни	Встановлює задане навантаження на підлогу.
	Запуск	Якщо граничне навантаження в межах норми (але не відсутнє), зачиняє дверчата; за заданим цільовим номером поверху встановлює відповідний режим двигуна.
	Зупин	Зупиняє двигун; встановлює поточний номер поверху; відчиняє дверчата.
6	**Мікрохвильова піч з грилєм**	
	Властивості: • стан (увімкн./вимкн.); • режим (мікрохвилі/гриль); • потужність мікрохвиль (160/320/640/800 Вт); потужність грилю вважається фіксованою – 1000 Вт; • час за таймером (10 с – 60 хв); • стан випромінювача (увімкн./вимкн.); • стан нагрівача (увімкн./вимкн.).	
	Метод	**Правила роботи метода**
	Вмикання живлення	Встановлює режим мікрохвиль з потужністю 800 Вт і скидає таймер.
	Вимикання живлення	Вимикає випромінювач і нагрівач.
	Вибір режиму роботи	Встановлює вибраний режим (якщо не працює випромінювач і нагрівач).
	Встановлення потужності мікрохвиль	Встановлює потужність із стандартного ряду (якщо не працює випромінювач і нагрівач).
	Встановлення часу за таймером	Встановлює заданий час таймера.
	Запуск розігріву	Вмикає відповідний пристрій (випромінювач або нагрівач) якщо заданий час таймера.
	Відкриття дверці	Вимикає випромінювач або нагрівач.

	Закриття дверці	Якщо випромінювач або нагрівач раніше було увімкнено, вимкає відповідний пристрій.
	Завершення розігріву	Вимикає відповідний пристрій. Повертає кількість спожитої електроенергії у кВт·год.
7	**Пилосос з регулятором потужності**	
	Властивості: • стан електромережі (увімкн./вимкн.); • стан двигуна (увімкн./вимкн.); • стан індикатора переповнення контейнера пилу (увімкн./вимкн.); • потужність (1000 … 1500 Вт); • об'єм пилу в контейнері (0 … 2 л).	
	Метод	**Правила роботи метода**
	Підключення до електромережі	Встановлює відповідний стан.
	Відключення від електромережі	Вимикає двигун.
	Вмикання	Вмикає двигун; якщо контейнер пилу заповнений більше ніж на 90 % вмикається індикатор переповнення; якщо рівень пилу досягає або перевищує 100 % двигун не вмикається.
	Вимикання	Вимикає двигун; збільшує об'єм пилу в контейнері на величину, яка обчислюється за виразом: $V = S \cdot k$, де V – об'єм пилу, л; S – площа обробленої поверхні, м^2; k – коефіцієнт запиленості поверхні, л/м^2.
	Випорожнення контейнера пилу	Скидає рівень заповненості контейнера пилу до нуля.
8	**Електрочайник з системою підтримки температури**	
	Властивості: • стан електромережі (увімкн./вимкн.); • стан нагрівача (увімкн./вимкн.); • режим (кип'ятильник/термостат); • об'єм води (0 … 2 л); • початкова температура води (0… 30 °C); • цільова температура (50 … 100 °C).	
	Метод	**Правила роботи метода**
	Підключення до електромережі	Встановлює відповідний стан.
	Відключення від електромережі	Вимикає нагрівач.
	Встановлення режиму	Встановлює заданий режим; якщо вибрано режим термостату і цільова температура більша за початкову – вмикається нагрівач.
	Набір/злив води	Встановлює заданий об'єм води.
	Встановлення цільової температури	Встановлює задану цільову температуру.
	Вмикання	Якщо вибрано режим термостату, нагрівач вмикається тоді, коли цільова температура більша за початкову; в режимі кип'ятильника нагрівач вмикається завжди.
	Вимикання	Вимикає нагрівач; повертає кількість енергії, витраченої на нагрів: $Q = c \cdot m \cdot \Delta t$, де Q – кількість енергії, Дж; c – питома теплоємність води, c = 4200 Дж / (кг · °C); m – маса води, кг; Δt – різниця температур, °C.
9	**DVD-плейєр для домашнього кінотеатру**	
	Властивості: • стан (увімкн./вимкн.); • наявність диску; • активний трек (1 … X), де X – кількість треків на диску; • стан програвача (немає диску/простій/програвання/кінець диску).	
	Метод	**Правила роботи метода**
	Вмикання/вимикання	Встановлює відповідний стан.
	Встановлення/витягнення диску	Встановлює відповідний стан та кількість треків.
	Старт програвання	Запускає програвач.
	Завершення програвання	Збільшує номер треку на одиницю; зупиняє програвач
	Зміна треку	Зупиняє програвач і встановлює заданий номер треку

<table>
<tr><td rowspan="7">10</td><td colspan="2">Банкомат</td></tr>
<tr><td colspan="2">Властивості:
• стан (працює/не працює/помилка);
• обсяг грошей (0 … 100000 грн); сума повинна бути кратна 5 гривням;
• довжина касової стрічки (0 … 30 м); один чек має довжину 10 см;
• індикатор відсутності касової стрічки;
• поточний рівень прав користувача (гість/адміністратор).</td></tr>
<tr><td>Метод</td><td>Правила роботи метода</td></tr>
<tr><td>Авторизація</td><td>Встановлює заданий рівень прав; для адміністратора виконується перевірка на фіксований пароль.</td></tr>
<tr><td>Вихід з системи</td><td>Встановлює рівень прав гостя; ліквідує помилковий стан.</td></tr>
<tr><td>Зміна обсягу наявних грошей</td><td>Встановлює заданий обсяг грошей (доступний лише для адміністратора).</td></tr>
<tr><td>Зміна довжини касової стрічки</td><td>Встановлює задану довжину касової стрічки (доступний лише для адміністратора).</td></tr>
<tr><td>Зняття грошей</td><td>Зменшує наявний обсяг грошей на задану суму та касову стрічку на довжину одного чеку; якщо доступна сума менше 50 грн. ініціює помилковий стан; якщо касова стрічка закінчилась, вмикає відповідний індикатор.</td></tr>
<tr><td rowspan="7">11</td><td colspan="2">Електронний годинник</td></tr>
<tr><td colspan="2">Властивості:
• живлення (увімкн./вимкн.);
• режим (12/24 год);
• індикатор AM/PM;
• кількість годин і хвилин поточного часу;
• стан будильника (встановлений/не встановлений);
• кількість годин і хвилин будильника;
• сигнал будильника.</td></tr>
<tr><td>Метод</td><td>Правила роботи метода</td></tr>
<tr><td>Вмикання</td><td>Скидає всі параметри на нуль.</td></tr>
<tr><td>Вимикання</td><td>Встановлює відповідний стан.</td></tr>
<tr><td>Встановлення годин/хвилин поточного часу</td><td>Встановлює відповідні параметри.</td></tr>
<tr><td>Встановлення годин/хвилин будильника</td><td>Встановлює відповідні параметри.</td></tr>
<tr><td>Перевірка будильника</td><td>Якщо поточний час і час будильника співпадають, вмикає сигнал будильника.</td></tr>
<tr><td>Встановлення режиму</td><td>Перелічує час; вмикає або вимикає індикатор AM/PM.</td></tr>
<tr><td rowspan="6">12</td><td colspan="2">Настольна лампа з автоматичним регулятором освітленості</td></tr>
<tr><td colspan="2">Властивості:
• стан електромережі (увімкн./вимкн.);
• режим регулювання (ручний/автоматичний);
• положення регулятора (0 … 100 %);
• температура плафона (не більше 100 °C);
• сигнал перегрівання (увімкн./вимкн.).</td></tr>
<tr><td>Метод</td><td>Правила роботи метода</td></tr>
<tr><td>Вмикання на заданий час</td><td>Встановлює ручний режим регулювання і відповідний стан; визначає температуру плафона залежно від часу роботи лампи за виразом:
$$t = t_{нс}\left(1+\frac{P}{100}\right)+\frac{P}{3}th\left(\frac{T-30}{30}\right),$$
де $t_{нс}$ – температура навколишнього середовища, °C;
P – положення регуклятора, %;
T – час роботи лампи, хв.
Якщо температура перевищує допустиму, вмикає сигнал перегрівання.</td></tr>
<tr><td>Вимикання</td><td>Встановлює відповідний стан; скидає температуру плафона до 25 °C; вимикає сигнал перегрівання.</td></tr>
<tr><td>Зміна положення регулятора на заданий час</td><td>У включеному стані переобчислює температуру плафона (з урахуванням поточної температури і нового заданого часу роботи лампи); враховує можливість перегрівання.</td></tr>
<tr><td>Встановлення автоматичного режиму регулювання на заданий час</td><td>У включеному стані встановлює положення регулятора залежно від рівня освітленості згідно закону:</td></tr>
</table>

		$$P = \frac{1}{\frac{L}{5000} + 0,01},$$ де L – рівень освітленості, Лк (0 … 50000); переобчислює температуру плафона (з урахуванням поточної температури і нового заданого часу роботи лампи); враховує можливість перегрівання.
13	**Праска**	
	Властивості: • стан електромережі (увімкн./вимкн.); • режим роботи (I, II, III з відповідними найбільшими температурами поверхні 100, 150 і 200 °C); • наповнення паровика (0 … 200 г); • температура поверхні.	
	Метод	**Правила роботи метода**
	Вмикання	Встановлює відповідний стан; обчислює час нагріву до найбільшої температури поточного вибраного режиму за виразом: $T = \lvert t_{кінц} - t_{поч} \rvert / 2$, де $t_{поч}$ – початкова температура поверхні, °C; $t_{кінц}$ – кінцева температура поверхні, °C; T – час нагріву, с.
	Вимикання	Знижує температуру поверхні до рівня навколишньої.
	Набір води в паровик	Встановлює заданий рівень води в паровику (можливо лише у виключеному стані).
	Перемикання режиму роботи	Встановлює вибраний режим; обчислює час досягнення відповідної температури (у включеному стані).
	Паровий удар	Зменшує кількість води у паровику на величину, яка визначається відповідно до поточного режиму наступним чином: I – 5 г, II – 7,5 г, III – 10 г (можливо лише у включеному стані).
14	**Кондиціонер**	
	Властивості: • стан електромережі (увімкн./вимкн.); • цільова температура (16 … 25 °C); • стан вентилятора (увімкн./вимкн.); • стан охолоджувача (увімкн./вимкн.); • режим роботи (охолодження, осушення); • індикатор забруднення фільтру (при напрацюванні більше 1000 год).	
	Метод	**Правила роботи метода**
	Вмикання	Встановлює відповідний стан та режим охолодження з цільовою температурою 20 °C; вмикає охолоджувач; обчислює час, необхідний для досягнення заданої температури за виразом: $T = 10 \ln \lvert t_1 - t_2 \rvert$, де t_1, t_2 – кінцева і початкова температури відповідно, °C; T – час охолодження, хв.; якщо цільова температура менша заданої більш ніж у два рази, вмикає вентилятор; якщо загальний час роботи перевищує допустимий, вмикає індикатор забруднення фільтру.
	Вимикання	Встановлює відповідний стан; вимикає охолоджувач і вентилятор.
	Вибір цільової температури	Обчислює час, необхідний для досягнення заданої температури.
	Вибір режиму роботи	В режимі охолодження виконує дії, аналогічні вмиканню; в режимі осушення вимикає охолоджувач і вмикає вентилятор.
	Заміна фільтру	Скидає лічильник забруднення (можливо лише у виключеному стані).
15	**Дитячий калькулятор**	
	Властивості: • стан (увімкн./вимкн.); • регістри операндів X та Y; • регістр операції.	
	Метод	**Правила роботи метода**
	Вмикання	Встановлює відповідний стан; обнуляє всі регістри.
	Вимикання	Встановлює відповідний стан
	Натиснення клавіши з цифрою	Залежно від стану вводу (перший або другий операнд) формує числовий вміст регістра X або Y.

	Натиснення клавіши з операцією	Формує вміст регістра операції
	Натиснення клавіши “дорівнює”	Виконує операцію згідно вмісту регістра операції; затирає регістр X результатом обчислень.

Контрольні запитання

1) дайте визначення об’єктно-орієнтованого програмування (ООП), сформулюйте мету його впровадження;
2) поясніть принцип абстракції в ООП;
3) дайте визначення класу, об’єкта, властивості, метода, інкапсуляції;
4) перелічіть специфікатори прав доступу для елементів класу і поясніть їх призначення;
5) сформулюйте правила ініціалізації масивів.

Лабораторна робота № 2

СТВОРЕННЯ КОНСТРУКТОРІВ І ДЕСТРУКТОРА

Мета роботи:

- вивчити протокол оголошення та опису конструкторів і деструктора в мові С++;
- закріпити на практиці знання властивостей конструкторів і деструктора;
- набути вміння використовувати конструктори і деструктор при розв’язанні задач.

Теоретичні відомості

1. Вступ (постановка проблеми)

При створенні об’єкта класу його властивості ініціалізуються компялітором автоматично, а, отже, можуть набувати недопустимих (некоректних з точки зору змісту об’єкта) значень. Використання власних функцій ініціалізації не вирішує цю проблему у повній мірі, тому що програміст може просто забути викликати необхідну функцію в програмі.

2. Конструктори

Конструктор – це спеціальний метод класу, який викликається автоматично при створенні об’єкта класу. Головною відзнакою конструктора від інших методів класу полягає в тому, що його назва співпадає з назвою класу.

Конструктори призначені для виконання певних ініціалізуючих операцій в момент створення об’єкта класу: привласнення початкових значень змінним, динамічного виділення пам’яті, відкриття файлу, встановлення з’єднання з базою даних або мережевого з’єднання тощо.

Конструктор не може повертати значення, тому при його оголошенні та

реалізації **не** вказується навіть кваліфікатор “пустого” типу – **void.**

Розрізняють три типи конструкторів:

- конструктор за замовчуванням;
- конструктор ініціалізації;
- конструктор копіювання.

Конструктором за замовчуванням називається конструктор, який не має аргументів або всі аргументи якого є аргументами за замовчуванням. В класі може бути оголошений лише один конструктор за замовчуванням.

Приклад:

```
class Sphere
{
    double Radius;
public:
    // оголошення конструктора за замовчуванням
    // інший варіант: Sphere (double = 0.0);
    Sphere ();
    void SetRadius (double);
    double GetVolume ();
};

// реалізація конструктора:
// створюється сфера з одиничним радіусом
Sphere::Sphere ()
{
    Radius = 1.0;
}

// неявне використання конструктора в тілі програми
...
Sphere A; // властивість Radius об’єкта A дорівнює одиниці
```

Якщо в класі немає жодного конструктора, оголошеного програмістом, конструктор за замовчуванням створюється неявно автоматично.

Конструктором ініціалізації називають будь-який конструктор, який приймає довільні аргументи (за виключенням конструктора копіювання, який приймає лише один аргумент – *посилання на об’єкт відповідного класу*). Клас

може мати довільну кількість конструкторів ініціалізації, які перевантажуються аналогічно до звичайних функцій (методів).

Приклад:

```
class Array
{
    int Count;
    float *Data;
public:
    // оголошення конструктора за замовчуванням
    Array ();
    // оголошення конструктора ініціалізації 1
    Array (int);
    // оголошення конструктора ініціалізації 2
    Array (int, float);
    // оголошення конструктора ініціалізації 3
    Array (int, float*);
    ...
};


// реалізація конструктора за замовчуванням
Array::Array ()
{
    Count = 0;
    Data = NULL;
}
// реалізація конструктора ініціалізації 1:
// створюється масив із заданою кількістю елементів
Array::Array (int Count)
{
    this->Count = Count;
    Data = new float[Count];
}
// реалізація конструктора ініціалізації 2:
// створюється масив із заданою кількістю елементів;
// всі елементи ініціалізуються заданим значенням
Array::Array (int Count, float Value)
{
    this->Count = Count;
    Data = new float[Count];
    for (int i = 0; i < Count; i++) Data[i] = Value;
}
```

```
// реалізація конструктора ініціалізації 3:
// створюється масив із заданою кількістю елементів;
// елементи копіюються із заданого масиву
Array::Array (int Count, float *Data)
{
    this->Count = Count;
    Data = new float[Count];
    for (int i = 0; i < Count; i++)
    {
         this->Data[i] = Data[i];
    }
}

// використання конструкторів у тілі програми
...
Array X; // використано конструктор за замовчуванням
Array A (10); // використано конструктор ініціалізації 1
Array B (10, 0.5); // використано конструктор ініціалізації 2
float m[5] = {0.0, -0.1, 0.2, -0.5 12.6};
Array C (5, m); // використано конструктор ініціалізації 3
```

За наявності у класі конструкторів ініціалізації конструктор за замовчуванням не створюється автоматично.

Конструктором копіювання називається конструктор, який в якості єдиного аргумента приймає посилання на об’єкт (як правило константний) відповідного класу.

Приклад:

```
class Array
{
    int Count;
    float *Data;
public:
    ...
    // оголошення конструктора копіювання
    Array (const Array&);
    ...
};
...
// реалізація конструктора копіювання
```

```
Array::Array (const Array& X)
{
    Count = X.Count;
    Data = new float[Count];
    for (int i = 0; i < Count; i++)
    {
         Data[i] = X.Data[i];
    }
}

// використання конструктора копіювання у тілі програми
...
Array A (10); // використано конструктор ініціалізації
Array B (A); // використано конструктор копіювання
Array C = A; // використано конструктор копіювання
```

Якщо в класі не оголошено конструктор копіювання він створюється компілятором автоматично і при використанні виконує побітове копіювання даних об'єкта-аргумента в створюваний об'єкт.

3. Деструктор

Деструктор – це спеціальний метод класу, який викликається автоматично в момент завершення часу життя об'єкта класу. Головною відзнакою деструктора від інших методів класу полягає в тому, що його назва співпадає з назвою класу, але починається символом "~" (тільда).

Деструктор призначений для виконання певних завершувальних операцій перед знищенням об'єкта класу: звільнення динамічно виділеної пам'яті, закриття файлу, завершення з'єднання з базою даних або мережевого з'єднання тощо.

Деструктор не може ані повертати значення, ані приймати аргументи, тому що його виклик здійснюється повністю неявно.

Приклад:

```
class Array
{
```

```
    int Count;
    float *Data;
public:
    // оголошення конструктора за замовчуванням
    Array ();
    // оголошення конструктора ініціалізації
    Array (int);
    // оголошення деструктора
    ~Array ();
    ...
};


// реалізація деструктора
Array::~Array ()
{
    delete[] Data;
}
```

Методичні вказівки

Лабораторна робота складається з двох завдань, першим з яких є модифікація вже існуючого класу (створеного в процесі виконання лабораторної роботи № 1), а другим – створення нового класу. В кожному завданні необхідно виконати тестування реалізованих класів за допомогою тестуючої програми, яка створює об’єкти класу, використовуючи конструктори різних типів. *Результатом* виконання лабораторної роботи має бути демонстрація коректної логіки функціонування методів реалізованого класу в програмі.

У створюваному класі необхідно реалізувати:

- конструктор за замовчуванням;
- конструктор копіювання;
- конструктори ініціалізації згідно варіанту завдання;
- деструктор;
- методи, передбачені завданням.

При створенні класу не слід обмежуватись реалізацією лише тих *специфічних* методів, які перелічені у тексті завдання. Викладач має право

вимагати реалізації у класі інших, природно необхідних методів. Наприклад, для класу “Динамічний масив” такими методами є читання і запис елементів масиву.

Тестуюча програма повинна створювати і тестувати декілька об’єктів класу, демонструючи роботу всіх реалізованих конструкторів, а також інших методів.

Приклад

Реалізувати та протестувати клас “Динамічний масив” для роботи з речовими числами. Передбачити наступні конструктори ініціалізації:

- *з одним аргументом – кількістю елементів у майбутньому масиві;*
- *з двома аргументами – кількістю елементів масиву та ініціалізуючим значенням для всіх елементів;*
- *з двома аргументами – кількістю елементів масиву та покажчиком на масив речових чисел з ініціалізуючими значеннями;*
- *з трьома аргументами – кількість елементів масиву, значенням першого елементу та кроком, на який змінюється кожен наступний елемент відносно попереднього (в арифметичній прогресії).*

Передбачити методи доступу до елементів масиву (читання і запису), а також обчислення середнього арифметичного його елементів.

Розв’язання:

Проведемо аналіз заданого класу – динамічного масиву. Очевидно, що властивостями, які характеризують масив є кількість елементів у ньому та власне масив, тобто сховище значень.

Для доступу до елементів масиву реалізуємо методи SetValue (запис) та GetValue (читання), які захищатимуть сховище даних від виходу індексу за його межі.

Лістинг програми мовою C++ (IDE BC, BDS/ECRS):

```
// вміст файлу array.hpp
// ------------------------------------------------------------------------------------
// оголошення класу масиву
class Array
{
	// властивості класу
	int Number;
	float *Data;
	// службовий метод динамічного створення масиву
	void NewMemory (int);
	// службовий метод динамічного знищення масиву
	void DelMemory ();
public:
	// конструктор за замовчуванням
	Array ();
	// конструктор ініціалізації 1
	Array (int);
	// конструктор ініціалізації 2
	Array (int, float);
	// конструктор ініціалізації 3
	Array (int, float*);
	// конструктор ініціалізації 4
	Array (int, float, float);
	// конструктор копіювання
	Array (Array&);
	// деструктор
	~Array ();
	// метод динамічного створення масиву
	void Create (int);
	// метод динамічного знищення масиву
	void Destroy ();
	// метод запису заданого значення на задану позицію масиву
	bool SetValue (int, float);
	// метод читання значення із заданої позиції масиву
	float GetValue (int);
	// метод обчислення середнього арифметичного елементів масиву
	float GetAverage ();
};
// вміст файлу array.cpp
// ------------------------------------------------------------------------------------
#include "array.hpp"
Array::Array ()
{
	// конструктор за замовчуванням обнуляє вміст змінних-властивостей
	Number = 0;
	Data = NULL;
}
Array::Array (int Number)
{
	// конструктор ініціалізації 1 викликає метод динамічного створення масиву
	Create (Number, true);
}
Array::Array (int Number, float Value)
{
	// конструктор ініціалізації 1 викликає метод динамічного створення масиву
	// та заповнює масив заданим значенням
	Create (Number, true);
	for (int i = 0; i < Number; i++) Data[i] = Value;
}
Array::Array (int Number, float *Data)
{
```

```
	// конструктор ініціалізації 2 викликає метод динамічного створення масиву
	// та заповнює масив значеннями із іншого масиву
	Create (Number, true);
	for (int i = 0; i < Number; i++) this->Data[i] = Data[i];
}
Array::Array (int Number, float StartValue, float StepValue)
{
	// конструктор ініціалізації 3 викликає метод динамічного створення масиву
	// та заповнює масив значеннями в арифметичній прогресії
	Create (Number, true);
	Data[0] = StartValue;
	for (int i = 1; i < Number; i++) Data[i] = Data[i - 1] + StepValue;
}
Array::Array (Array& AnotherArray)
{
	// конструктор копіювання викликає метод динамічного створення масиву
	// та заповнює масив значеннями з масиву іншого об’єкта
	Create (AnotherArray.Number, true);
	for (int i = 0; i < Number; i++) Data[i] = AnotherArray.Data[i];
}
void Array::NewMemory (int Number)
{
	// метод динамічно виділяє пам’ять
	this->Number = Number;
	Data = new float[Number];
}
void Array::DelMemory ()
{
	// метод динамічно видаляє пам’ять
	delete[] Data;
	Number = 0;
	Data = NULL;
}
void Array::Create (int Number)
{
	// метод перевіряє необхідність динамічного видалення
	// раніше зарезервованої пам'яті
	// і потім динамічно виділяє пам'ять
	if (Number) DelMemory ();
	NewMemory (Number);
}
void Array::Destroy ()
{
	// метод динамічно видаляє лише раніше виділену пам'ять
	if (Number) DelMemory ();
}
Array::~Array ()
{
	// деструктор викликає метод динамічного знищення маисву
	Destroy ();
}
bool Array::SetValue (int Pos, float Value)
{
	// перевірка, чи лежить у коректних межах індекс елемента, що записується
	if (Pos >= 0 && Pos < Number)
	{
		Data[Pos] = Value;
		return true;
	}
	return false;
}
float Array::GetValue (int Pos)
```

```
{
	// перевірка, чи лежить у коректних межах індекс елемента, що зчитується
	if (Number && Pos >= 0 && Pos < Number) return Data[Pos];
	return 0;
}
float Array::GetAverage ()
{
	// перевірка існування масиву
	if (Number)
	{
		// обчислення середнього арифметичного
		float Avg = 0;
		for (int i = 0; i < Number; i++) Avg += Data[i];
		return Avg / Number;
	}
	return 0;
}
// вміст файлу testarray.cpp
// ---------------------------------------------------------------------------------------
#include <conio.h>
#include <stdio.h>
#include "array.hpp"
// основна програма
void main ()
{
	float source[5] = {0.1, -0.2, 0.3, -0.4, 0.5};
	// створення об'єкта класу Array з використанням конструктора за замовчуванням
	Array A;
	// створення об'єкта класу Array з використанням конструктора ініціалізації 1
	Array B (5);
	// створення об'єкта класу Array з використанням конструктора ініціалізації 2
	Array C (10, 3.14);
	// створення об'єкта класу Array з використанням конструктора ініціалізації 3
	Array D (5, source);
	// створення об'єкта класу Array з використанням конструктора ініціалізації 4
	Array E (10, 1.0, -0.1);
	// створення об'єкта класу Array з використанням конструктора копіювання
	Array F (D);
	printf ("Array A is empty: %4.2f\n", A.GetValue (5));
	printf ("\nArray B:\n");
	for (int i = 0; i < 5; i++) printf ("%4.2f ", B.GetValue (i));
	printf ("\nArray C:\n");
	for (int i = 0; i < 10; i++) printf ("%4.2f ", C.GetValue (i));
	printf ("\nArray D:\n");
	for (int i = 0; i < 5; i++) printf ("%4.2f ", D.GetValue (i));
	printf ("\nArray E:\n");
	for (int i = 0; i < 10; i++) printf ("%4.2f ", E.GetValue (i));
	printf ("\nArray F (try to get non-existing element with number 11):\n");
	for (int i = 0; i < 11; i++) printf ("%4.2f ", F.GetValue (i));
	D.Create (10);
	printf ("\nArray D after re-creating and filling:\n");
	for (int i = 0; i < 10; i++) D.SetValue (i, (float) i * i);
	for (int i = 0; i < 10; i++) printf ("%4.2f ", D.GetValue (i));
	F.Destroy ();
	printf ("\nArray F after destroying:\n");
	for (int i = 0; i < 10; i++) printf ("%4.2f ", F.GetValue (i));
	printf ("\nAverage value of array E: %4.2f\n", E.GetAverage ());
	getch ();
}
```

Завдання

1) вивчити теоретичні відомості та методичні вказівки до лабораторної роботи;
2) виконати завдання 1 по удосконаленню класу, реалізованого в лабораторній роботі № 1;
3) змінити тестуючу програму, реалізовану в лабораторній роботі № 1, для перевірки коректності доповнень, зроблених у класі; відкомпілювати та відладити програму;
4) вибрати завдання 2 для виконання згідно з варіантом;
5) реалізувати клас та тестуючу програму;
6) відкомпілювати та відладити програму;
7) відповісти на контрольні запитання;
8) зробити висновки.

Загальне завдання 1: доповнити клас, реалізований у лабораторній роботі № 1 конструктором за замовчуванням, конструктором ініціалізації (з самостійно визначеними аргументами) та конструктором копіювання. Перевірити тестуючу програму, послідовно використовуючи кожен з реалізованих конструкторів при створенні об’єкту класу.

Загальне завдання 2: реалізувати та протестувати клас, який містить конструктор за замовчуванням, конструктор копіювання, конструктори ініціалізації згідно з варіантом та деструктор.

Варіанти завдань (для загального завдання 2)

Варіант	Клас, його властивості та логіка роботи методів
1	**Динамічна матриця** Характеризується кількістю рядків та стовпчиків; передбачена для роботи з речовими числами подвійної точності. Підтримує специфічні методи: • читання і запис елементів із заданими координатами;

	• визначення координат сідлової точки. Конструктори: • з трьома аргументами (третій аргумент – за замовчуванням) – динамічно резервує пам'ять на задану кількість рядків і стовпчиків та заповнює матрицю заданим значенням; якщо значення (третій аргумент) відсутнє, заповнює матрицю випадковими числами з діапазону 0 … 99; • з трьома аргументами – динамічно резервує пам'ять на задану кількість рядків і стовпчиків та заповнює матрицю значеннями з речової матриці, покажчик на яку переданий у третьому аргументі; • з шістьма аргументами – динамічно резервує пам'ять на задану кількість рядків і стовпчиків та заповнює випадковими числами з діапазону 0 … 99 підматрицю, задану парою координат (лівого верхнього та правого нижнього кута).
2	**Бінарний редактор файлів** Характеризується назвою та дескриптором файлу, який редагується. Підтримує специфічні методи: • читання і запис байта на заданій позиції; • визначення розміру файлу. Конструктори: • з одним аргументом – відкриває файл із заданою назвою; • з двома аргументами – відкриває файл із заданою назвою і створює його копію у файлі із йншою заданою назвою (у другому аргументі); • з одним аргументом – відкриває файл із заданою назвою і виконує операцію XOR над кожним його байтом і заданим байтовим значенням.
3	**Динамічний багатокутник** Характеризується кількістю вершин. Підтримує специфічні методи: • читання і запис координат вершин; • визначення площі багатокутника. Конструктори: • з одним аргументом – динамічно виділяє пам'ять під задану кількість вершин; • з двома аргументами – динамічно виділяє пам'ять під задану кількість вершин та ініціалізує координати всіх вершин із заданого одновимірного масиву, який зберігає їх у вигляді x_1, y_1, x_2, y_2, …; • з двома аргументами – динамічно виділяє пам'ять під задану кількість вершин та ініціалізує координати всіх вершин значеннями, які відповідають координатам правильного багатокутника, вписаного у коло із заданим у другому аргументі радіусом.
4	**Копіювальник файлів** Характеризується назвою та дескриптором оригінального та клонованого файлу. Підтримує специфічні методи: • копіювання файлу; • визначення розміру файлу. Конструктори: • з одним аргументом – відкриває файл із заданою назвою; назва копії файлу складається з назви вихідного файлу та доданого рядка "copy" (наприклад: myfile.txt → myfilecopy.txt); • з двома аргументами – відкриває файл із заданою назвою та файл-копію із іншою заданою назвою; • з трьома аргументами – відкриває файл із заданою назвою та файл-копію із іншою заданою назвою; файл-копія створюється в директорії, шлях до якої вказаний третім аргументом.
5	**Динамічний рядок**

	Характеризується кількістю символів у рядку. Підтримує методи: • читання і запис символа на заданій позиції; • пошук першої позиції заданого підрядка в рядку. Конструктори: • з одним аргументом – довжиною рядка; • з одним аргументом – рядком; • з двома аргументами – довжиною рядка та символом, яким треба заповнити рядок.
6	**Статистичний аналізатор файлів** Характеризується назвою, довжиною та дескриптором файлу, який аналізується. Підтримує методи: • визначення наявності у файлі заданого ланцюжка довжиною від 1 до $N / 2$ байтів у файлі розміром N байтів; • визначення кількості повторень заданого ланцюжка довжиною від 1 до $N / 2$ байтів у файлі розміром N байтів. Конструктори: • з одним аргументом – відкриває файл із заданою назвою; • з двома аргументами – відкриває файл із заданою назвою та запам'ятовує значення його довжини, яке буде використовуватись для аналізу, з другого аргументу.
7	**Динамічне регістрове кільце** Характеризується довжиною (кількістю регістрів) та номером вихідного регістра (доступного для редагування вмісту); передбачене для роботи із цілочисельними елементами. Підтримує методи: • читання регістрів кільця; • запис вихідного регістра; • обертання кільця за годинниковою стрілкою; • обертання кільця проти годинникової стрілки. Конструктори: • з одним аргументом – створює регістрове кільце заданої довжини; вихідним регістром встановлює нульовий елемент кільця; • з двома аргументами – створює регістрове кільце заданої довжини та встановлює заданий другим аргументом номер елемента, що відповідає вихідному регістру; • з трьома аргументами – створює регістрове кільце заданої довжини, встановлює номер вихідного регістра, заповнює кільце значеннями із заданого третім аргументом цілочисельного масиву.
8	**Порівняльник файлів** Характеризується назвами, довжинами та дескрипторами двох файлів, які порівнюються між собою. Підтримує методи: • визначення кількості різних байтів у порівнюваних файлах, тобто таких, які розташовані у файлах на однакових позиціях, але мають різні значення; • створення текстового файлу, який містить номери позицій байтів, що мають різні значення у порівнюваних файлах. Конструктори: • з двома аргументами – відкриває файли із заданими назвами; • з трьома аргументами – відкриває файли із заданими назвами та запам'ятовує довжину, на якій буде виконуватись порівняння.
9	**Сортувальник масиву** Характеризується кількістю елементів; передбачений для роботи з цілими числами.

	Підтримує методи: • читання і запис елементів; • сортування деяким методом (повертає покажчик на константний масив). Конструктори: • з одним аргументом – динамічно резервує пам’ять під задану кількість елементів масиву; • з одним аргументом – зчитує кількість елементів масиву та власне елементи з файлу із заданою аргументом назвою; • з двома аргументами – динамічно резервує пам’ять під задану кількість елементів масиву та заповнює масив значеннями із заданого другим аргументом цілочисельного масиву.
10	**Захисник файлів** Характеризується назвою, довжиною та дескриптором файлу, який захищається (шифрується), а також рядком-паролем. Підтримує методи: • кодування файлу за методом байтового гамування; • декодування файлу за аналогічним методом. Конструктори: • з одним аргументом – відкриває файл із заданою назвою; • з двома аргументами – відкриває файл із заданою назвою та запам’ятовує пароль.

Контрольні запитання

1) дайте визначення конструктора і деструктора;
2) перелічіть та поясніть властивості конструктора;
3) перелічіть та поясніть властивості деструктора;
4) дайте визначення конструктора за замовчуванням та назвіть його особливості;
5) дайте визначення конструктора копіювання та назвіть його особливості;
6) поясніть різницю між поверхневим та глибинним копіюванням, яке виконує конструктор копіювання.

Лабораторна робота № 3

ПЕРЕВАНТАЖЕННЯ ОПЕРАТОРІВ

Мета роботи:

- вивчити правила та особливості перевантаження операторів в мові C++;
- закріпити на практиці знання протоколу перевантаження операторів;
- набути вміння використовувати перевантаження операторів для збільшення зручності використання об’єктів класу.

Теоретичні відомості

1. Вступ (постановка проблеми)

Для виконання маніпуляцій над об’єктами класу, як правило, використовуються їх власні методи. Розглянемо приклад реалізації класу комплексних чисел з можливістю їх додавання:

```
class Complex
{
    float Re, Im;
public:
    Complex ();
    Complex (float, float);
    // метод - додавання
    Complex Add (Complex&);
    ...
};
Complex::Complex ()
{
    Re = Im = 0;
}
Complex::Complex (float Re, float Im)
{
    this->Re = Re;
    this->Im = Im;
```

```
}
Complex Complex::Add (Complex& X)
{
    return Complex (Re + X.Re, Im + X.Im);
}
```

Як видно з прикладу метод Add, призначений для додавання двох комплексних чисел, повертає об’єкт класу Complex, створений з використанням конструктора ініціалізації. Параметрами, які передаються в конструктор, є суми відповідних складових двох комплексних чисел, що надходять з наступних джерел: власне об’єкта, для якого викликається метод, та об’єкта, посилання на який передається у вигляді аргумента.

Використання метода Add у програмі матиме наступний вигляд:

```
Complex A (2, 3);
Complex B (-4, 5);
Complex C;
...
// додаються складові this->Re + B.Re та this->Im + B.Im
// покажчик this посилається на об’єкт,
// для якого викликано метод, тобто на A
C = A.Add (B);
```

Іншим способом реалізації додавання комплексних чисел є створення дружньої функції, аргументами та вихідним значенням якої є посилання на (об’єкт) класу Complex. Оголошення класу в цьому випадку повинно включати декларацію дружньої функції:

```
class Complex
{
    friend Complex Add (Complex&, Complex&);
    ...
};
...
Complex Add (Complex& X, Complex& Y)
{
```

```
    return Complex (X.Re + Y.Re, X.Im + Y.Im);
}
...
Complex A (2, 3);
Complex B (-4, 5);
Complex C;
...
// додаються складові A.Re + B.Re та A.Im + B.Im
C = Add (A, B);
```

Обидва наведені способи реалізації не забезпечують такого рівня наочності виконання додавання, який досягається при застосуванні “звичайного” оператора додавання (“+”) зі змінними базових (атомарних) типів:

```
int a, b, c;
float p, q, r;
...
c = a + b;
r = p + q;
```

2. Основи перевантаження операторів

Перевантаження операторів це механізм, який дозволяє надати операторам та операціям C++ можливість працювати з об’єктами класів.

Мова C++ дозволяє програмісту перевантажувати більшість операторів та наділяти їх чутливістю до контексту, в якому вони використовуються.

Не дозволяється перевантажувати наступні оператори: “.”, “.*”, “::”, “?:”.

Перевантаженням не можна змінити пріоритет операції, їх асоціативність та кількість операндів (“арність”), а також створити новий оператор. Не можна, також, перевантажити оператор з метою зміни сенсу його роботи по відношенню до базових (атомарних) типів.

Складні оператори, наприклад, “+=”, перевантажуються окремо, тобто наявність перевантаженого оператора додавання (“+”) та оператора привласнення (“=”) не означає автоматичного неявного перевантаженого оператора додавання з накопиченням (“+=”).

3. Способи перевантаження операторів

Оператори можуть перевантажуватись як методи класу та як дружні функції. Розглянемо різницю між цими способами на прикладах.

Перевантаження оператора додавання, як метода класу:

```
class Complex
{
    float Re, Im;
public:
    Complex ();
    Complex (float, float);
    // оператор додавання двох комплексних чисел
    Complex operator + (Complex&);
    // оператор додавання комплексного та дійсного числа
    // Увага! Дійсне число може бути лише правим операндом!
    Complex operator + (float);
    ...
};
...
Complex Complex::operator + (Complex& X)
{
    return Complex (Re + X.Re, Im + X.Im);
}
Complex Complex::operator + (float X)
{
    return Complex (Re + X, Im);
}
...
Complex A (2, 3);
Complex B (-3, 7);
Complex C, D, E;
...
// використовується перший перевантажений оператор "+"
C = A + B;
```

```
// використовується другий перевантажений оператор "+"
D = C + 1.5;
// Помилка! Працює лише якщо аргументи поміняються місцями!
E = 0.8 + D;
```

Перевантаження оператора додавання, як дружньої функції:

```
class Complex
{
    friend Complex operator + (Complex&, Complex&);
    friend Complex operator + (Complex&, float);
    friend Complex operator + (float, Complex&);
    float Re, Im;
public:
    Complex ();
    Complex (float, float);
    ...
};
...
Complex operator + (Complex& X, Complex& Y)
{
    return Complex (X.Re + Y.Re, X.Im + Y.Im);
}
Complex operator + (Complex& X, float Y)
{
    return Complex (X.Re + Y, X.Im);
}
Complex operator + (float X, Complex& Y)
{
    return Complex (X + Y.Re, Y.Im);
}
...
Complex A (2, 3);
Complex B (-3, 7);
Complex C, D, E;
...
```

```
// використовується перший перевантажений оператор "+"
C = A + B;
// використовується другий перевантажений оператор "+"
D = C + 1.5;
// використовується третій перевантажений оператор "+"
E = 0.8 + D;
```

Як видно з прикладів, перевантаження операторів як дружніх функцій усуває обмеження на порядок передачі аргументів. В той же час дружність порушує інкапсуляцію, тому її застосування не завжди є виправданим.

Оператори “( )”, “[ ]”, “->” та “=” можуть бути перевантажені лише як методи класу.

Методичні вказівки

Завданням даної лабораторної роботи є модифікація класу, створеного в процесі виконання лабораторної роботи № 1, шляхом додавання в нього (або в модуль, в якому він розміщений) перевантажених операторів. *Результатом* виконання лабораторної роботи має бути демонстрація коректної логіки функціонування перевантажених операторів в програмі.

Приклад

Доповнити клас “Принтер”, реалізований в процесі виконання лабораторної роботи № 1 наступними перевантаженими операторами:

- *“+=” – для завантаження паперу;*
- *“-=” – для вивантаження паперу;*
- *“*=” – для заправки чорнилами;*
- *“=” – для вмикання (true) та вимикання (false).*

Розв’язання:

Оператори “+=”, “-=” та “=” реалізуємо як методи класу, а оператор

“*=” – як дружню функцію.

Фрагмент модифікованого модуля printer.h:

```
...
class Printer
{
	// оголошення оператора “*=” (аналога метода LoadInk) як дружньої функції
	friend int operator *= (Printer&, int);
	bool Power;
	int State;
	int Paper;
	float Ink;
	void SetState (int);
public:
	void Reset (void);
	void TurnOn (void);
	void TurnOff (void);
	// оголошення оператора “=” (аналога методів TurnOn і TurnOff) як метода класу
	void operator = (bool);
	bool GetPower (void) {return Power;}
	int LoadPaper (int);
	// оголошення оператора “+=” (аналога метода LoadPaper) як метода класу
	int operator += (int);
	int UnloadPaper (int);
	// оголошення оператора “-=” (аналога метода UnloadPaper) як метода класу
	int operator -= (int);
	int GetPaper (void) {return Paper;}
	int LoadInk (int);
	int GetInk (void) {return Ink;}
	int Print (int, int);
	int GetState (void);
	const char* const GetStateName () {return PrinterStateName[State];}
};
```

Фрагмент модифікованого модуля printer.cpp:

```
...
// реалізація оператора – дружньої функції
int operator *= (Printer& P, int Drops)
{
	if (P.Power) return 0;
	if (Drops < 0) return 0;
	int NewInk = P.Ink + Drops;
	if (NewInk > MAXINK)
	{
		P.Ink = MAXINK;
		return MAXINK - Drops;
	}
	P.Ink = NewInk;
	return Drops;
}
...
// реалізація оператора – метода класу
void Printer::operator = (bool State)
{
	State ? TurnOn () : TurnOff ();
```

```
}
...
// реалізація оператора – метода класу
int Printer::operator += (int Sheets)
{
	return LoadPaper (Sheets);
}
...
// реалізація оператора – метода класу
int Printer::operator -= (int Sheets)
{
	return UnloadPaper (Sheets);
}
...
```

Фрагмент тестуючої програми testprn.cpp:

```
...
switch (key)
{
	// використання оператора "="
	case '0': Canon = false; break;
	case '1': Canon = true; break;
	case '2':
	{
		int Sheets;
		printf ("Input the number of sheets to load: ");
		scanf ("%d", &Sheets);
		// використання оператора "+="
		Canon += Sheets;
		break;
	}
	case '3':
	{
		int Sheets;
		printf ("Input the number of sheets to unload: ");
		scanf ("%d", &Sheets);
		// використання оператора "-="
		Canon -= Sheets;
		break;
	}
	case '4':
	{
		int Drops;
		printf ("Input the volume of ink to load: ");
		scanf ("%d", &Drops);
		// використання оператора "*="
		Canon *= Drops;
		break;
	}
	...
```

Завдання

1) вивчити теоретичні відомості та методичні вказівки до лабораторної роботи;
2) вибрати завдання для виконання згідно з варіантом;
3) доповнити клас, реалізований в процесі виконання лабораторної роботи № 1, перевантаженими операторами згідно варіанту;
4) модифікувати тестуючу програму, замінивши в ній виклики методів на виклики відповідниїх перевантажених операторів;
5) відкомпілювати та відладити програму;
6) відповісти на контрольні запитання;
7) зробити висновки.

Загальне завдання: доповнити клас, реалізований в процесі виконання лабораторної роботи № 1, перевантаженими операторами згідно з варіантом.

Варіанти завдань

Варіант	Оператори, які необхідно перевантажити та спосіб перевантаження
1	“=” (привласнення true/false) – вмикання/вимикання; “++” та “– –” (у префіксній формі) – запуск збільшення та зменшення частоти відповідно; “++” та “– –” (у постфіксній формі) – завершення збільшення та зменшення частоти відповідно.
2	“=” (привласнення числа) – встановлення температури; “<<” (із значенням true/false) – відчинення/зачинення дверці.
3	“=” (привласнення true/false) – підключення/відключення лінії; “*=” (з числом) – встановлення рівня гучності; “|=” (з рядком, який містить номер) – набір номеру.
4	“=” (привласнення true/false) – вмикання/вимикання; “+=” (з числом) – запуск встановлення рівня води в баку; “++” та “– –” (у постфіксній формі) – запуск роботи/зупинка.
5	“+=” (з числом) – завантаження кабіни; “++” та “– –” (у префіксній формі) – старт/кінець виклику.
6	“=” (привласнення true/false) – вмикання/вимикання живлення; “<<=” (з числом) – встановлення потужності мікрохвиль; “^=” (з числом) – встановлення часу за таймером.
7	“=” (привласнення true/false) – підключення/відключення електромережі; “++” та “– –” (у постфіксній формі) – вмикання/вимикання; “– –” (у префіксній формі) – випорожнення контейнера пилу.

8	“=” (привласнення true/false) – підключення/відключення електромережі; “+=” та “– =” (з числом) – набір/злив води; “&=” – встановлення цільової температури.
9	“=” (привласнення true/false) – вмикання/вимикання; “+=” та “– =” (з числом) – встановлення/витягнення диску; “++” та “– –” (у префіксній формі) – старт/завершення програвання.
10	“+=” (з рядком – паролем) – авторизація; “==” (з true/false) – визначення рівня прав (false – гість, true – адміністратор); “=” (з числом) – зміна обсягу наявних грошей; “>>” (з посиланням) – зняття грошей.
11	“=” (привласнення true/false) – вмикання/вимикання; “>>” (з посиланням) – перевірка будильника; “|=” (з true/false) – встановлення режиму (false – AM, true – PM).
12	“+=” (з числом) – вмикання на заданий час; “– –” (у префіксній формі) – вимикання; “^=” (з числом) – зміна положення регулятора на заданий час.
13	“=” (привласнення true/false) – вмикання/вимикання; “+=” (з числом) – набір води в паровик; “<<” (з числом) – перемикання режиму роботи; “– –” (у постфіксній формі) – паровий удар.
14	“=” (привласнення true/false) – вмикання/вимикання; “++” (у постфіксній формі) – заміна фільтру; “&” (з числом) – вибір цільової температури.
15	“=” (привласнення true/false) – вмикання/вимикання; “=” (привласнення числа) – натиснення клавіши з цифрою; “>>” – (з посиланням) – натиснення клавіши “дорівнює”.

Контрольні запитання

1) дайте визначення перевантаження оператора;
2) сформулюйте правила перевантаження операторів;
3) назвіть способи перевантаження операторів у C++;
4) перелічіть, які оператори в C++ перевантажувати не можна;
5) поясніть, в чому полягає відзнака перевантаження операторів, як методів класу від операторів, як дружніх функцій;
6) поясніть особливості перевантаження оператора привласнення.

Лабораторна робота № 4

РЕАЛІЗАЦІЯ УСПАДКУВАННЯ

Мета роботи:

- вивчити принципи та правила реалізації успадкування в мові С++;
- закріпити на практиці знання протоколу успадкування;
- набути вміння використовувати успадкування для побудови ієрархічних класових структур для розв'язання різних класів задач.

Теоретичні відомості

1. Вступ

Успадкування – це спосіб повторного використання програмного коду, при якому нові класи створюються на основі вже існуючих шляхом доповнення їх можливостями нових класів.

При створенні нового класу замість написання повністю нових властивостей і методів програміст може вказати, що новий клас може *успадковувати* властивості і методи існуючого класу. Цей існуючий клас називається *базовим* або *батьківським* класом, а новий клас – *похідним* класом, або класом-*нащадком*.

Мова С++ підтримує як *просте*, так і *множинне успадкування*. При простому успадкуванні похідний клас створюється на основі одного базового класу. При множинному успадкуванні похідний клас створюється на основі декількох базових класів.

2. Реалізація простого успадкування

Протокол успадкування в С++ має наступний вигляд:

```
class <Назва похідного класу>: <кваліфікатор> <Назва базового класу>
{
    ...
```

```
};
```

Поле “кваліфікатор” забезпечує три типи успадкування: **public** (відкрите), **protected** (захищене) і **private** (закрите).

При відкритому успадкуванні властивості і методи базового класу зберігають рівень прав доступу відповідно до кваліфікаторів, з якими вони були оголошені.

При захищеному успадкуванні приватні елементи базового класу залишаються приватними в похідному класі, а відкриті стають захищеними.

При закритому успадкуванні всі елементи базового класу стають приватними в похідному класі.

Найчастіше використовується відкрите (public) успадкування.

3. Використання конструкторів і деструкторів у ланцюгах успадкування

При створенні об’єкта похідного класу виклик конструкторів здійснюється послідовно від конструктора базового класу до конструктора похідного класу, який замикає ланцюг. Кожен конструктор при цьому ініціалізує дані власного класу.

Конструктор похідного класу може викликати конструктор базового класу явно або неявно.

Коли об’єкт похідного класу знищується, деструктори викликаються у порядку, зворотньому до виклику конструкторів.

Приклад:

```
// батьківський клас
class AbstractForm
{
     int x, y;
protected:
     int color;
public:
     AbstractForm ();
```

```
     AbstractForm (int, int, int);
     void SetX (int x) {this->x = x;}
     void SetY (int y) {this->y = y;}
     ...
};
// конструктор за замовчуванням базового класу
AbstractForm::AbstractForm ()
{
     x = y = 0;
     color = 0;
}
// конструктор ініціалізації базового класу
AbstractForm::AbstractForm (int x, int y, int color)
{
     this->x = x;
     this->y = y;
     this->color = color;
}
// відкрите успадкування:
// клас Rectangle доповнює клас AbstractForm
// новими властивостями і методами
class Rectangle: public AbstractForm
{
     int w, h;
public:
     Rectangle ();
     Rectangle (int, int, int, int, int);
     void SetPosition (int, int);
     void SetColor (int);
     void SetSize (int, int);
     ...
};
// конструктор за замовчуванням похідного класу
Rectangle::Rectangle ()
// здіснює явний виклик конструктора за замовчуванням базового класу
: AbstractForm ()
{
     w = h = 0;
}
// конструктор ініціалізації похідного класу
Rectangle::Rectangle (int x, int y, int w, int h, int color)
// здіснює явний виклик конструктора ініціалізації базового класу
// і передає йому відповідну частину своїх аргументів
```

```
: AbstractForm (x, y, color)
{
     this->w = w;
     this->h = h;
}
void Rectangle::SetPosition (int x, int y)
{
     // this->x = x; – так не можна: x є приватним в базовому класі
     // this->y = y; – так не можна: y є приватним в базовому класі
     SetX (x); // Ok – використано відкритий метод доступу базового класу
     SetY (y); // Ok – використано відкритий метод доступу базового класу
}
void Rectangle::SetColor (int c)
{
     color = c; // Ok – color є захищеним у базовому класі
}
void Rectangle::SetSize (int w, int h)
{
     this->w = w;
     this->h = h;
}
```

Методичні вказівки

Завданням даної лабораторної роботи є створення, компіляція, відладка та виконання програми, в якій реалізується і тестується певна ієрархія успадкування класів. *Результатом* виконання лабораторної роботи має бути демонстрація коректної логіки функціонування методів похідного класу в програмі.

Для кожного створюваного класу в ієрархії необхідно реалізувати конструктор за замовчуванням, конструктори ініціалізації та копіювання.

Приклад

Реалізувати базовий клас "Птах" з наступними властивостями: розмах крил, швидкість руху, висота польоту та методами, які дозволяють тримувати доступ до них. На основі даного класу утворити два похідних

класи: "Горобець" та "Страус". Клас "Горобець" доповнити масою. Клас "Страус"доповнити довжиною шиї. В обох класах реалізувати метод польоту. У кожному класі реалізувати конструктор за замовчуванням та конструктор ініціалізації.

Розв'язання:

```
// базовий клас – "Птах"
class Bird
{
	float Wingspan;
	float FlyHeight;
protected:
	void LiftUp (float);
public:
	Bird ();
	Bird (float);
	void SetWingspan (float);
	float GetWingspan () {return Wingspan;}
	float GetFlyHeight () {return FlyHeight;}
};
Bird::Bird ()
{
	Wingspan = FlyHeight = 0;
}
Bird::Bird (float Wingspan)
{
	SetWingspan (Wingspan);
	FlyHeight = 0;
}
void Bird::SetWingspan (float Wingspan)
{
	this->Wingspan = (Wingspan > 0) ? Wingspan : 0;
}
void Bird::LiftUp (float FlyHeight)
{
	this->FlyHeight = (FlyHeight >= 0) ? FlyHeight : 0;
}
// похідний клас – "Горобець"
class Sparrow : public Bird
{
	float Weight;
public:
	Sparrow ();
	Sparrow (float, float);
	void SetWeight (float);
	float GetWeight () {return Weight;}
	void Fly ();
};
Sparrow::Sparrow ()
: Bird ()
{
	Weight = 0;
}
Sparrow::Sparrow (float Wingspan, float Weight)
: Bird (Wingspan)
{
```

```
	SetWeight (Weight);
}
void Sparrow::SetWeight (float Weight)
{
	this->Weight = (Weight > 0) ? Weight : 0;
}
void Sparrow::Fly ()
{
	LiftUp (10);
}
// похідний клас – "Страус"
class Ostrich : public Bird
{
	float NeckLength;
public:
	Ostrich ();
	Ostrich (float, float);
	void SetNeckLength (float);
	float GetNeckLength () {return NeckLength;}
	void Fly ();
};
Ostrich::Ostrich ()
: Bird ()
{
	NeckLength = 0;
}
Ostrich::Ostrich(float Wingspan, float NeckLength)
: Bird (Wingspan)
{
	SetNeckLength (NeckLength);
}
void Ostrich::SetNeckLength (float NeckLength)
{
	this->NeckLength = (NeckLength > 0) ? NeckLength : 0;
}
void Ostrich::Fly ()
{
	LiftUp (0.5);
}
#include <conio.h>
#include <iostream.h>
// тестова програма
void main ()
{
  Sparrow Chirik;
  Ostrich Nandu;

  Chirik.SetWeight (150);
  Nandu.SetNeckLength (50);
  Chirik.SetWingspan (12.5);
  Nandu.SetWingspan (150);
  clrscr ();
  cout << "Weight of sparrow: " << Chirik.GetWeight() << " g" << endl;
  cout << "Length of neck of ostrich: " << Nandu.GetNeckLength() << " sm" << endl;
  cout << "Wingspan of sparrow: " << Chirik.GetWingspan () / 10 << " m" << endl;
  cout << "Wingspan of neck of ostrich: " << Nandu.GetWingspan () / 10 << " m" << endl;
  Chirik.Fly ();
  Nandu.Fly ();
  cout << "Height of fly of sparrow: " << Chirik.GetFlyHeight () << " m" << endl;
  cout << "Height of fly of neck of ostrich: " << Nandu.GetFlyHeight () << " m" << endl;
  getch ();
}
```

Завдання

1) вивчити теоретичні відомості та методичні вказівки до лабораторної роботи;
2) реалізувати задану ієрархію класів та тестуючу програму;
3) відкомпілювати та відладити програму;
4) відповісти на контрольні запитання;
5) зробити висновки.

Загальне завдання: реалізувати та протестувати ієрархію класів (батьківський та похідні) згідно з варіантом.

Варіанти завдань

Варіант	Ієрархія класів, їх властивості та логіка роботи методів	
	Батьківський клас	**Класи-нащадки**
1	**“Число”** Властивості: • кількість десяткових цифр (розрядність); • масив цифр; • знак. Методи: • зміна розрядності; • зміна знаку.	**“Ціле число”** Методи: • порозрядний зсув праворуч і ліворуч; • оператори додавання та віднімання. **“Речове число”** Властивості: • позиція десяткової точки. Методи: • округлення до заданого знаку; • оператори додавання та віднімання.
2	**“Масив”** Властивості: • кількість елементів; • динамічне сховище значень. Методи: • зміна розміру.	**“Стек”** Властивості: • глибина. Методи: • запис та вийняття значення вершини; • читання значення вершини без вийняття. **“Черга”** Властивості: • позиція контрольної точки. Методи: • додавання у чергу; • читання значення з контрольної точки; • читання значення з кінця черги.
3	**“Рівняння”** Властивості: • кількість коренів; • динамічний масив коренів. Методи: • читання коренів.	**“Лінійне рівняння”** Властивості: • коефіцієнти *A, B.* Методи: • обчислення кореня. **“Квадратне рівняння”**

		Властивості: • коефіцієнти *A*, *B*, *C*; • дискримінант. Методи: • обчислення дискримінанту; • обчислення коренів.
4	**“Файл”** Властивості: • назва; • довжина у байтах; • режим доступу (читання/запис). Методи: • зміна назви; • встановлення режиму доступу.	**“Бінарний файл”** Властивості: • розмір елемента даних. Методи: • читання елемента даних; • запис елемента даних. **“Текстовий файл”** Властивості: • максимальна довжина рядка. Методи: • читання рядка; • запис рядка.
5	**“Об’єкт у просторі”** Властивості: • координати. Методи: • переміщення.	**“Паралелепіпед”** Властивості: • розміри (довжина, ширина, висота). Методи: • зміна розмірів; • пропорційна зміна розмірів для отримання заданого об’єму. **“Сфера”** Властивості: • радіус. Методи: • зміна радіусу; • зміна радіусу для отримання заданого об’єму.
6	**“Фрагмент послідовності”** Властивості: • розмір; • динамічне сховище значень. Методи: • зміна розміру.	**“Арифметична прогресія”** Властивості: • початковий елемент; • крок. Методи: • зміна параметрів; • визначення елемента за його порядковим номером (в межах вихідного масиву); • обчислення суми елементів. **“Послідовність Фібоначі”** Методи: • визначення елемента за його порядковим номером (в межах вихідного масиву); • обчислення відношення елемента (задається порядковим номером) до його сусіда зліва (в межах вихідного масиву).
7	**“Трикутник”** Властивості: • довжини сторін. Методи: • зміна довжин сторін.	**“Рівнобічний трикутник”** Методи: • обчислення периметра; • обчислення площі. **“Прямокутний трикутник”** Методи: • обчислення висоти до гіпотенузи; • обчислення периметра; • обчислення площі.
8	**“Абстрактна команда”** Властивості: • адресність (1, 2 або 3); • масив операндів.	**“Команда додавання”** Властивості: • номер операнда-приймача. Методи:

	Методи: • зміна адресності.	• виконання; • обмін доданків місцями. **“Команда кільцевого зсуву”** Властивості: • напрямок (ліворуч/праворуч). Методи: • виконання.
9	**“Коло”** Властивості: • радіус. Методи: • зміна радіусу.	**“Кільце”** Властивості: • внутрішній радіус. Методи: • обчислення площі; • визначення належності точки. **“Еліпсоїд”** Властивості: • радіуси за вісями *Y* та *Z*. Методи: • обчислення об’єму; • визначення належності точки.
10	**“Джерело напруги”** Властивості: • напруга на виході. Методи: • зміна напруги.	**“Ланцюг”** Властивості: • опір споживача. Методи: • визначення струму у споживачі; • визначення споживаної потужності. **“Акумулятор”** Властивості: • внутрішній опір; • ємність (ампер-годин). Методи: • визначення часу роботи; • визначення струму короткого замикання.

Контрольні запитання

1) дайте визначення успадкування та назвіть цілі з якими воно використовується;
2) перелічіть можливі кваліфікатори успадкування та поясніть їх вплив на доступність елементів батьківського класу для класів-нащадків.
3) розкрийте особливості реалізації та виклику конструкторів класів, що утворюють ланцюг успадкування;
4) охарактеризуйте проблеми, які можуть виникати при множинному успадкуванні, та поясніть механізм їх усунення.

Лабораторна робота № 5

РЕАЛІЗАЦІЯ ДИНАМІЧНОГО ПОЛІМОРФІЗМУ

Мета роботи:

- вивчити принципи реалізації динамічного поліморфізму в мові C++;
- закріпити на практиці зання протоколів взаємодії класів, їх властивостей і методів у ланцюзі успадкування;
- набути вміння використовувати поліморфні методи для управління поведінкою об’єктів, які належать до одного ієрархічного ланцюга успадкування.

Теоретичні відомості

1. Вступ

В узагальненому сенсі *поліморфізм* – це можливість об’єктів з однаковою специфікацією мати різну реалізацію.

На практичному рівні розрізняють *статичний* і *динамічний* поліморфізм. Статичний поліморфізм забезпечує можливість використання одноіменних методів з різними наборами параметрів в одному класі і реалізується *перевантаженням методів*. Відомо, що при використанні перевантаженних методів їх адреси підставляються на місця виклику в програмі на етапі компіляції, що відповідає статичній (“жорсткій”) поведінці коду.

Динамічний поліморфізм (далі просто – поліморфізм), у свою чергу, забезпечує можливість використання одноіменних методів з однаковими наборами параметрів в групі класів, пов’язаних відношенням успадкування.

2. Реалізація поліморфізму

Для реалізації поліморфізму в базовому класі необхідно оголосити один або декілька методів, які повинні бути визначені для кожного класу в ієрархії успадкування, а у похідних класах надати власні реалізації цих методів.

Методи базового класу, які планується реалізувати подібним чином, називаються *віртуальними* і повинні оголошуватись з модифікатором **virtual**.

Приклад:

```
// батьківський клас
class AbstractForm
{
     int x, y;
protected:
     int color;
public:
     ...
     // поліморфний метод обчислення площі
     virtual float GetArea ();
     ...
};
...
// у базовому класі поліморфний метод обчислення площі повертає нуль
float AbstractForm::GetArea ()
{
     return 0;
}
...
class Rectangle: public AbstractForm
{
     int w, h;
public:
     ...
     // поліморфний метод обчислення площі для прямокутника
     virtual float GetArea ();
     ...
};
...
// у похідному класі поліморфний метод визначає площу певним шляхом
float Rectangle::GetArea ()
{
     return w * h;
}
...
class Circle: public AbstractForm
```

```
{
     int r;
public:
     ...
     // поліморфний метод обчислення площі для кола
     virtual float GetArea ();
     ...
};
...
// у похідному класі поліморфний метод визначає площу певним шляхом
float Circle::GetArea ()
{
     return M_PI * r * r;
}
...
```

Якщо оголошення віртуального методу в базовому класі має за мету лише створення певного *інтерфейсу*, який повинні реалізовувати похідні класи, а реалізація цього методу в базовому класі не планується (через відсутність сенсу), віртуальний метод можна оголосити таким, який взагалі не може бути реалізованим. Такий метод називається *повністю віртуальним*.

Якщо клас містить хоча б одним повністю віртуальний метод, він називається *абстрактним*. Об’єкти абстрактного класу створювати не можна.

Приклад:

```
// абстрактний батьківський клас
class AbstractForm
{
     int x, y;
protected:
     int color;
public:
     ...
     // повністю віртуальний метод
     virtual float GetArea () = 0;
     ...
};
...
```

```
void main ()
{
     // Увага! Наступні рядки – помилкові!
     // Об’єкти абстрактного класу створювати не можна!
     AbstractForm F;
     AbstractForm *PF = new AbstractForm;
     ...
}
```

3. Використання поліморфних (віртуальних) методів

Ключовим моментом у використанні поліморфних методів є те, що об’єкт похідного класу може розглядатись як об’єкт власного базового класу. Це дозволяє маніпулювати поліморфними методами.

Приклад:

```
void main ()
{
     AbstractForm *F;
     Rectangle *R = new Rectangle;
     Circle *C = new Circle;
     ...
     // покажчик батьківського типу ініціалізується адресою нащадка
     F = R;
     // реалізація поліморфного методу GetArea для класу прямокутника
     // викликається через покажчик батьківського типу
     cout << “Площа прямокутника: ” << F->GetArea () << endl;
     F = C;
     // реалізація поліморфного методу GetArea для класу кола
     // викликається через покажчик батьківського типу
     cout << “Площа кола: ” << F->GetArea () << endl;
     ...
}
```

В наведеному прикладі поліморфний метод обчислення площі фігури – GetArea – викликається через покажчик на батьківського типу – F. Залежно від того, на об’єкт якого похідного класу в даний момент посилається цей

покажчик, здійснюється виклик відповідної реалізації поліморфного методу, причому визначення його конкретної реалізації відбувається динамічно, в процесі виконання програми (run-time).

Методичні вказівки

Завданням даної лабораторної роботи є створення, компіляція, відладка та виконання програми, в якій реалізується певна ієрархія успадкування класів з певним поліморфним (віртуальним) методом. *Результатом* виконання лабораторної роботи має бути демонстрація коректної логіки функціонування реалізацій поліморфного методу у похідних класах. При плануванні демонстрації необхідно дотримуватись наступних правил:

- залежно від сенсу поліморфного методу його слід оголошувати віртуальним або повністю віртуальним;
- всі об’єкти реалізованих класів у програмі повинні створюватись динамічно;
- виклик реалізацій поліморфного методу у похідних класах повинен здійснюватись через покажчик на базовий клас;
- якщо базовий клас містить деструктор, він повинен оголошуватись віртуальним для запобігання можливим помилкам, таким, як, наприклад, витік пам’яті.

Приклад

Реалізувати метод польоту базового класу “Птах” поліморфним.

Розв’язання:

```
// базовий клас – "Птах" – є абстрактним
class Bird
{
	float Wingspan;
	float FlyHeight;
protected:
	void LiftUp (float);
```

```
public:
	...
	// повністю віртуальний метод польоту
	virtual void Fly () = 0;
};
// похідний клас – "Горобець"
class Sparrow : public Bird
{
	float Weight;
public:
	...
	void Fly ();
};
...
void Sparrow::Fly ()
{
	LiftUp (10);
}
// похідний клас – "Страус"
class Ostrich : public Bird
{
	float NeckLength;
public:
	...
	void Fly ();
};
...
void Ostrich::Fly ()
{
	LiftUp (0.5);
}
#include <conio.h>
#include <iostream.h>
// тестова програма
void main ()
{
  Bird *AbstractBird;
  Sparrow *Chirik = new Sparrow;
  Ostrich *Nandu = new Ostrich;
  clrscr ();
  AbstractBird = Chirik;
  AbstractBird->Fly ();
  AbstractBird = Nandu;
  AbstractBird->Fly ();
  cout << "Height of fly of sparrow: " << Chirik->GetFlyHeight () << " m" << endl;
  cout << "Height of fly of neck of ostrich: " << Nandu->GetFlyHeight () << " m" << endl;
  getch ();
  delete Chirik;
  delete Nandu;
}
```

Завдання

1) вивчити теоретичні відомості та методичні вказівки до лабораторної роботи;
2) реалізувати заданий поліморфний метод та тестуючу програму;

3) відкомпілювати та відладити програму;
4) відповісти на контрольні запитання;
5) зробити висновки.

Загальне завдання: реалізувати та протестувати поліморфний метод в ієрархії класів, створеній у попередній лабораторній роботі згідно з варіантом.

Варіанти завдань

Варіант	Поліморфні методи
1	Оператор додавання Оператор віднімання
2	Читання значення з кінця (вершини)
3	Обчислення коренів
4	Читання даних Запис даних
5	Зміна розмірів для отримання заданого об’єму
6	Визначення елемента за його порядковим номером
7	Обчислення площі Обчислення периметра
8	Виконання
9	Визначення належності точки
10	Визначення сили струму

Контрольні запитання

1) дайте визначення поліморфізму;
2) поясніть різницю між статичним і динамічним поліморфізмом;
3) опишіть принципи реалізації поліморфізму в C++;
4) розкрийте поняття віртуального методу, повністю віртуального методу, абстрактного класу;
5) обгрунтуйте необхідність використання віртуальних деструкторів.

Лабораторна робота № 6

СТВОРЕННЯ ШАБЛОНА КЛАСУ

Мета роботи:

- закріпити на практиці зання протоколу оголошення та використання шаблона класу;
- набути вміння використовувати шаблони класів для забезпечення можливості багаторазового використання коду.

Теоретичні відомості

1. Вступ

Шаблон класу – це засіб мови програмування C++, призначений для кодування узагальнених класів, без їх прив'язки до певних параметрів, наприклад, типів та обсягів даних, значень за замовчуванням.

Шаблони є однією з найбільш потужних можливостей мови програмування C++ по створенню універсального програмного забезпечення, яке можна використовувати повторно.

Шаблони класів дають можливість за допомогою одного фрагмента коду визначати цілий набір взаємозв'язаних класів, які називаються *шаблонними класами*.

2. Оголошення шаблона класу та створення об'єктів на його основі

Шаблони класів часто називають параметризованими типами, тому що вони мають один або більше параметрів типу, які визначають налаштування базового шаблона класу на специфічний тип даних при створенні об'єкта класу.

Оголошення шаблона класу здійснюється за наступним форматом (**зверніть увагу на те, що трикутні дужки використовуються фактично, а квадратні – умовно!**):

```
template <class [назва параметризованого типу]>
```

```
class [назва класу]
{
    ...
    // назва параметризованого типу
    // може використовуватись всередині класу
    // для оголошення властивостей, аргументів тощо
};
```

Приклад:

```
// оголошення шаблона з параметризованим типом – SomeType
template <class SomeType>
class MyClass
{
    // властивість Value матиме тип SomeType
    SomeType Value;
public:
    // конструктор ініціалізації
    // приймає аргумент параметризованого типу
    MyClass (SomeType);
    // метод повертає результат параметризованого типу
    SomeType GetValue ();
};
```

Оголошення **template <**class [назва параметризованого типу]**>** використовується, також, перед реалізацією всіх методів класу, які використовують об’єкти параметризованого типу у своєму тілі. Крім того, після назви класу також вказується назва параметризованого типу.

Приклад:

```
template <class SomeType>
MyClass<SomeType>::MyClass (SomeType Value)
{
```

```
    this->Value = Value;
}
template <class SomeType>
MyClass<SomeType>::SomeType GetValue ()
{
    return Value;
}
```

При створенні об’єкта шаблонного класу назва справжнього типу даних, який повинен замістити шаблонний прототип, вказується після назви класу.

Приклад:

```
// створення та використання об’єкта класу MyClass,
// що базується на типі даних float
MyClass<float> A (3.14);
cout << A.GetValue ();
// створення та використання об’єкта класу MyClass,
// що базується на типі даних char
MyClass<char> A (‘x’);
cout << A.GetValue ();
```

Окрім параметризованих типів даних, в шаблон можуть передаватись так звані *нетипові параметри*. Нетипові параметри дозволяють визначити потрібні параметри класу під час компіляції.

Приклад:

```
// оголошення шаблона класу з нетиповим параметром – Count
template <class SomeType, int Count>
class StaticArray
{
    // використання нетипового параметра
```

```
    SomeType Mem[Count];
public:
    StaticArray (SomeType*);
    ...
};
template <class SomeType, int Count>
StaticArray<SomeType>::StaticArray (SomeType* Mem)
{
    for (int i = 0; i < Count; i++)
    this->Mem[i] = Mem[i];
}
...
void main ()
{
    int Arr[5] = {0.1, 0.2, 0.3, 0.4, 0.5};
    StaticArray<int, 5> Z (Arr);
    ...
}
```

3. Властивості шаблонів

Відносно успадкування шаблони характеризуються наступними властивостями:

- шаблон класу може бути похідним від шаблонного або нешаблонного класу;
- шаблонний та нешаблонний класи можуть бути похідними від шаблона класу.

Відносно дружності шаблони характеризуються наступними властивостями:

- функція, оголошена дружньою всередині шаблона класу, є дружньою по відношенню до кожного шаблонного класу, отриманого з даного шаблона;
- якщо в переліку аргументів дружньої функції, оголошеної всередині шаблона класу, є аргументи параметризованого типу, то

для шаблонного класу дружньою буде лише функція з відповідними типами аргументів.

Методичні вказівки

Завданням даної лабораторної роботи є створення, компіляція, відладка та виконання програми, в якій реалізується шаблон деякого класу, а також оголошуються і тестуються декілька об'єктів на основі шаблонних класів, створених із даного шаблона. *Результатом* виконання лабораторної роботи має бути демонстрація коректного функціонування створених об'єктів незалежно від використаного при їх створенні параметризованого типу даних.

Приклад

Реалізувати та протестувати шаблон класу "Динамічний масив" для роботи з числами параметризованого типу (цілі або речові).

Розв'язання:

В якості параметризованого для класу "Динамічний масив" доцільно вибрати тип даних елементів масиву.

```
// вміст файлу array.hpp
// ---------------------------------------------------------------------------
// оголошення шаблона класу
template <class TypeOfEl>
class Array
{
       int Number;
       // масив має параметризований тип
       TypeOfEl *Data;
       void NewMemory (int);
       void DelMemory ();
public:
       Array ();
       Array (int);
       // деякі конструктори працюють з параметризованим типом
       Array (int, TypeOfEl);
       Array (int, TypeOfEl*);
       Array (int, TypeOfEl, TypeOfEl);
       Array (Array&);
       ~Array ();
       void Create (int, bool);
       void Destroy ();
```

```
        // методи читання, запису та обчислення працюють з параметризованим типом
        bool SetValue (int, TypeOfEl);
        TypeOfEl GetValue (int);
        TypeOfEl GetAverage ();
};
// вміст файлу array.cpp
// ---------------------------------------------------------------------------
#include "array.hpp"
template <class TypeOfEl>
Array<TypeOfEl>::Array ()
{
        Number = 0;
        Data = NULL;
}
template <class TypeOfEl>
Array<TypeOfEl>::Array (int Number)
{
        Create (Number, false);
}
template <class TypeOfEl>
Array<TypeOfEl>::Array (int Number, TypeOfEl Value)
{
        Create (Number, false);
        for (int i = 0; i < Number; i++) Data[i] = Value;
}
template <class TypeOfEl>
Array<TypeOfEl>::Array (int Number, TypeOfEl *Data)
{
        Create (Number, false);
        for (int i = 0; i < Number; i++) this->Data[i] = Data[i];
}
template <class TypeOfEl>
Array<TypeOfEl>::Array (int Number, TypeOfEl StartValue, TypeOfEl StepValue)
{
        Create (Number, false);
        Data[0] = StartValue;
        for (int i = 1; i < Number; i++) Data[i] = Data[i - 1] + StepValue;
}
template <class TypeOfEl>
Array<TypeOfEl>::Array (Array& AnotherArray)
{
        Create (AnotherArray.Number, false);
        for (int i = 0; i < Number; i++) Data[i] = AnotherArray.Data[i];
}
template <class TypeOfEl>
void Array<TypeOfEl>::NewMemory (int Number)
{
        this->Number = Number;
        Data = new TypeOfEl[Number];
}
template <class TypeOfEl>
void Array<TypeOfEl>::DelMemory ()
{
        delete[] Data;
        Number = 0;
        Data = NULL;
}
template <class TypeOfEl>
void Array<TypeOfEl>::Create (int Number, bool Clear)
{
        if (Clear) DelMemory ();
        NewMemory (Number);
```

```
}
template <class TypeOfEl>
void Array<TypeOfEl>::Destroy ()
{
	if (Number) DelMemory ();
}
template <class TypeOfEl>
Array<TypeOfEl>::~Array ()
{
	Destroy ();
}
template <class TypeOfEl>
bool Array<TypeOfEl>::SetValue (int Pos, TypeOfEl Value)
{
	// перевірка, чи лежить у коректних межах індекс елемента, що записується
	if (Pos >= 0 && Pos < Number)
	{
		Data[Pos] = Value;
		return true;
	}
	return false;
}
template <class TypeOfEl>
TypeOfEl Array<TypeOfEl>::GetValue (int Pos)
{
	if (Number && Pos >= 0 && Pos < Number) return Data[Pos];
	return 0;
}
template <class TypeOfEl>
TypeOfEl Array<TypeOfEl>::GetAverage ()
{
	if (Number)
	{
		float Avg = 0;
		for (int i = 0; i < Number; i++) Avg += Data[i];
		return Avg / Number;
	}
	return 0;
}
// вміст файлу testarray.cpp
// ---------------------------------------------------------------------------------
#include <conio.h>
#include <stdio.h>
#include "array.hpp"
// основна програма
void main ()
{
	// створення об'єкта класу Array з параметром float
	Array<float> A;
	// створення об'єкта класу Array з параметром int
	Array<int> B (5, 2, 3);
	printf ("Array A is empty: %4.2f\n", A.GetValue (5));
	printf ("Array B:\n");
	for (int i = 0; i < 5; i++) printf ("%i ", B.GetValue (i));
	printf ("\nAverage value of array A: %4.2f\n", A.GetAverage ());
	printf ("Average value of array B: %i\n", B.GetAverage ());
	getch ();
}
```

Завдання

1) вивчити теоретичні відомості та методичні вказівки до лабораторної роботи;
2) реалізувати заданий шаблон класу та тестуючу програму;
3) відкомпілювати та відладити програму;
4) відповісти на контрольні запитання;
5) зробити висновки.

Загальне завдання: перетворити клас, реалізований згідно завдання 2 лабораторної роботи “Створення конструкторів і деструктора” на шаблон класу з параметром відповідно до власного варіанту та протестувати його.

Варіанти завдань

Варіант	Параметр шаблона
1	Тип елементів матриці
2	Тип даних у файлі, що редагується (байт/слово/подвійне слово)
3	Тип координат
4	Тип розміру файлу (слово/подвійне слово)
5	Тип позиції (байт/слово)
6	Тип розміру файлу (слово/подвійне слово)
7	Тип значень у регістрі
8	Тип розміру файлу (слово/подвійне слово)
9	Тип елементів масиву
10	Тип розміру файлу (слово/подвійне слово)

Контрольні запитання

1) поясніть призначення шаблонів та їх роль у створенні програмного забезпечення;
2) перелічіть властивості шаблонів.

Лабораторна робота № 7

РЕАЛІЗАЦІЯ ПЕРЕХОПЛЕННЯ ВИКЛЮЧЕНЬ

Мета роботи:

- закріпити на практиці зання протоколу перехоплення та обробки виключень;
- навчитись створювати та використовувати власні класи виключень.

Теоретичні відомості

1. Вступ

Виключення – це повідомлення про проблеми, що виникають в процесі виконання програм. Механізм перехоплення та обробки виключень дозволяє програмістам створювати програми, які можуть усувати виключні ситуації або відповідним чином обробляти їх.

Використання техніки перехоплення виключень дозволяє підвищити відмовостійкість програми.

2. Постановка проблеми

Фрагмент узагальненого алгоритму, який зазвичай реалізується у програмах з метою забезпечення їх коректної роботи залежно від деяких обставин, має вигляд, представлений на рисунку 1. З рисунку видно, що алгоритм починається з виконання певної задачі, після чого перевіряється коректність цього виконання. Якщо задача виконана некоректно, обробляється помилка. В іншому випадку процес триває і виконується наступна задача.

Недоліком наведеної форми перехоплення та обробки можливих помилок є змішування логіки програми з логікою перехоплення помилок, що ускладнює читабельність програми, можливість її модифікації та відладки. Крім того, якщо деякі помилки виникають доволі рідко, змішування логіки програми з логікою перехоплення помилок може знизити продуктивність

програми через те, що остання повинна виконувати тестування умов для визначення можливості продовження виконання коду.

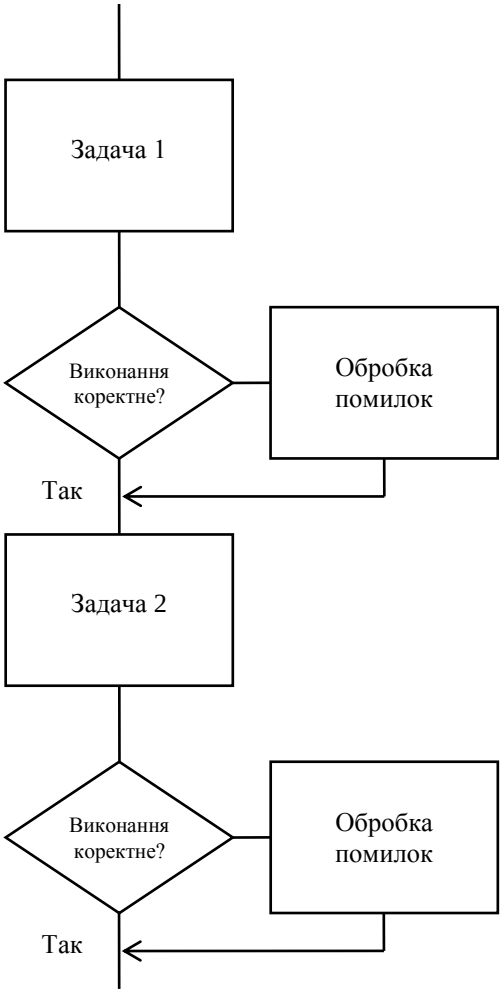

Рисунок 1 – Фрагмент схеми алгоритму перехоплення виключень

Механізм перехоплення та обробки виключень дозволяє програмісту винести оператори обробки помилок з основного коду програми. Це покращує читабельність програми та можлдивість внесення змін до неї.

3. Механізм перехоплення і обробки виключень

У мові C++ для перехоплення та обробки виключень призначена операторна конструкція **try – catch**, яка має наступний узагальнений формат:

```
try
{
    <потенційно небезпечний код>
}
```

```
catch (<клас виключення 1>)
{
    <перелік дій при виникненні виключення типу 1>
}
catch (<клас виключення 2>)
{
    <перелік дій при виникненні виключення типу 2>
}
...
catch (<клас виключення N>)
{
    <перелік дій при виникненні виключення типу N>
}
```

Кожен операнд блока catch вказує тип перехоплюваного та оброблюваного цим блоком виключення. Якщо крім типу вказана, також, назва об'єкта цього типу, то цю назву можна використовувати в обробнику для взаємодії з перехопленим об'єктом виключення.

Приклад:

```
try
{
    // якщо b = 0 виконання наступного рядка
    // призведе до виникнення виключення "Ділення на нуль"
    x = a / b;
}
// передбачається, що існує клас виключення
// під назвою "DivisionByZero"
catch (DivisionByZero&)
{
    cout << "Увага! Відбулось ділення на нуль!" << endl;
    return -1;
}
```

```
...
```

Якщо в блоці try виникає виключення, цей блок припиняє виконання і управління до того блоку catch, параметром якого є відповідний тип виключення. Будь-які інші блоки catch, що відповідають даному блоку try, ігноруються, і виконання коду продовжується після послідовності try – catch.

4. Створення та використання власних виключень

Власні класи виключень можуть створюватись програмістом на основі успадкування від бібліотечного класу **exception**, який належить простору імен **std**. У створюваному класі достатньо визначити конструктор (за замовчуванням або ініціалізації), який передаватиме до конструктора базового класу рядок – повідомлення про помилку. Таке повідомлення далі може бути використане при перехопленні відповідного виключення.

Для генерування виключення деякого типу використовується оператор **throw**, операндом якого є клас виключення.

Приклад:

```
#include <iostream>
#include <exception>
using std::exception;

// оголошення класу виключення
// шляхом його успадкування від бібліотечного класу exception
class TooHighVoltageException : public exception
{
public:
    // конструктор за замовчуванням
    TooHighVoltageException ();
};
// конструктор похідного класу
// викликає конструктор базового класу
// і передає йому рядок із назвою помилки
TooHighVoltageException::TooHighVoltageException ()
: exception ("There is too high voltage in grid!")
{
}
```

```
float GetVoltage (float Power, float Current)
{
    float Voltage = Power / Current;
    // якщо напруга більша за 230 В
    // генерується виключення раніше створеного типу
    if (Voltage > 230) throw TooHighVoltageException ();
    return Voltage;
}
void main ()
{
    ...
    // перехоплення виключення
    try
    {
         float V = GetVoltage (1000, 4);
    }
    // обробка виключення
    catch (TooHighVoltageException& e)
    {
         // використовується метод what класу exception,
         // який повертає рядок з назвою помилки
         cout << "Attention! " << e.what () << endl;
    }
    ...
}
```

Перелік виключень, які можуть генеруватись деякою функцією, може бути вказаний при оголошенні цієї функції. У попередньому прикладі такий перелік може мати наступний вигляд:

```
float GetVoltage (float Power, float Current)
throw (TooHighVoltageException, EDivisionByZero)
{
    float Voltage = Power / Current;
    ...
}
```

В даному прикладі передбачається, що існує бібліотечний клас виключення, пов’язаний із діленням на нуль, який називається EDivisionByZero.

Методичні вказівки

Завданням даної лабораторної роботи є створення, компіляція, відладка та виконання програми, в якій реалізуються і тестуються класи виключень, пов'язані із деякими подіями. *Результатом* виконання лабораторної роботи має бути демонстрація коректної обробки створених виключень.

Приклад

Доповнити клас "Принтер", реалізований в лабораторній роботі № 1, виключеннями, пов'язаними із закінченням паперу та чорнил.

Розв'язання:

```
...
class NoPaperException : public exception
{
public:
	NoPaperException ();
};
NoPaperException::NoPaperException
: exception ("There is no paper in printer!")
{
}
...
class NoInkException : public exception
{
public:
	NoInkException ();
};
NoInkException::NoInkException
: exception ("There is no ink in cartridge!")
{
}
...
class Printer
{
	...
	int UnloadPaper (int) throw (NoPaperException);
	int Print (int, int) throw (NoPaperException, NoInkException);
	...
};
...
int Printer::UnloadPaper (int Sheets) throw (NoPaperException)
{
	if (Sheets < 0) return 0;
	int OldPaper = Paper;
	int NewPaper = Paper - Sheets;
	// генерування виключення
	if (NewPaper < 0) throw NoPaperException ();
	Paper = NewPaper;
```

```
	return Sheets;
}
...
int Printer::Print (int Sheets, int Fill) throw (NoPaperException, NoInkException)
{
	if (!Power || State != READY) return 0;
	if (Sheets < 0 || Fill < 0 || Fill > 100) return 0;
	int LuckSheets;
	if (random (100) < ERRORPROB * 100) LuckSheets = random (Sheets);
	else LuckSheets = Sheets;
	int PosSheetsByInk = Ink * 50 / Fill, PosPaper;
	if (PosSheetsByInk < LuckSheets)
	{
		PosPaper = PosSheetsByInk;
		SetState (NOINK);
		Ink = 0;
		// генерування виключення
		throw NoInkException ();
	}
	else if (Paper < LuckSheets)
	{
		PosPaper = Paper;
		SetState (NOPAPER);
		// генерування виключення
		throw NoPaperException ();
	}
	// вистачить всього
	else
	{
		PosPaper = LuckSheets;
		SetState (READY);
	}
	Ink -= PosPaper * Fill / 50;
	UnloadPaper (PosPaper);
	if (LuckSheets < Sheets) SetState (ERROR);
	return PosPaper;
}
...
		case '3':
		{
			int Sheets;
			printf ("Input the number of sheets to unload: ");
			scanf ("%d", &Sheets);
			// перехоплення виключення
			try
			{
				Canon.UnloadPaper (Sheets);
			}
			catch (NoPaperException& e)
			{
				printf ("\n%s\n", e.what ());
				getch ();
			}
			break;
		}
		...
		case '5':
		{
			int Sheets, Fill;
			printf ("Input the number of sheets and fill: ");
			scanf ("%d%d", &Sheets, &Fill);
			// перехоплення виключення
```

```
                    try
                    {
                        Canon.Print (Sheets, Fill);
                    }
                    catch (NoPaperException& e)
                    {
                        printf ("\nNoPaperException: %s\n", e.what ());
                        getch ();
                    }
                    catch (NoInkException& e)
                    {
                        printf ("\n NoInkException: %s\n", e.what ());
                        getch ();
                    }
                    break;
                }
...
```

Завдання

1) вивчити теоретичні відомості та методичні вказівки до лабораторної роботи;
2) реалізувати класи заданих виключень і тестуючу програму;
3) відкомпілювати та відладити програму;
4) відповісти на контрольні запитання;
5) зробити висновки.

Загальне завдання: доповнити клас, реалізований у лабораторній роботі "Проектування та реалізація класу" виключеннями відповідно до власного варіанту та реалізувати їх обробку у тестуючій програмі.

Варіанти завдань

Варіант	Події, які викликають виключення
1	Вибір неіснуючої станції.
2	Встановлення температури при відчиненій дверці.
3	Прийом сигналу "зайнято".
4	Рівень води в баку поза допустимими межами.
5	Запуск при перевищенні граничного навантаження.
6	Вмикання при відкритій дверці.

7	Переповнення контейнера пилу.
8	Некоректна цільова температура.
9	Старт без диску.
10	Закінчення касової стрічки.
11	Спрацювання будильника.
12	Перевищення допустимої температури плафона.
13	Набір води у паровик у включеному стані.
14	Перевищення загального часу роботи гранично допустимого.
15	Переповнення розрядної сітки.

Контрольні запитання

1) дайти визначення виключення;
2) поясніть переваги механізму перехоплення виключень у порівнянні із звичайним контролем помилок за допомогою розгалужень;
3) опишіть формат операторної конструкції try-catch.

ПЕРЕЛІК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ

1 Б. Страуструп. Язык программирования С++. – М.: “Бином-Пресс”, 2006. – 1104 с.

2 Глушаков С. В., Коваль А. В., Смирнов С. В. Язык программирования С++: Учебный курс. – Харьков.: “Фолио”, 2001. – 500 с.

3 Х. М. Дейтел, П. Дж. Дейтел. Как программировать на С++: Четвертое издание. Пер. с англ. – М.: ООО “Бином-Пресс”, 2005. – 1248 с.

4 Шамис В. А. С++ Builder Borland Developer Studio 2006. Для профессионалов. – СПб.: Питер, 2007. – 781.

5 Шилдт Г. “С++ Руководство для начинающих”. – М.: Издательский дом “Вильямс”, 2005. – 669 с.

Додаток А – Зразок оформлення титульного аркуша звіту з лабораторної роботи

Міністерство освіти і науки України
Промислово-економічний коледж
Національного авіаційного університету

З В І Т

з лабораторної роботи № Х
з дисципліни
“Об’єктно-орієнтоване програмування”
студента ІІІ денного відділення
групи ХХХ – ПОМ
Коваленка Сергія Петровича

Роботу виконав:
студент групи ХХХ – ПОМ
_____ С. П. Коваленко

Перевірив викладач:
_____ В. В. Левченко

м. Київ 20ХХ

Тема роботи: “*Назва теми лабораторної роботи*”.

Мета роботи: *мета лабораторної роботи*.

Завдання № X

Текст завдання згідно варіанту.

Розв’язання

Текст програми розв’язання задачі.

Висновки: *текст висновків за результатами виконання*.