МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Хмельницький політехнічний фаховий коледж

Національного університету «Львівська політехніка»

МЕТОДИЧНІ ВКАЗІВКИ

до лабораторних робіт

з курсу “Об’єктно-орієнтоване програмування ”

для студентів спеціальності

121 Інженерія програмного забезпечення

УКЛАДАЧ: ГУМЕННА В.В.

Розглянуто та схвалено

на засіданні циклової комісії програмного забезпечення

(протокол № 2 від 22.09.2022р.)

2022

Методичні вказівки складені згідно з програмою курсу “Об'єктно-орієнтованого програмування” для спеціальності 121 Інженерія програмного забезпечення і призначені для виконання лабораторних робіт студентів

Автор:

Викладач -методист В.В. Гуменна

Забороняється тиражування та розповсюдження без відома автора.

**ЗМІСТ**

[ВСТУП 5](#_Toc87542157)

[Лабораторна робота №1. Класи й об'єкти в С++ 6](#_Toc87542158)

[Лабораторна робота №2. Перевантаження операцій. 23](#_Toc87542159)

[Лабораторна робота №3. Успадкування і віртуальні функції 34](#_Toc87542160)

[**Лабораторна робота № 4 Побудова множинного наслідування класів.(С++ ) Інтерфейси- С#** 67](#_Toc87542161)

[Лабораторна робота №5 Ієрархія об'єктів і групи. Агрегація.Ітератори. 80](#_Toc87542162)

[Тут *gr-об'єкт-група*. 83](#_Toc87542163)

[Лабораторна робота №6. Делегати. 97](#_Toc87542164)

[Лабораторна робота №7. Шаблони функцій і класів.Контейнери. STL. Колекції в С#. Серіалізація і Десеріалізація 108](#_Toc87542165)

[Стандартна бібліотека шаблонів 110](#_Toc87542166)

[**Атрибут Serializable** 123](#_Toc87542167)

[**Формат серіалізації** 124](#_Toc87542168)

[**Бінарна серіалізація. BinaryFormatter** 125](#_Toc87542169)

[**Серіалізація в формат SOAP. SoapFormatter** 128](#_Toc87542170)

[**Серіалізація в XML. XmlSerializer** 130](#_Toc87542171)

[**Серіалізація в JSON. DataContractJsonSerializer** 135](#_Toc87542172)

[Лабораторна робота №8. Робота з формами. Обробка подій. 144](#_Toc87542173)

[Лабораторна робота №9. Проектування класів n та компонент. 149](#_Toc87542174)

[Проект «Каток» (katok.exe) 149](#_Toc87542175)

[Базовий матеріал 149](#_Toc87542176)

[Детальний опис функціонування проекту 149](#_Toc87542177)

[План розробки 150](#_Toc87542178)

[Завдання для вдосконалення проекту 152](#_Toc87542179)

[Завдання для самостійної розробки 152](#_Toc87542180)

[Проект «Редактор візерунків» (stitch.exe) 152](#_Toc87542181)

[Детальний опис функціонування проекту 152](#_Toc87542182)

[План розробки 153](#_Toc87542183)

[Приклад програмного коду 153](#_Toc87542184)

[Завдання для вдосконалення проекту 155](#_Toc87542185)

[Завдання для самостійної розробки 156](#_Toc87542186)

[Проект «Хрестики-нулики» (xo.exe) 156](#_Toc87542187)

[Детальний опис функціонування проекту 156](#_Toc87542188)

[План розробки 3](#_Toc87542189)

[Приклад програмного коду необхідних класів. 3](#_Toc87542190)

[Завдання для вдосконалення проекту 9](#_Toc87542191)

[Завдання для самостійної розробки 10](#_Toc87542192)

## ВСТУП

Даний лабораторний практикум складений відповідно до програми курсу “Об’єктно-орієнтоване програмування”, і призначений для студентів спеціальності 121 „Інженерія програмного забезпечення”

Мета практикуму — закріпити знання, отримані при вивченні теоретичної частини курсів і одержати практичні навички розробки об’єктно-орієнтованих програм. Практикум охоплює всі розділи об’єктно-орієнтованого програмування мовою С++ і включає виконання десяти лабораторних робіт. Перші чотири роботи пов'язані з базовими поняттями С++, такими як об'єкти і класи, спадкування, поліморфізм і віртуальні функції, обробка подій. Останні оботи присвячені розвинутому програмуванню на С++ і охоплюють розділи професійного програмування, такі як об’ектний аналіз, моделювання, шаблони, потокові класи і стандартна бібліотека шаблонів.

У посібнику для кожної лабораторної роботи зазначені мета й основний зміст роботи. Наведено теоретичні відомості, необхідні для проведення роботи, порядок виконання роботи і методичні вказівки. Наприкінці наведені варіанти завдань і зміст звіту по роботі.

Лабораторні роботи виконуються: у середовищі Visual Studio C++(та C#).

## Лабораторна робота №1. Класи й об'єкти в С++

**Мета.** Одержати практичні навички реалізації класів на С++.

**Основний зміст роботи.**

Написати програму, в якій створюються і руйнуються об'єкти, визначеного користувачем класу. Виконати дослідження викликів конструкторів і деструкторів.

**Короткі теоретичні зведення**

**Клас.**

Клас — фундаментальне поняття С++ і лежить в основі багатьох властивостей С++. Клас надає механізм для створення об'єктів. У класі відбиті найважливіші концепції об’єктно-орієнтованого програмування: інкапсуляція, спадкування, поліморфізм.

З погляду синтаксису клас у С++ — це структурований тип, утворений на основі вже існуючих типів.

У цьому сенсі клас є розширенням поняття структури. У найпростішому випадку клас можна визначити за допомогою конструкції:

***тип\_класу ім'я\_класу {список\_членів\_класу};***

де

*тип\_класу* – одне з службових слів **class, struct, union**;

*ім'я\_класу* – ідентифікатор;

*список\_членів\_класу* – визначення й описи типізованих даних і приналежних класу функцій.

Функції – це методи класу, що визначають операції над об'єктом.

Дані – це поля об'єкта, що утворять його структуру. Значення полів визначає стан об'єкта.

##### Приклади

struct date //дата

{int month,day,year; // поля: місяць, день, рік

void set(int,int,int); // метод - установити дату

void get(int\*,int\*,int\*); // метод- одержати дату

void next(); // метод- установить наступну дату

void print(); // метод - вивести дату

};

struct class complex // комплексне число

{double re,im;

double real(){return(re);}

double imag(){return(im);}

void set(double x,double y){re = x; im = y;}

void print(){cout<<”re = ”<<re; cout<<“im = ”<<im;}

};

Для опису об'єкта класу (екземпляра класу) використовується конструкція:

***ім'я\_класу ім'я\_об'єкта;***

date today,my\_birthday;

date \*point = &today; //покажчик на об'єкт типу date

date clim[30]; // масив об'єктів

date &name = my\_birthday; //посилання на об'єкт

В обумовлені об'єкти входять дані, які відповідають членам-даним класу. Функції — члени класу дозволяють обробляти дані конкретних об'єктів класу. Звертатися до даних об'єкта і викликати функції для об'єкта можна двома способами. По-перше, за допомогою “кваліфікованих” імен:

***ім'я\_об'єкта. ім'я\_даного***

***ім'я\_об'єкта. ім'я\_функції***

Наприклад:

complex x1,x2;

x1.re = 1.24;

x1.im = 2.3;

x2.set(5.1,1.7);

x1.print();

Другий спосіб доступу використовує покажчик на об'єкт

***покажчик\_на\_об'єкт–>ім'я\_компонента***

complex \*point = &x1; // чи point = new complex;

point –>re = 1.24;

point –>im = 2.3;

point –>print();

**Доступність компонентів класу**

У розглянутих раніше прикладах класів компоненти класів є загальнодоступними. У будь-якому місці програми, де “видно” визначення класу, можна одержати доступ до компонентів об'єкта класу. Тим самим не виконується основний принцип абстракції даних – інкапсуляція (приховання) даних всередині об'єкта. Для зміни видимості компонент у визначенні класу можна використовувати специфікатори доступу : **public, private, protected**.

Загальнодоступні (public) компоненти доступні в будь-якій частині програми. Вони можуть використовуватися будь-якою функцією як всередині даного класу, так і поза ним. Доступ ззовні здійснюється через ім'я об'єкта :

**ім'я\_об'єкта.ім'я\_члена\_класу;**

**посилання\_на\_об’єкт.ім'я\_члена\_класу;**

**покажчик\_на\_об'єкт->ім'я\_члена\_класу;**

Власні (private) компоненти локалізовані в класі і не доступні ззовні. Вони можуть використовуватися функціями-членами даного класу і функціями-“друзями” того класу, в якому вони описані.

Захищені (protected) компоненти доступні всередині класу і в похідних класах.

Змінити статус доступу до компонентів класу можна і за допомогою використання у визначенні класу ключового слова **class**. У цьому випадку усі компоненти класу за замовчуванням є власними.

**Приклад.**

class complex

{

double re, im; // private за замовчуванням

public:

double real(){return re;}

double imag(){return im;}

void set(double x,double y){re = x; im = y;}

};

**Конструктор.**

Недоліком розглянутих раніше класів є відсутність автоматичної ініціалізації створюваних об'єктів. Для кожного знову створюваного об'єкта необхідно було викликати функцію типу set (як для класу complex), або явно привласнювати значення даним об'єкта. Однак для ініціалізації об'єктів класу в його визначення можна явно включити спеціальну компонентну функцію, що називається **конструктором**. Формат визначення конструктора наступний

***ім'я\_класу(список\_форм\_параметрів){оператори\_тіла\_конструктора}***

Ім'я цієї компонентної функції за правилами мови С++ повинно збігатися з ім'ям класу. Така функція автоматично викликається при визначенні чи розміщенні в пам'яті за допомогою оператора new кожного об'єкта класу.

##### Приклад

сomplex(double re1 = 0.0,double im1 = 0.0){re = re1; im = im1;}

Конструктор виділяє пам'ять для об'єкта та ініціалізує дані- члени класу.

Конструктор має ряд особливостей:

1. Для конструктора не визначається тип значення, що повертається. Навіть тип void не припустимий.
2. Покажчик на конструктор не може бути визначений і відповідно не можна одержати адресу конструктора.
3. Конструктори не успадковуються.
4. Конструктори не можуть бути описані з ключовими словами virtual, static, const, mutuable, valatile.

Конструктор завжди існує для будь-якого класу, причому, якщо він не визначений явно, він створюється автоматично. За замовчуванням створюється конструктор без параметрів і конструктор копіювання. Якщо конструктор описаний явно, то конструктор за замовчуванням не створюється. За замовчуванням конструктори створюються загальнодоступними (public).

Параметром конструктора не може бути його власний клас, але може бути посилання на нього (T&). Без явної вказівки програміста конструктор завжди автоматично викликається при визначенні (створенні) об'єкта. У цьому випадку викликається конструктор без параметрів. Для явного виклику конструктора використовуються дві форми:

*ім'я\_класу ім'я\_об'єкта(фактичні\_параметри);*

*ім'я\_класу(фактичні\_параметри);*

Перша форма допускається тільки при не порожньому списку фактичних параметрів. Вона передбачає виклик конструктора при визначенні нового об'єкта даного класу:

complex ss(5.9,0.15);

Друга форма виклику приводить до створення об'єкта без імені:

complex ss = complex(5.9,0.15);

Існує два способи ініціалізації даних об'єкта за допомогою конструктора. Раніше ми розглядали перший спосіб, а саме передачу значень параметрів у тіло конструктора. Другий спосіб передбачає застосування списку ініціалізаторів даного класу. Цей список міститься між списком параметрів і тілом конструктора. Кожен ініціализатор списку відноситься до конкретного компонента і має вид

*ім'я\_даного(вираження)*

Приклади.

class CLASS\_A

{

int i; float e; char c;

public:

CLASS\_A(int ii,float ee,char cc) : i(8),e( i \* ee + ii ),з(сс){}

. . .

};

Клас “символьний рядок”.

#include <string.h>

#include <iostream.h>

class string

{

char \*ch; // покажчик на текстовий рядок

int len; // довжина текстового рядка

public:

// конструктори

// створює об'єкт – порожній рядок

string(int N = 80): len(0){ch = new char[N+1]; ch[0] = '\0';}

// створює об'єкт по заданому рядку

string(const char \*arch){len = strlen(arch);

ch = new char[len+1];

strcpy(ch,arch);}

// компонента-функції

// повертає посилання на довжину рядка

int& len\_str(void){return len;}

// повертає покажчик на рядок

char \*str(void){return ch;}

. . .};

Тут в класі string два конструктори – функції, що перевантажуються.

За замовчуванням створюється також конструктор копіювання виду T::T(const T&), де Т — ім'я класу. Конструктор копіювання викликається кожен раз, коли виконується копіювання об'єктів, що належать класу. Зокрема він викликається:

а)коли об'єкт передається функції за значенням;

б)при побудові тимчасового об'єкта значення функції, що повертається;

в)при використанні об'єкта для ініціалізації іншого об'єкта.

Якщо клас не містить явно визначеного конструктора копіювання, то при виникненні однієї з цих трьох ситуацій, виконується побітове копіювання об'єкта. Побітове копіювання не у всіх випадках є адекватним. Саме для таких випадків і необхідно визначити власний конструктор копіювання. Наприклад, у класі string:

string(const string& st)

{len=strlen(st.len);

ch=new char[len+1];

strcpy(ch,st.ch); }

Можна створювати масив об'єктів, однак при цьому відповідний клас повинний мати конструктор за замовчуванням(без параметрів).

Масив об'єктів може ініціюватися або автоматично конструктором за замовчуванням, або явним присвоюванням значень кожному елементу масиву.

class demo{

int x;

public:

demo(){x=0;}

demo(int i){x=i;}

};

void main(){

class demo a[20]; //виклик конструктора без параметрів(за замовчуванням)

class demo b[2]={demo(10),demo(100)};//явне присвоювання

**Деструктор.**

Динамічне виділення пам'яті для об'єкта створює необхідність звільнення цієї пам'яті при знищенні об'єкта. Наприклад, якщо об'єкт формується як локальний всередині блоку, то доцільно, щоб при виході з блоку, коли вже об'єкт перестає існувати, виділена для нього пам'ять була повернута. Бажано щоб звільнення пам'яті відбувалося автоматично. Таку можливість забезпечує спеціальний компонент класу – **деструктор** класу. Його формат:

***~ім’я\_класу( ){оператори\_тіла\_деструктора}***

Ім'я деструктора збігається з ім'ям його класу, але добавляється префіксний символ “~” (тильда).

Деструктор не має параметрів і значення, що повертається. Виклик деструктора виконується не явно (автоматично), як тільки об'єкт класу знищується.

Наприклад, при виході за область визначення або при виклику оператора delete для покажчика на об'єкт.

string \*p=new string(“рядок”);

delete p;

Якщо в класі деструктор не визначений явно, то компілятор генерує деструктор за замовчуванням, що просто звільняє пам'ять, зайняту даними об'єкта. У тих випадках, коли потрібно виконати звільнення й інші об'єкти пам'яті, наприклад область, на яку вказує ch в об'єкті string, необхідно визначити деструктор явно: ~string(){delete []ch;}

Також як і для конструктора, не може бути визначений покажчик на деструктор.

**Покажчики на компоненти-функції.**

Можна визначити покажчик на компоненти-функції.

***тип\_поверн\_значення(ім'я\_класу::\*ім'я\_покажчика\_на\_функцію)(специф\_параметрів\_функції);***

Приклад .

double(complex : :\*ptcom)(); // Визначення покажчика

ptcom = &complex : : real; // Настроювання покажчика

// Тепер для об'єкта А можна викликати його функцію

complex A(5.2,2.7);

cout<<(A.\*ptcom)();

Можна визначити також тип покажчика на функцію

typedef double&(complex::\*PF)();

а потім визначити і сам покажчик

PF ptcom=&complex::real;

**Порядок виконання роботи.**

1.Визначити клас користувача відповідно до варіанта завдання (дивись додаток).

2.Визначити в класі наступні конструктори: без параметрів, з параметрами, копіювання.

3.Визначити в класі деструктор.

4.Визначити в класі компонента-функції для перегляду і встановлення полів даних.

5.Визначити покажчик на компонент-функцію.

6.Визначити покажчик на екземпляр класу.

7.Написати демонстраційну програму, в якій створюються і руйнуються об'єкти класу користувача і кожен виклик конструктора і деструктора супроводжується видачею відповідного повідомлення (який об'єкт, який конструктор чи деструктор викликав).

8.Показати в програмі використання покажчика на об'єкт і покажчика на компонент-функцію.

**Методичні вказівки.**

1.Приклад визначення класу.

const int LNAME=25;

class STUDENT{

char name[LNAME]; // ім'я

int age; // вік

float grade; // рейтинг

public: receipt

STUDENT(); // конструктор без параметрів

STUDENT(char\*,int,float); // конструктор з параметрами

STUDENT(const STUDENT&); // конструктор копіювання

~STUDENT();

char \* GetName() ;

int GetAge() const;

float GetGrade() const;

void SetName(char\*);

void SetAge(int);

void SetGrade(float);

void Set(char\*,int,float);

void Show(); };

Більш професійне визначення поля **name** типу покажчик: char\* name. Однак у цьому випадку реалізація компонентів-функцій ускладнюється.

2.Приклад реалізації конструктора з видачею повідомлення.

STUDENT::STUDENT(char\*NAME,int AGE,float GRADE)

{

strcpy(name,NAME); age=AGE; grade=GRADE;

cout<<”\nКонструктор з параметрами викликаний для об'єкта “<<this<<endl;

}

3.Варто передбачити в програмі всі можливі способи виклику конструктора копіювання. Нагадуємо, що конструктор копіювання викликається:

а) при використанні об'єкта для ініціалізації іншого об'єкта

Приклад

STUDENT a(“Іванов”,19,50), b=a;

б) коли об'єкт передається функції за значенням

Приклад

void View(STUDENT a){a.Show;}

в) при побудові тимчасового об'єкта значення функції, що повертається

Приклад

STUDENT NoName(STUDENT & student)

{STUDENT temp(student);

temp.SetName(“NoName”);

return temp;}

STUDENT c=NoName(a);

4.У програмі необхідно передбачити розміщення об'єктів як в статичній, так і в динамічній пам'яті, а також створення масивів об'єктів.

Приклади.

а)Масив студентів розміщається в статичній пам'яті

STUDENT gruppa[3];

gruppa[0].Set(“Іванов”,19,50);

і т.д.

або

STUDENT gruppa[3]={STUDENT(“Іванов”,19,50),

STUDENT(“Петрова”,18,25.5),

STUDENT(“Сидоров”,18,45.5)};

б)Масив студентів розміщається в динамічній пам'яті

STUDENT \*p;

p=new STUDENT[3];

p-> Set(“Іванов”,19,50);

і т.д.

5.Приклад використання покажчика на компонентну функцію

void (STUDENT::\*pf)();

pf=&STUDENT::Show;

(p[1].\*pf)();

6.Програма використовує три файли

1. заголовний h-файл із визначенням класу
2. cpp-файл із реалізацією класу
3. сpp-файл демонстраційної програми

Для запобігання багаторазового включення файлу-заголовка варто використовувати директиви препроцесора

#ifndef STUDENTH

#define STUDENTH

// модуль STUDENT.H

...

#endif

**Зміст звіту.**

1.Назва роботи, мета, прізвище, ім'я, по батькові студента, дата виконання.

2.Постановка задачі. Варто дати конкретну постановку, тобто вказати який клас повинен бути реалізований, які повинні бути в ньому конструктори, компоненти-функції і т.д.

3.Визначення класу користувача з коментарями.

4.Реалізація конструкторів і деструктора.

5.Фрагмент програми, що показує використання покажчика на об'єкт і покажчика на функцію з поясненням.

6.Лістінг основної програми, де повинно бути зазначене в якому місці і який конструктор чи деструктор викликаються.

**Приклади виконаних завдань**

**Person.h**

#pragma once

class Person

{

public:

char \*name;

int age;

bool pol;

public:

Person();

Person(char \*name,int age, bool pol);

Person(const Person&);

~Person(void);

char\* getName();

int getAge();

bool getPol();

void setPerson(char \*n, int old, bool pol1);

void setName(char \*n);

void setAge(int old);

void setPol(bool pol1);

void printPerson();

};

**Person.cpp**

#include "StdAfx.h"

#include "Person.h"

#include <iostream>

using namespace std;

Person::Person():name(""),age(0),pol(false)

{

cout << "Vuklukano konstruktor bez parametriv " << this << endl;

}

Person::Person(char \*n, int old, bool pol1):name(n),age(old),pol(pol1)

{

cout << "Vuklukano konstruktor z parametrom" << this << endl;

}

Person::Person(const Person&)

{

cout << "Vuklukano konstruktor kopiruvania" << this << endl;

};

Person::~Person(void)

{

cout << "Vuklukano destruktor " << this << endl;

};

char\* Person::getName()

{

return name;

};

int Person::getAge()

{

return age;

};

bool Person::getPol()

{

return pol;

};

void Person::setName(char\* n)

{

name=n;

};

void Person::setAge(int old)

{

age=old;

};

void Person::setPol(bool pol1)

{

pol=pol1;

};

void Person::printPerson()

{

char\* pol1;

if(pol) pol1="men"; else pol1="women";

cout << "Name:"<< name << "\t age:" << age << "\t pol:" << pol1 << endl;

}

void Person::setPerson(char \*n, int old, bool pol1)

{

name=n;

age=old;

pol=pol1;

char\* pol2;

if(pol) pol2="men"; else pol2="women";

cout << "Name:"<< name << "\t age:" << age << "\t pol:" << pol2 << endl;

}

**Project\_1.cpp**

// Project\_1.cpp: определяет точку входа для консольного приложения.

//

#include "stdafx.h"

#include "Person.h"

#include <iostream>

#include <conio.h>

using namespace std;

void print(Person ob)

{

cout << "Drue z funkcii: ";

ob.printPerson();

}

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

{

Person chel;

Person \*chel1;

chel1= new Person;

Person Spis[2];

chel.setPerson("Andriy",17,true);

chel1->setPerson("Marina", 17,true);

chel1->printPerson();

chel.printPerson();

chel1->~Person();

chel.~Person();

Spis[0].~Person();

Spis[1].~Person();

Spis[2].~Person();

\_getch();

return 0;

}

**Завдання 2 Варіант 10**

**DataTimes.h**

char name;

int kil;

float coast;

unsigned short int dd;

unsigned short int mm;

unsigned int yyyy;

public:

DataTimes();

DataTimes(unsigned short int d,unsigned short int m,unsigned int y);

~DataTimes();

void setAll(unsigned short int d,unsigned short int m,unsigned int y);

int getYear();

int getMounth();

int getDay();

DataTimes Input();

void Print();

DataTimes NextDay(DataTimes ob);

DataTimes PrevDay(DataTimes ob);

int CountDay();

};

**DataTimes.cpp**

#include "StdAfx.h"

#include "DataTimes.h"

#include <iostream>

using namespace std;

DataTimes::DataTimes():dd(1),mm(1),yyyy(2009){};

DataTimes::DataTimes(unsigned short int d = 1,unsigned short int m = 1,unsigned int y = 2009)

{

}

DataTimes::~DataTimes(void)

{

}

void DataTimes::setAll(unsigned short int d,unsigned short int m,unsigned int y)

{

dd=d;

mm=m;

yyyy=y;

}

int DataTimes::getYear()

{

return yyyy;

}

int DataTimes::getMounth()

{

return mm;

}

int DataTimes::getDay()

{

return dd;

}

DataTimes DataTimes::Input()

{

setlocale(LC\_ALL,"rus");

cout << "Ââåäèòå ãîä:";

cin >> yyyy;

cout << "Ââåäèòå ìåñÿö:";

cin >> mm;

cout << "Ââåäèòå äåíü:";

cin >> dd;

setlocale(LC\_ALL,"");

return DataTimes(dd,mm,yyyy);

}

DataTimes DataTimes::NextDay(DataTimes ob)

{

this->dd=ob.dd;

this->mm=ob.mm;

this->yyyy=ob.yyyy;

this->dd++;

if((mm==1)||(mm==3)||(mm==5)||(mm==7)||(mm==8)||(mm==10)||(mm==12))

{

if(dd>31) {dd=1;mm++;}

} else

{

if((mm==4)||(mm==6)||(mm==9)||(mm==11))

{

if(dd>30) {dd=1;mm++;}

} else

{

if(((yyyy%4==0) && (yyyy%100!=0))||((yyyy%4==0) && (yyyy%100!=0) && (yyyy%400==0)))

{

if(dd>29)

{

dd=1;mm++;

}

} else

{

if(dd>28)

{

dd=1;mm++;

}

}

}

}

if(mm>12) {

mm=1;

yyyy++;

}

return DataTimes(dd,mm,yyyy);

}

DataTimes DataTimes::PrevDay(DataTimes ob)

{

this->dd=ob.dd;

this->mm=ob.mm;

this->yyyy=ob.yyyy;

this->dd--;

if((mm==1)||(mm==3)||(mm==5)||(mm==7)||(mm==8)||(mm==10)||(mm==12))

{

if(dd<0) {dd=30;mm--;

if(((yyyy%4==0) && (yyyy%100!=0)&& (mm==2))||((yyyy%4==0) && (yyyy%100!=0) && (yyyy%400==0) && (mm==2)))

{

if(dd<1)

{

dd=29;

}

} else

{

if(dd<1)

{

dd=28;

}

}}

} else

{

if((mm==2)||(mm==4)||(mm==6)||(mm==9)||(mm==11))

{

if(dd<0) {dd=31;mm--;}

}

}

if((dd<1)&(mm==1)) {

dd=31;

mm=12;

yyyy--;

}

return DataTimes(dd,mm,yyyy);

}

int DataTimes::CountDay()

{

switch (mm)

{

case 1:case 3:case 5:case 7:case 8:case 10:case 12: { return 31-dd;break;};

case 4:case 6:case 9:case 11: {return 30-dd;break;}

case 2:

{

if(((yyyy%4==0) && (yyyy%100!=0)&& (mm==2))||((yyyy%4==0) && (yyyy%100!=0) && (yyyy%400==0) && (mm==2)))

return 29-dd; else return 28-dd;

break;

}

default: break;

}

}

void DataTimes::Print() {

printf("%02d.%02d.%04d\n",dd,mm,yyyy);

}

**Project\_2.cpp**

// Project\_2.cpp: определяет точку входа для консольного приложения.

//

#include "stdafx.h"

#include "DataTimes.h"

#include <iostream>

using namespace std;

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

{

int n;

setlocale(LC\_ALL,"");

cout<<"Введите количество обектов:";

cin>>n;

DataTimes ob[100];

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

{

cout<<"Ведите обект №"<<i+1<<"\n";

ob[i].Input();

cout<<"Количество дней до конца месяца="<<ob[i].CountDay()<<"\n";

DataTimes tmp;

tmp.PrevDay(ob[i]);

cout<<"Дата предидущего дня:";

tmp.Print();

tmp.NextDay(ob[i]);

cout<<"Дата следуещого дня:";

tmp.Print();

}

setlocale(LC\_ALL,"");

system("pause");

return 0;

}

**Додаток. Варіанти завдань**

***Опису членів-даних класів користувача***

|  |  |  |
| --- | --- | --- |
| **1.СТУДЕНТ** | **2.СЛУЖБОВЕЦЬ** | **3.КАДРИ** |
| ім'я – char\* | ім'я – char\* | ім'я – char\* |
| курс – int | вік – int | номер цеху-int |
| стать - int(bool) | робітник стаж – int | розряд- int |
| **4.ВИРІБ** | **5.БІБЛІОТЕКА** | **6.ІСПИТ** |
| ім'я- char\* | ім'я- char\* | ім'я студента-char\* |
| шифр – char\* | автор – char\* | дата - int |
| кількість- int | вартість- float | оцінка- int |
| **7.АДРЕСА** | **8.ТОВАР** | **9.КВИТАНЦІЯ** |
| ім'я- char\* | ім'я- char\* | номер- int |
| вулиця- char\* | кількість- int | дата- int |
| номер будинку- int | вартість- float | сума- float |
| **10.ЦЕХ** | **11.ПЕРСОНА** | **12.АВТОМОБІЛЬ** |
| ім'я- char\* | ім'я- char\* | марка-char\* |
| начальник – char\* | вік – int | потужність - int |
| кількість працюючих- int | стать - int(bool) | ціна- float |
| **13.КРАЇНА** | **14.ТВАРИНА** | **15.КОРАБЕЛЬ** |
| ім'я- char\* | ім'я- char\* | ім'я-char\* |
| форма правління – char\* | клас-char\* | водотоннажність - int |
| площа – float | середня вага – int | тип- char\* |

**Завдання 2**

**Постановка задачі :**

Розробити програмну реалізацію класу з тестовою програмою для перевірки функціонування всіх методів класу.

**Варіанти завдань для самостійної роботи**

1. Створити клас Point, розробивши наступні елементи класу:
   * Поля:
     + int x, у;
   * Конструктори, що дозволяють створити екземпляр класу:
     + з нульовими координатами;
     + із заданими координатами.
   * Методи, що дозволяють:
     + вивести координати точки на екран;
     + розрахувати відстань від початку координат до точки;
     + перемістити точку на площині на вектор (а, b).
   * Властивості:
     + одержати-встановити координати точки (доступне для читань і запису);
     + дозволяючі помножити координати точки на скаляр (доступне тільки для запису).
2. Створити клас Triangle, розробивши наступні елементи класу:
   * Поля:
     + int а, b, c;
   * Конструктор, що дозволяє створити екземпляр класу із заданими довжинами сторін.
   * Методи, що дозволяють:
     + вивести довжини сторін трикутника на екран;
     + обчислити периметр трикутника;
     + обчислити площу трикутника.
   * Властивості:
     + які надають можливість одержати-встановити довжини сторін трикутника (доступне для читання і запису);
     + які надають можливість встановити, чи існує трикутник з даними довжинами сторін (доступне тільки для читання).
3. Створити клас Rectangle, розробивши наступні елементи класу:
   * Поля:
     + int а, b;
   * Конструктор, що дозволяє створити екземпляр класу із заданими довжинами сторін.
   * Методи, що дозволяють:
     + вивести довжини сторін прямокутника на екран;
     + обчислити периметр прямокутника;
     + обчислити площу прямокутника.
   * Властивості:
     + одержати-встановити довжини сторін прямокутника (доступне для читання і запису);
     + які надають можливість встановити, чи є даний прямокутник квадратом (доступне тільки для читання).
4. Створити клас Money, розробивши наступні елементи класу:
   * Поля:
     + int first;//номинал купюри
     + int second; //кількість купюр
   * Конструктор, що дозволяє створити екземпляр класу із заданими значенням полів.
   * Методи, що дозволяють:
     + вивести номінал і кількість купюр;
     + визначити, чи вистачить грошових коштів на покупку товару на суму N гривнів.
     + визначити, скільки шт товару вартості n гривнів можна купити на наявні грошові кошти.
   * Властивості:
     + які надають можливість одержати-встановити значення полів (доступне для читання і запису);
     + які надають можливість расчитатать суму грошей (доступне тільки для читання).
5. Створити клас для роботи з одновимірним масивом цілих чисел. Розробити наступні елементи класу:
   * Поля:
     + int [] IntArray;
     + int n.
   * Конструктор, що дозволяє створити масив розмірності n.
   * Методи, що дозволяють:
     + ввести елементи масиву з клавіатури;
     + вивести елементи масиву на екран;
     + відсортувати елементи масиву в порядку зростання.
   * Властивості:
     + повертаюче розмірність масиву (доступне тільки для читання);
     + які надають можливість домножить всі елементи масиву на скаляр (доступне тільки для запису).
6. Створити клас для роботи з двовимірним масивом цілих чисел. Розробити наступні елементи класу:
   * Поля:
     + int [,] IntArray;
     + int n.
   * Конструктор, що дозволяє створити масив розмірності nЧn.
   * Методи, що дозволяють:
     + ввести елементи масиву з клавіатури;
     + вивести елементи масиву на екран;
     + обчислити суму элеметов i-того стовпця.
   * Властивості:
     + які надають можливість обчислити кількість нульових елементів в масиві (доступне тільки для читання);
     + які надають можливість встановити значення всіх елементи головної діагоналі масиву рівне скаляру (доступне тільки для запису).
7. Створити клас для роботи з двовимірним масивом дійсних чисел. Розробити наступні елементи класу:
   * Поля:
     + double [][] DoubelArray;
     + int n, m.
   * Конструктор, що дозволяє створити масив розмірності nЧm.
   * Методи, що дозволяють:
     + ввести елементи масиву з клавіатури;
     + вивести елементи масиву на екран;
     + відсортувати елементи кожного рядка масиву в порядку убування.
   * Властивості:
     + повертаюча загальна кількість елементів в масиві (доступне тільки для читання);
     + які надають можливість збільшити значення всіх елементів масиву на скаляр (доступне тільки для запису).
8. Створити клас для роботи з рядками. Розробити наступні елементи класу:
   * Поля:
     + Line;
     + int n.
   * Конструктор, що дозволяє створити рядок з n символів.
   * Методи, що дозволяють:
     + підрахувати кількість пропусків в рядку;
     + замінити в рядку всі прописні символи на рядкові;
     + видалити з рядка всі розділові знаки.
   * Властивості:
     + Що повертають загальна кількість елементів в рядку (доступне тільки для читання);
     + які надають можливість встановити значення поля, відповідно до введеного значення рядка з клавіатури, а також отримати значення даного поля (доступно для читання і запису)
9. Створити клас Cіrcle. Розробити наступні елементи класу:
   * Поля:
     + Double R;
   * Методи, що дозволяють:
     + Визначити довжину кола ;
     + Площу круга;
     + Об’эм сфери
     + вивести на екран всі даны та результати;
   * Властивості:
     + які надають можливість встановити або отримати R поле класу (доступно для читання і запису)
     + які надають можливість встановити або отримати нове значення - діаметр, що зберігається у відповідному полі класу (доступно для читання і запису)
10. Створити клас для роботи з датою. Розробити наступні елементи класу:
    * Поле DataTime data.
    * Конструктори, що дозволяють встановити:
      + задану дату
      + дату 1.01.2009
    * Методи, що дозволяють:
      + обчислити дату попереднього дня;
      + обчислити дату наступного дня;
      + визначити скільки днів залишилося до кінця місяця.
    * Властивості:
      + які надають можливість встановити або отримати значення поля класу (доступно для читання і запису)
      + які надають можливість визначити рік высокосным (доступно тільки для читання)

## Лабораторна робота №2. Перевантаження операцій.

**Мета.** Одержати практичні навички створення абстрактних типів даних і перевантаження операцій у мові С++.

**Основний зміст роботи.**

Визначити і реалізувати клас — абстрактний тип даних. Визначити і реалізувати операції над даними цього класу. Написати і виконати програму повного тестування цього класу.

**Короткі теоретичні відомості.**

**1.Абстрактний тип даних (АТД).**

Тип даних, обумовлений тільки через операції, що можуть виконуватися над відповідними об'єктами без відношення до способу представлення цих об'єктів.

АТД містить у собі абстракцію як через параметризацію, так і через специфікацію. **Абстракція через параметризацію** може бути здійснена так само як і для процедур (функцій) використанням параметрів там, де це має сенс. **Абстракція через специфікацію** досягається за рахунок того, що операції представляються як частина типу.

Для реалізації АТД необхідно, по-перше, вибрати представлення пам'яті для об'єктів і, по-друге, реалізувати операції в термінах обраного представлення.

Прикладом абстрактного типу даних є клас у мові С++.

**2.Перевантаження операцій.**

Можливість використовувати знаки стандартних операцій для запису виразів як для вбудованих, так і для АТД.

У мові С++ для перевантаження операцій використовується ключове слово **operator**, за допомогою якого визначається спеціальна операція-функція (operator function).

Формат операції-функції:

тип\_поверн\_значення operator знак\_операції(специф\_параметрів)

{оператори\_тіла\_функції}

Перевантаження унарних операцій

1. Будь-яка унарна операція Å може бути визначена двома способами: або як компонентна функція без параметрів, або як глобальна функція з одним параметром. У першому випадку вираз Å Z означає виклик Z.operatorÅ (), у другому- виклик operatorÅ (Z).
2. Унарні операції, що перевантажуються в рамках визначеного класу, можуть перевантажуватися тільки через нестатичну компонентну функцію без параметрів. Викликаний об'єкт класу автоматично сприймається як операнд.
3. Унарні операції, що перевантажуються поза областю класу( як глобальні функції), повинні мати один параметр типу класу. Переданий через цей параметр об'єкт сприймається як операнд.

Синтаксис:

а) у першому випадку (опис в області класу):

тип\_поверн\_значення operator знак\_операції

б) у другому випадку (опис поза областю класу):

тип\_поверн\_значення operator знак\_операції(ідентифікатор\_типу)

Приклади.

|  |  |
| --- | --- |
| 1)class person | 2)class person |
| { int age; | { int age; |
| ... | ... |
| public: | public: |
| … | ... |
| void operator++(){ ++age;} | friend void operator++(person&); |
| }; | }; |
| void main() | void person::operator++(person& ob) |
| {class person jon; | {++ob.age;} |
| ++jon;} | void main() |
|  | {class person jon; |
|  | ++jon;} |

Перевантаження бінарних операцій

1. Будь-яка бінарна операція Å може бути визначена двома способами: або як компонентна функція з одним параметром, або як глобальна (можливо дружня) функція з двома параметрами. У першому випадку **xÅy** означає виклик **x.operatorÅ(y)**, у другому – виклик **operatorÅ(x,y).**
2. Операції, що перевантажуються всередині класу, можуть перевантажуватися тільки нестатичними компонентними функціями з параметрами. Викликуваний об'єкт класу автоматично сприймається в якості першого операнда.
3. Операції, що перевантажуються поза областю класу, повинні мати два операнди, один із яких повинний мати тип класу.

Приклади.

1)class person{...};

class adresbook

{

// містить у якості компонентних даних множину об'єктів типу

//person, що представляються як динамічний масив, чи список дерево

…

public:

person& operator[](int); //доступ до i-ому об'єкта

};

person& adresbook : : operator[](int i){. . .}

void main()

{class adresbook persons;

class person record;

…

record = persons[3];

}

2) class person{...};

class adresbook

{ // містить у якості компонентних даних множину об'єктів типу //person, що представляються як динамічний масив, чи список дерево

…

public:

friend person& operator[](const adresbook&,int); //доступ до i-ого об'єкта

};

person& operator[](const adresbook& ob ,int i){. . .}

void main()

{class adresbook persons;

class person record;

…

record = persons[3];

}

Перевантаження операції присвоювання

Операція відрізняється трьома особливостями:

1. операція не успадковується;
2. операція визначена за замовчуванням для кожного класу як операцію порозрядного копіювання об'єкта, що стоїть праворуч від знака операції, в об'єкт, що стоїть ліворуч.
3. операція може перевантажуватися тільки в області визначення класу. Це гарантує, що першим операндом завжди буде ліводопустимий вираз.

Формат перевантаженої операції присвоювання:

**ім'я\_класу***& operator=*( **ім'я\_класу***&*);

Відзначимо дві важливі особливості функції  *operanor=.* По-перше, в ній використовується посилання-параметр-посилання. Це необхідно для запобігання створення копії об'єкта, переданого через параметр за значенням. У випадки створення копії, вона видаляється викликом деструктора при завершенні роботи функції. Але деструктор звільняє розподілену пам'ять, ще необхідну об'єкту, що є аргументом. Посилання-параметр-посилання допомагає вирішити цю проблему.

По-друге, функція operator=() повертає не об'єкт, а посилання на нього. Зміст цього той же, що і при використанні посилання-параметр-посилання. Функція повертає тимчасовий об'єкт, що видаляється після завершення її роботи. Це означає, що для тимчасової змінної буде викликаний деструктор, що звільняє розподілену пам'ять. Але вона необхідна для присвоювання значення об'єкту. Тому, щоб уникнути створення тимчасового об'єкта, як такого, що повертає значення, використовується посилання.

Традиційно інтерфейс і реалізацію класу розміщують у різних файлах, як це

було з оголошеннями і визначеннями об'єктів. Інтерфейс записують у заголовні

файли.

Ось приклад заголовного файлу для класу точок

//Point.h

#ifndef POINT\_H

#define POINT\_H

class Point

{

// Атрибути

private:

double \_x;

double \_y;

// Методи

public:

Point (double a=0, double b=0);

Point operator+(Point);

bool operator==(Point);

double modulus ();

double phi ();

}

#endif

Ось файл реалізації

//Point.cpp

#include ”Point.h”

#include <cmath>

Point :: Point(double a, double b)

{

\_x=a; \_y=b;

}

Point Point::operator+(Point v)

{

Point w;

w.\_x = \_x + v.\_x;

w.\_y = \_y + v.\_y;

return w;

}

bool operator== (Point v)

{

return (\_x == v.\_x) && (\_y == v.\_y);

}

double modulus ()

{

return sgrt(\_x\*\_x+\_y\*\_y);

}

double Point::phi ()

{

return atan(\_y/\_x);

}

//Кінець Point.cpp

**Приклад виконання завдань**

**Варіант 10.**

**Datatimes.h**

#pragma once

#include <string>

class DataTimes

{

public:

unsigned short int dd;

unsigned short int mm;

unsigned int yyyy;

public:

DataTimes();

DataTimes(unsigned short int d,unsigned short int m,unsigned int y);

~DataTimes();

void setAll(unsigned short int d,unsigned short int m,unsigned int y);

int getYear();

int getMounth();

int getDay();

DataTimes Input();

void Print();

DataTimes NextDay(DataTimes ob);

DataTimes PrevDay(DataTimes ob);

int CountDay();

bool operator&(DataTimes ob1);

bool operator!();

bool logic();

string st1();

};

**Datatimes.cpp**

#include "StdAfx.h"

#include "DataTimes.h"

#include <iostream>

#include <string>

using namespace std;

DataTimes::DataTimes():dd(1),mm(1),yyyy(2009){};

DataTimes::DataTimes(unsigned short int d = 1,unsigned short int m = 1,unsigned int y = 2009)

{

}

DataTimes::~DataTimes(void)

{

}

void DataTimes::setAll(unsigned short int d,unsigned short int m,unsigned int y)

{

dd=d;

mm=m;

yyyy=y;

}

int DataTimes::getYear()

{

return yyyy;

}

int DataTimes::getMounth()

{

return mm;

}

int DataTimes::getDay()

{

return dd;

}

DataTimes DataTimes::Input()

{

setlocale(LC\_ALL,"rus");

cout << "Введите год:";

cin >> yyyy;

cout << "Введите месяц:";

cin >> mm;

cout << "Введите день:";

cin >> dd;

setlocale(LC\_ALL,"");

return DataTimes(dd,mm,yyyy);

}

DataTimes DataTimes::NextDay(DataTimes ob)

{

this->dd=ob.dd;

this->mm=ob.mm;

this->yyyy=ob.yyyy;

this->dd++;

if((mm==1)||(mm==3)||(mm==5)||(mm==7)||(mm==8)||(mm==10)||(mm==12))

{

if(dd>31) {dd=1;mm++;}

} else

{

if((mm==4)||(mm==6)||(mm==9)||(mm==11))

{

if(dd>30) {dd=1;mm++;}

} else

{

if(((yyyy%4==0) && (yyyy%100!=0))||((yyyy%4==0) && (yyyy%100!=0) && (yyyy%400==0)))

{

if(dd>29)

{

dd=1;mm++;

}

} else

{

if(dd>28)

{

dd=1;mm++;

}

}

}

}

if(mm>12) {

mm=1;

yyyy++;

}

return DataTimes(dd,mm,yyyy);

}

DataTimes DataTimes::PrevDay(DataTimes ob)

{

this->dd=ob.dd;

this->mm=ob.mm;

this->yyyy=ob.yyyy;

this->dd--;

if((mm==1)||(mm==3)||(mm==5)||(mm==7)||(mm==8)||(mm==10)||(mm==12))

{

if(dd<1) {dd=30;mm--;

if(((yyyy%4==0) && (yyyy%100!=0)&& (mm==2))||((yyyy%4==0) && (yyyy%100!=0) && (yyyy%400==0) && (mm==2)))

{

if(dd<1)

{

dd=29;

}

} else

{

if(dd<1)

{

dd=28;

}

}}

} else

{

if((mm==2)||(mm==4)||(mm==6)||(mm==9)||(mm==11))

{

if(dd<1) {dd=31;mm--;}

}

}

if((mm==0)) {

mm=12;

yyyy--;

}

return DataTimes(dd,mm,yyyy);

}

int DataTimes::CountDay()

{

switch (mm)

{

case 1:case 3:case 5:case 7:case 8:case 10:case 12: { return 31-dd;break;};

case 4:case 6:case 9:case 11: {return 30-dd;break;}

case 2:

{

if(((yyyy%4==0) && (yyyy%100!=0)&& (mm==2))||((yyyy%4==0) && (yyyy%100!=0) && (yyyy%400==0) && (mm==2)))

return 29-dd; else return 28-dd;

break;

}

default: return 0; break;

}

}

void DataTimes::Print() {

printf("%02d.%02d.%04d\n",dd,mm,yyyy);

}

bool DataTimes::operator&(DataTimes ob) {

if((ob.dd==dd)&&(ob.mm==mm)&&(ob.yyyy==yyyy)) return 1; else return 0;

}

bool DataTimes::operator!(){

if(dd<28) return true; else

{

if((mm==1)||(mm==3)||(mm==5)||(mm==7)||(mm==8)||(mm==10)||(mm==12)) {

if(dd==31) return false; else return true;

} else

{

if((mm==4)||(mm==6)||(mm==9)||(mm==11))

{

if(dd==30) return false; else return true;

} else

{

if(((yyyy%4==0) && (yyyy%100!=0)&& (mm==2))||((yyyy%4==0) && (yyyy%100!=0) && (yyyy%400==0) && (mm==2)))

{

if(dd==29) return false; else return true;

} else if(dd==28) return false; else return true;

}

}

}

}

bool DataTimes::logic() {

if(dd==1||mm==1) return true;

else return false;

}

**Project\_1.cpp**

// Project\_1.cpp: определяет точку входа для консольного приложения.

//

#include "stdafx.h"

#include "DataTimes.h"

#include <iostream>

using namespace std;

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

{

int n;

setlocale(LC\_ALL,"");

cout<<"Введите количество обектов:";

cin>>n;

DataTimes ob[100];

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

{

cout<<"Ведите обект №"<<i+1<<"\n";

ob[i].Input();

cout<<"Количество дней до конца месяца="<<ob[i].CountDay()<<"\n";

DataTimes tmp;

tmp.PrevDay(ob[i]);

cout<<"Дата предидущего дня:";

tmp.Print();

tmp.NextDay(ob[i]);

cout<<"Дата следуещого дня:";

tmp.Print();

}

setlocale(LC\_ALL,"");

bool flag=ob[0]&ob[1];

cout<<"Они равны:"<<flag<<"\n";

flag=!ob[2];

cout<<"Дате не является последним днем месяца"<<flag<<"\n";

cout<<"Датa #1 начало года "<<ob[0].logic()<<"\n";

system("pause");

return 0;

}

**Варіанти завдань для виконання роботи.**

**Постановка задачі: Розширити функціональність класу, реалізовану в попередній роботі, згідно тогож варіанту.**

1. В клас Point додати:
   * Перевантаження:
     + операції ++ (--): одночасно збільшує (зменшує) значення полів х і у на 1;
     + констант true і false: звернення до екземпляра класу дає значення true, якщо значення полів x і у співпадає, інакше false;
     + операції бінарний +: одночасно додає до полів х і у значення скаляра;
     + перетворення типу Point в string (і навпаки).
2. В клас Triangle додати:
   * Перевантаження:
     + операції ++ (--): одночасно збільшує (зменшує) значення полів а, b і з на 1;
     + констант true і false: звернення до екземпляра класу дає значення true, якщо трикутник із заданими довжинами сторін існує, інакше false;
     + операції \*: одночасно домножает поля а, b і c на скаляр;
     + перетворення типу Triangle в string (і навпаки).
3. В клас Rectangle додати:
   * Перевантаження:
     + операції ++ (--): одночасно збільшує (зменшує) значення полів а і b;
     + констант true і false: звернення до екземпляра класу дає значення true, якщо прямокутник із заданими довжинами сторін є квадратом, інакше false;
     + операції \*: одночасно домножает поля а і b на скаляр;
     + перетворення типу Rectangle в string (і навпаки).
4. В клас Money додати:
   * Перевантаження:
     + операції ++ (--): одночасно збільшує (зменшує) значення полів first і second;
     + операції !: повертає значення true, якщо поле second не нульове, інакше false;
     + операції бінарний +: додає до значення поля second значення скаляра;
     + перетворення типу Money в string (і навпаки).
5. Додати в клас для роботи з одновимірним масивом цілих чисел:
   * Перевантаження:
     + операції ++ (--): одночасно збільшує (зменшує) значення всіх елементів масиву на 1;
     + операції !: повертає значення true, якщо елементи масиву не впорядковані за збільшенням, інакше false;
     + операції бінарний \*: домножить всі елементи масиву на скаляр;
     + перетворення класу масив в тип string (і навпаки).
6. Додати в клас для роботи з двовимірним масивом цілих чисел:
   * Перевантаження:
     + операції ++ (--): одночасно збільшує (зменшує) значення всіх елементів масиву на 1;
     + констант true і false: звернення до екземпляра класу дає значення true, якщо двовимірний масив є квадратним;
     + операції бінарний +: дозволяючої скласти два масививдповідного розміру і типу;
     + перетворення класу масив в тип string (і навпаки).
7. Додати в клас для роботи з двовимірним масивом дійсних чисел:
   * Перевантаження:
     + операції ++ (--): одночасно збільшує (зменшує) значення всіх елементів масиву на 1;
     + констант true і false: звернення до екземпляра класу дає значення true, якщо кожний рядок масиву упорядоченна за збільшенням, інакше false.
     + операції \*: дозволяючої помножити два масиви відповідних размерностей.
     + перетворення класу масив в східчастий масив (і навпаки).
8. Додати в клас для роботи з рядками:
   * Перевантаження:
     + операції унарного + (-): перетворюючої рядок до рядкових (прописним) символів;
     + констант true і false: звернення до екземпляра класу дає значення true, якщо рядок не порожній, інакше false.
     + операції &: повертає значення true, якщо рядкові поля двох об'єктів посимвольний рівні (без урахування регістра), інакше false;
     + перетворення класса-строка в тип string (і навпаки).
9. Додати в клас для роботи :
   * Перевантаження:
     + констант true і false: звернення до екземпляра класу дає значення true, якщо поле R додатне, інакше false;
     + операції бінарного +: , яка збільшує значення поля R на 2;.
     + перетворення класу Circle в тип string (і навпаки).
10. Додати в клас для роботи з датою:
    * Перевантаження:
      + операції !: повертає значення true, якщо встановлена дата не є останнім днем місяця, інакше false;
      + констант true і false: звернення до екземпляра класу дає значення true, якщо встановлена дата є початком року, інакше false;
      + операції &: повертає значення true, якщо поля двох об'єктів рівні, інакше false;
      + перетворення класу DataTime в тип string (і навпаки).

## 

## Лабораторна робота №3. Успадкування і віртуальні функції

**Мета.** Одержати практичні навички створення ієрархії класів і використання статичних компонентів класу.

**Основний зміст роботи.**

Написати програму, в якій створюється ієрархія класів. Включити поліморфні об'єкти в зв'язаний список, використовуючи статичні компоненти класу. Показати використання віртуальних функцій.

**Короткі теоретичні відомості**

Тема: Наслідування. Доступ до членів класу.  
План  
1. Управління доступом до членів базового класу  
2. Наслідування і захищені члени  
3. Захищене наслідування  
4. Множинне наслідування  
Наслідування — один з наріжних каменів об'єктно-орієнтованого програмування, оскільки воно дозволяє створювати ієрархічні класифікації Використовуючи Наслідування, можна створювати загальні класи, що визначають властивості, характерні для всієї сукупності споріднених класів. Ці класи можуть успадковувати властивості один у одного, додаючи до них свої власні унікальні характеристики.  
Згідно стандартної термінології мови C++ клас, лежачий в основі ієрархії, називається базовим (base class), а клас, що успадковує властивості базового класу, — похідним (derived class). Похідні класи, у свою чергу, можуть бути базовими по відношенню до інших класів.  
У мові C++ передбачений могутній і гнучкий механізм наслідування.   
Управління доступом до членів базового класу  
При наслідуванні члени базового класу стають членами похідного класу. Як правило, для наслідування використовується наступна синтаксична конструкція.  
с1ass імя-похідного-класу: рівень\_доступу імя-базового-класу   
{   
// тіло класу   
}  
Параметр рівень\_доступу визначає статус членів базового класу в похідному класі. Як цей параметр використовуються специфікатори public, private або protected. Якщо рівень доступу не вказаний, то для похідного класу за умовчанням використовується специфікатор private, а для похідної структури - public. Розглянемо варіанти, що виникають в цих ситуаціях. (Специфікатор protected буде описаний в наступному розділі.)  
Якщо рівень доступу до членів базового класу задається специфікатором publiс то всі відкриті і захищені члени базового класу стають відкритими і захищеними членами похідного класу. При цьому закриті члени базового класу не міняють свого статусу і залишаються недоступними членам похідного. Як демонструє наступна програма, об'єкти класу derived можуть безпосередньо посилатися на відкриті члени класу base.  
#include <iostream>  
using namespace std;  
class base {  
int i, j;   
public:  
void set(int а, int b) { i=a; j=b; }  
void show() { cout « i << " " « j « "\n"; }   
};  
class derived : public base {  
int k;   
public:  
derived(int x) { k=x; }  
void showkO { cout « k « "\n"; }   
};  
int main() {  
derived ob (3) ;  
fcb.set(l, 2); // Звернення до члена класу base   
ob.showO; // Звернення до члена класу base  
job.showkO; // Звернення до члена класу derived  
return 0 ;  
}  
Якщо властивості базового класу успадковуються за допомогою специфікатора доступу private, всі відкриті і захищені члени базового класу стають закритими членами похідного класу. Наприклад, наступна програма навіть не буде скомпільована, оскільки обидві функції set() і show() тепер є закритими членами класу derived.  
Ця програма не буде скомпільована.   
#include <iostream>   
using namespace std;  
class base {   
int i, j ;   
public:  
void set (int а, int b) { i=a; j=b; }   
void show() { cout « i « " " « j « "\n";}  
};  
  
Відкриті члени класу base є закритими членами класу derived.   
class derived : private base {  
int k;  
public:  
derived(int x) { k=x; }  
};  
void main() {  
derived ob (3) ;  
ob.set (1, pb.show());   
// Помилка, доступ до функції set() заборонений.   
ob.show();   
// Помилка, доступ до функції show() заборонений.  
}  
При закритому наслідуванні всі відкриті і захищені члени базового класу стають закритими членами похідного класу. Це означає, що вони залишаються доступними членам похідного класу, але недоступні решті елементів програми, що не є членами базового або похідного класів.  
  
Наслідування і захищені члени  
Специфікатор protected підвищує гнучкість механізму наслідування. Якщо член класу оголошений захищеним (protected), то поза класом він недоступний. З цієї точки зору захищений член класу нічим не відрізняється від закритого. Єдине виключення з цього правила стосується наслідування. У цій ситуації захищений член класу істотно відрізняється від закритого.  
Як вказувалося в попередньому розділі, закритий член базового класу не доступний іншим елементам програми, включаючи похідний клас. Проте захищені члени базового класу поводяться інакше. При відкритому наслідуванні захищені члени базового класу стають захищеними членами похідного класу до отже, доступні решті членів похідного класу. Іншими словами захищені члени класу по відношенню до свого класу є закритими і в той же час, можуть успадковуватися похідним класом. Розглянемо приклад.  
#include <iostream>   
using namespace std;  
class base {   
protected:  
int i, j; // Закриті по відношенню до класу base  
// але доступні класу derived.  
public:  
void set(int а, int b) { i=a; j=b; }  
void showO { cout « i « " " « j « "\n"; }   
};  
class derived : public base {  
int k;   
public:  
// Клас derived має доступ до членів i і j з класу base  
void setk() { k=i\*j; }  
void showkO { cout « k << "\n"; }  
};  
int main() {  
derived ob;  
ob.set(2, 3);   
// Все гаразд, цей член доступний класу derived  
ob.showO; // Все гаразд, цей член доступний класу derived  
ob.setk();   
ob.showk();  
return 0;  
}  
  
У даному прикладі, оскільки клас derived успадковує властивості класу base за допомогою відкритого наслідування, а змінні i і j оголошені захищеними, функція seek () з класу derived має до них доступ. Якби змінні i і j були оголошені в класі base закритими, то клас derived не мав би до них доступу, і програму не можна було скомпілювати.  
Якщо похідний клас є базовим по відношенню до іншого похідного класу, то будь-який захищений член початкового базового класу, відкрито успадкований першим похідним класом, також може успадковуватися другим похідним класом як захищений член.   
Захищене наслідування  
До базового класу можна застосовувати механізм захищеного наслідування. При цьому всі відкриті і захищені члени базового класу стають захищеними членами похідного класу. Розглянемо приклад.  
#include <iostream>  
using namespace std;  
class base {  
protected:  
int i, j; // Закриті члени класу base  
// доступні класу derived.  
public:  
void setij(int а, int b) { i=a; j=b; }  
void showij() { cout « i « " " « j « "\n"; }  
};  
class derived : protected base {  
int k;  
public:  
//Клас derived має доступ до членів i, j і setij()   
// з класу base  
void setk() { setijd(10, 12); до = i\*j; }  
// Звідси можна викликати функцію showij().   
void showall() { cout << k « " "; showij(); }  
};  
int main() {  
derived ob;  
ob.setij (2, 3);   
// Невірно, функція setij() є закритим членом класу derived.  
ob.setk();   
// Вірно, викликається відкритий член класу derived.   
ob.showall();   
// Вірно, викликається відкритий член класу derived.  
//ob.showij () ; // Невірно, функція showij() є   
// захищеним членом класу derived  
return 0;  
}  
Як випливає з коментарів, не дивлячись на те що функції setij() і showij() є відкритими членами класу base, в класі derived, утвореному за допомогою захищеного наслідування, вони стають захищеними. Це означає, що функції main() вони не доступні.  
Множинне наслідування  
Похідний клас може одночасно успадковувати властивості декілька базових Наприклад, в програмі, приведеній нижче, клас derived успадковує властивості класів base1 і base2.  
Приклад множинного наслідування.  
include <iostream>   
using namespace std;  
class base1 {   
protected:  
int x;  
puclic:  
void showx() { cout << x « "\n"; }  
};  
class base2 {   
protected:  
int у;   
public:  
void showy() {cout « у « "\n";}  
};  
  
// Множинне Наслідування.  
class derived: public basel, public base2 {  
public:  
void set(int i, int j) { x=i; y=j; }   
};  
int main() {  
derived ob;  
ob.set(10, 20); // Ця функція належить класу derived.  
ob.showx(); // Ця функція належить класу basel.   
ob.showy(); // Ця функція належить класу base2.  
return 0;  
}  
Як видно, при множинному наслідуванні імена базових класів перераховуються в списку і розділяються комами, причому перед кожним ім'ям базового класу указується свій специфікатор доступу.

**Статичні члени класу**

Такі компоненти повинні бути визначені в класі як **статичні** (**static**). Статичні дані класів не дублюються при створенні об'єктів, тобто кожен статичний компонент існує в єдиному екземплярі. Доступ до статичного компонента можливий тільки після його ініціалізації. Для ініціалізації використовується конструкція

*тип ім'я\_класу : : ім'я\_даного ініціалізатор;*

Наприклад, int complex : : count = 0;

Ця визначення повинно бути розміщене в глобальній області після визначення класу. Тільки при ініціалізації статичне дане класу одержує пам'ять і стає доступним. Звертатися до статичного даного класу можна звичайним способом через ім'я об'єкта

*ім'я\_об'єкта.ім'я\_компонента*

Але до статичних компонентів можна звертатися і тоді, коли об'єкт класу ще не існує. Доступ до статичних компонентів можливий не тільки через ім'я об'єкта, але і через ім'я класу

*ім'я\_класу : : ім'я\_компонента*

Однак так можна звертатися тільки до *public* компонентів.

Для звертання до *private* статичного компонента ззовні можна за допомогою **статичних компонентів-функцій**. Ці функції можна викликати через ім'я класу.

*ім'я\_класу : : ім'я\_статичної\_функції*

Приклад.

#include <iostream.h>

class **TPoint**

{

double x,y;

static int N; // статичний компонент- дане : кількість крапок

public:

TPoint(double x1 = 0.0,double y1 = 0.0){N++; x = x1; y = y1;}

static int& count(){return N;} // статичний компонент-функція

};

int TPoint : : N = 0; //ініціалізація статичного компонент-даного

void **main**(void)

{TPoint A(1.0,2.0);

TPoint B(4.0,5.0);

TPoint C(7.0,8.0);

cout<<“\nВизначені ”<<TPoint : : count()<<“крапки.”; }

**Покажчик this**

Коли функція-член класу викликається для обробки даних конкретного об'єкта, для цієї функції автоматично і неявно передається покажчик на той об'єкт, для якого функція викликана. Цей покажчик має ім'я **this** і неявно визначене в кожній функції класу в такий спосіб

ім'я\_класу \*const this = адреса\_об'єкта

Покажчик this є додатковим схованим параметром кожної нестатичної компонентної функції. При вході в тіло приналежної класу функції this ініціюється значенням адреси того об'єкта, для якого викликана функція. В результаті цей об'єкт стає доступним всередині цієї функції.

В більшості випадків використання **this** є неявним. Зокрема, кожне звертання до нестатичного функції-члену класу неявно використовує this для доступу до члена відповідного об'єкта.

Прикладом широко розповсюдженого явного використання this є операції зі зв'язаними списками.

**Успадкування**

Успадкування — це механізм одержання нового класу на основі вже існуючого. Існуючий клас може бути доповнений чи змінений для створення нового класу.

Існуючі класи називаються **базовими**, а нові – **похідними**. Похідний клас успадковує опис базового класу; потім він може бути змінений додаванням нових членів, зміною існуючих функцій-членів і зміною прав доступу. За допомогою спадкування може бути створена ієрархія класів, що спільно використовують код і інтерфейси.

Наслідувані компоненти не переміщаються в похідний клас, а залишаються в базових класах.

В ієрархії похідний об'єкт успадковує дозволені для спадкування компонента всіх базових об'єктів (*public, protected*).

Допускається множинне спадкування – можливість для деякого класу успадковувати компоненти декількох ніяк не зв'язаних між собою базових класів. В ієрархії класів угода щодо приступності компонентів класу наступні:

***private*** *–* Член класу може використовуватися тільки функціями-членами даного класу і функціями-“друзями” свого класу. У похідному класі він недоступний.

***protected*** – Те ж, що і ***private,*** але додатково член класу з даним атрибутом доступу може використовуватися функціями-членами і функціями-“друзями” класів, похідних від даного.

***public*** – Член класу може використовуватися будь-якою функцією, яка є членом даного чи похідного класу, а також до ***public*** - членів можливий доступ ззовні через ім'я об'єкта.

Варто мати на увазі, що оголошення friend не є атрибутом доступу і не успадковується.

Синтаксис визначення похідного класу :

***class ім'я\_класу : список\_базових\_класів***

***{список\_компонентів\_класу};***

У похідному класі успадковані компоненти одержують статус доступу **private**, якщо новий клас визначений за допомогою ключового слова **class**, і статус **public**, якщо за допомогою **struct**

Явно змінити статус доступу, що замовчується, при спадкуванні можна за допомогою атрибутів доступу *- private, protected* і *public*, що вказуються безпосередньо перед іменами базових класів.

**Конструктори і деструктори похідних класів**

Оскільки конструктори не успадковуються, при створенні похідного класу наслідувані їм дані-члени повинні ініціюватися конструктором базового класу. Конструктор базового класу викликається автоматично і виконується до конструктора похідного класу. Параметри конструктора базового класу вказуються у визначенні конструктора похідного класу. У такий спосіб відбувається передача аргументів від конструктора похідного класу конструктору базового класу.

Наприклад.

class Basis

{ int a,b;

public:

Basis(int x,int y){a=x;b=y;}

};

class Inherit:public Basis

{int sum;

public:

Inherit(int x,int y, int s):Basis(x,y){sum=s;}

};

Об'єкти класу конструюються знизу-вверх: спочатку базовий, потім компоненти-об'єкти (якщо вони маються), а потім сам похідний клас. Таким чином об'єкт похідного класу містить у якості підоб’єкта об'єкт базового класу.

Знищуються об'єкти в зворотному порядку: спочатку похідний, потім його компоненти-об'єкти, а потім базовий об'єкт.

Таким чином, порядок знищення об'єкта протилежний стосовно порядку його конструювання.

**Віртуальні функції.**

До механізму віртуальних функцій звертаються в тих випадках, коли в кожному похідному класі потрібно свій варіант деякої компонентної функції. Класи, що включають такі функції, **називаються** поліморфними і відіграють особливу роль в ООП.

Віртуальні функції надають механізм **пізнього** (**відкладеного**) чи **динамічного** **зв'язування.** Будь-яка нестатична функція базового класу може бути зроблена віртуальною, для чого використовується ключове слово **virtual**.

Приклад.

class base

{

public:

**virtual** void print(){cout<<“\nbase”;}

. . .

};

class **dir** : public base

{

public:

void print(){cout<<“\ndir”;}

};

void **main**()

{

base B,\*bp = &B;

dir D,\*dp = &D;

base \*p = &D;

bp –>print(); // base

dp –>print(); // dir

p –>print(); // dir

}

Таким чином інтерпретація кожного виклику віртуальної функції через покажчик на базовий клас залежить від значення цього покажчика, тобто від типу об'єкта, для якого виконується виклик.

Вибір того, яку віртуальну функцію викликати буде залежати від типу об'єкта, на який фактично (у момент виконання програми) спрямований покажчик, а не від типу покажчика.

Віртуальними можуть бути тільки нестатичні функції-члени.

Віртуальність успадковується. Після того як функція визначена як віртуальна, її повторне визначення в похідному класі (з тим же самим прототипом) створює в цьому класі нову віртуальну функцію, причому специфікатор virtual може не використовуватися.

Конструктори не можуть бути віртуальними, на відміну від деструкторів. Практично кожен клас, що має віртуальну функцію, повинний мати віртуальний деструктор.

**Абстрактні класи**

Абстрактним називається клас, у якому є хоча б одна чиста (порожня) віртуальна функція.

Чистою віртуальною функцією називається компонентна функція, що має наступне визначення:

***virtual тип ім'я\_функції(список\_формальних\_параметрів) = 0;***

Чиста віртуальна функція нічого не робить і недоступна для викликів. Її призначення – бути основою для підмінюючих її функцій у похідних класах. Абстрактний клас може використовуватися тільки в якості базового для похідних класів.

Механізм абстрактних класів розроблений для представлення загальних понять, що надалі передбачається конкретизувати. При цьому побудова ієрархії класів виконується за наступною схемою. На чолі ієрархії стоїть абстрактний базовий клас. Він використовується для спадкування інтерфейсу. Похідні класи будуть конкретизувати і реалізувати цей інтерфейс. В абстрактному класі оголошені чисті віртуальні функції, що по суті є **абстрактними методи**.

Приклад .

class Base{

public:

Base(); // конструктор за замовчуванням

Base(const Base&); // конструктор копіювання

virtual ~Base(); // віртуальний деструктор

virtual void Show()=0; // чиста віртуальна функція

// інші чисті віртуальні функції

protected: // захищені члени класу

private:

// часто залишається порожнім, інакше буде заважати майбутнім розробкам

};

class Derived: virtual public Base{

public:

Derived(); // конструктор за замовчуванням

Derived(const Derived&); // конструктор копіювання

Derived(параметри); // конструктор з параметрами

virtual ~Derived(); // віртуальний деструктор

void Show(); // перевизначена віртуальна функція

// інші перевизначені віртуальні функції

// інші перевантажені операції

protected:

// використовується замість private, якщо очікується спадкування

private:

// використовується для деталей реалізації

};

Об'єкт абстрактного класу не може бути формальним параметром функції, однак формальним параметром може бути покажчик на абстрактний клас. У цьому випадку з'являється можливість передавати у функцію, що викликається, як фактичний параметр значення покажчика на похідний об'єкт, замінюючи ним покажчик на абстрактний базовий клас. У такий спосіб ми одержуємо **поліморфні об'єкти**.

Абстрактний метод може розглядатися як узагальнення *перевизначення.* В обох випадках поводження батьківського класу змінюється для нащадка. Для абстрактного методу, однак, поводження просто не визначене. Будь-яке поводження задається в похідному класі.

Одна з переваг абстрактного методу є чисто концептуальною: програміст може думкою наділити потрібною дією абстракцію як завгодно високого рівня. Наприклад, для геометричних фігур ми можемо визначити метод *Draw*, що їх малює: трикутник *TTriangle*, окружність *TCircle*, квадрат *TSquare.* Ми визначимо аналогічний метод і для абстрактного батьківського класу *TGraphObject*. Однак, такий метод не може виконувати корисну роботу, оскільки в класі *TGraphObject* просто немає достатньої інформації для малювання чого-небудь. Проте присутність методу *Draw* дозволяє зв'язати функціональність (малювання) тільки один раз із класом *TGraphObject,* а не вводити три незалежні концепції для підкласів *TTriangle*, *TCircle*, *TSquare.*

Є і друга, більш актуальна причина використання абстрактного методу. В об’єктно-орієнтованих мовах програмування зі статичними типами даних, до яких відноситься і С++, програміст може викликати метод класу, тільки якщо компілятор може визначити, що клас дійсно має такий метод. Припустимо, що програміст хоче визначити поліморфну змінну типу *TGraphObject,* що буде в різні моменти часу містити фігури різного типу. Це припустимо для поліморфних об'єктів. Проте компілятор дозволить використовувати метод *Draw* для перемінної, тільки якщо він зможе гарантувати, що в класі змінної є цей метод. Приєднання методу *Draw* до класу *TGraphObject* забезпечує таку гарантію, навіть якщо метод *Draw* для класу  *TGraphObject* ніколи не виконується. Природно для того, щоб кожна фігура малювалася по своєму, метод *Draw* повинний бути *віртуальним.*

|  |  |  |
| --- | --- | --- |
| **9.1.** **Приклади програм**  **Задача 9.1**  Визначити клас "рядок" і перевантажити операцію + для з'єднання рядків. Використати різні категорії конструкторів та специфікації доступу до атрибутів.  **Об'єктно-орієнтований аналіз**  ***Об'єкт***- це рядок. Атрибути (властивості) об'єкта такі: довжина рядка, символи рядка у вигляді масиву символів. Методи об'єкта такі: створення пустого рядка, ініціалізація атрибутів об'єкта, копіювання об'єкта, з'єднання об'єктів. Для створення рядків, що з'єднуються, використовується конструктор, який ініціалізує такі атрибути: ім'я об'єкта та символи рядка. Для створення об'єкта, який є результатом з'єднання рядків, використовується конструктор, який ініціалізує такі атрибути: ім'я об'єкта та довжину об'єднаного рядка. Для з'єднання двох об'єктів використовується операторна функція, яка виконує цю операцію за допомогою стандартної арифметичної операції додавання (+). Під час виконання операції додавання створюється новий об'єкт з атрибутами заданої довжини рядка, іменем об'єкта, який заповнений пустими символами. З'єднання рядків виконується стандартною функці-єю конкатенації. Під час повернення результату з функції з'єднання викликається конструктор копії, який створює новий об'єкт на основі копії вже існуючого. Під час повернення з функції посилання на об'єкт конструктор копії не викликається, тому що функція змінює значення фактичних об'єктів, а не їх копії. Для доступу до закритих атрибутів класу використовується функція доступу, яка повертає значення відповідного атрибута об'єкта.  **Об'єктно-орієнтоване програмування**  ***Структура проекту програми 9.1***  image006  **Текст програми 9.1**  //Файл lab9\_1.h  - інтерфейс класу  class stroka  {  private:                       //закриті атрибути класу              int len; //довжина рядка  public:                        //відкриті атрибути та методи  char name\_str[10];   //ім’я об’єкта  char str[50];               //символи рядка  stroka(char name[10]);                                //конструктор ініціалізації  stroka (stroka& s);                           //конструктор копії  stroka(char name[10],int size);       //конструктор ініціалізації  stroka operator+(stroka& s);          //операторна функція перевантаження операції +  int get();                                             //функція доступу до закритих атрибутів  void show();                                      //відображення атрибутів об’єкта  };  //////////////////////////////////////////////////////////////////////////////////////////////////////  //Файл lab9\_1.cpp -реалізація класу  #include<stdio.h>  #include<iostream.h>  #include<string.h>  #include"lab9\_1.h"               //підключення заголовного файлу інтерфейсу класу  /////////////////////   конструктор ініціалізації          ////////////////////////  stroka::stroka(charname[10])  {                              //викликається під час створення об’єктів, які будуть з’єднуватися              strcpy(name\_str,name);              cout<<"Enter string "<<name\_str<<endl;              gets(str);        len=strlen(str);  }  /////////////////////////     конструктор ініціалізації   /////////////////////////  stroka::stroka(char name[10],int size)    {     //викликається під час створення об’єкта –результату з’єднання двох інших об’єктів              strcpy(name\_str,name);              strcpy(str," ");              len=size;   }  ///////////////////////  відображення атрибутів об’єкта ///////////////////  voidstroka::show()  {           cout<<"Initialization string  "<<name\_str<<endl;              puts(str);  }  ////////////////////////////  конструктор копії  ///////////////////////   stroka::stroka (stroka& s)   /\*викликається для копіювання результату з’єднання об’єктів в третій об’єкт  під час повернення результату з операторної функції. Метод відсутній, якщо операторна функція повертає посилання на об’єкт типу stroka& \*/      {              strcpy(name\_str,s.name\_str);               strcpy(str,s.str);     }  //////////////////////////////////////// операторна функція з’єднання об’єктів/////////////////////////////////  stroka stroka::operator+(stroka& s)  {                                        //можливий варіант повертання посилання на об’єкт stroka&              stroka tmp("str3",len+s.len);                        //створення об’єкта, що повертається              strcpy(tmp.str,str);              strcat(tmp.str,s.str);                               //об’єднання рядків              return tmp;                                               //виклик конструктор копії  }  ////////////////////////////////   функція доступу до закритих атрибутів /////////////////////////  intstroka::get()  {           returnlen;   }  /////////////////////////////  ТЕСТОВА ПРОГРАМА   ////////////////////////////////////  voidmain()    {  stroka a("str1");                        //ініціалізація першого об’єкта значеннями з клавіатури  a.show();                                                          //відображення атрибутів першого об’єкта  stroka b("str2");                      //ініціалізація другого об’єкта значеннями з клавіатури  b.show();                                                            //відображення атрибутів другого об’єкта  stroka c("str3",a.get()+b.get());      //ініціалізація об’єкта, який є з’єднанням об’єктів  c=a+b;                                                                                         // з’єднання двох  об’єктів  c.show();                                               //відображення атрибутів результуючого об’єкта  }        **Задача 9.2**  Розробити клас "матриця", який є похідним від класу "вектор". Клас "вектор" є похідним від класу "масив". Використати перевантажені констру-ктори. Виконати операції множення матриці на вектор та додавання матриць.  **Об'єктно-орієнтований аналіз**   |  |  | | --- | --- | | image008 | Ієрархія класів подана на рис. 9.1. Базовим класом є клас "масив" (array) з атрибутом - назва масиву (name), похідним від класу "масив" є клас "вектор" (vector) з атрибутами: кількість елементів (row), значення компонентів вектора (mas). Похідним від класу "вектор" є клас "матриця" (matrix) з атрибутами: кіль-кість стовпчиків матриці (column) та значення елементів матриці (matr).  Кожний клас має конструктор об'єктів та метод show() відображення їх атрибутів. Клас "матриця" має методи множення матриці на вектор mult() та додавання матриць add(). Класи мають перевантажені конструктори, які використовуються під час ініціалізації об'єктів в різних умовах. |     ***Сценарій взаємодії об'єктів***  1. Конструювання об'єкта "вектор" за допомогою конструктора неявно-го об'єкта "масив". 2. Ініціалізіція атрибутів об'єкта "вектор". 3. Відображення значень атрибутів об'єкта "вектор". 4. Конструювання об'єкта "матриця" за допомогою конструктора об'єкта "вектор", який створюється на базі об'єкта "масив". 5. Ініціалізація об'єкта "матриця". 6. Відображення атрибутів об'єкта "матриця". 7. Конструювання та ініціалізація об'єкта "матриця", який додається до першого об'єкта "матриця". 8. Додавання двох об'єктів "матриця". 9. Відображення результатів додавання об'єктів "матриця" 10. Конструювання пустого об'єкта "вектор", для збереження результату множення матриці на вектор. 11. Множення об'єктів "вектор" та першого об'єкта "матриця" з ініціал-зацією об'єкта "вектор", що є результатом множення.  **Об'єктно-орієнтоване програмування**  ***Структура проекту програми 9.2***  image012  **Текст програми 9.2**  //==============Файл  lab9\_2.h – інтерфейси класів=======================  class array                                                                                                            //клас масив  {   public:     char name[10];                                                                                                //ім’я масиву     array();                                                                                                           //конструктор     virtual void show();                                                       //відображення атрибутів об’єкта  };  ////////////////////////////////////////////////////////       клас вектор     ///////////////////////////////////////////////////  class vector:public array  { public:              intmas[5];                                                                                  //компоненти вектора              int row;                                                                                    //кількість компонентів      vector();                                                                                                          //конструктор      vector(int number\_element);                                             //перевантажений конструктор      virtual void show();                                                      //відображення атрибутів об’єкта  };  /////////////////////////////////////////////////   клас матриця   //////////////////////////////////////////////////////////////  Class matrix:public vector  { public:         intmatr[5][5];                                                                          //елементи матриці         int  column;//кількість стовпчиків, кількість рядків успадковується з класу вектор     matrix();                                                                                                          //конструктор     matrix(int number\_column);                                                //перевантажений конструктор     void show();                                                         //відображення атрибутів об’єкта     void mult(vector &v);                                                            //множення матриці на вектор     void add(matrix &other\_matr);                                               //додавання матриць  };  //===============Файл lab9\_2.срр - реалізація класів =====================//  #include<iostream.h>  #include<stdio.h>  #include<string.h>  #include"lab9\_2.h"  intflag=0;                                   //прапорець для визначення типу виведення  атрибутів  array::array()                                             //конструктор ініціалізації атрибутів класу array  {           cout<<"enter name of object"<<endl;              gets(name);  }  //////////////////////////////// відображення атрибутів об’єкта  //////////////////////////////////////////////////   void array::show()  {           cout<<name;   }  /////////// конструктор об’єкта vector, посилається на конструктор об’єкта array ////////////  vector::vector():array()  {           puts("enter size of column-vector");              cin>>row;                                                 //уведення кількості компонентів вектора              cout<<"enter "<<row<<" values of column-vector "<<endl;              for(inti=0;i<row;i++) cin>>mas[i];           //уведення значення компонентів  }  /////////конструктор об’єкта vector, посилається на конструктор об’єкта array  /////////////  vector::vector(intnumber\_row):array()  {           row=number\_row;  }  ///////////////////////////////  метод відображення атрибутів об’єкта /////////////////////////////////////////   void vector::show()  {           cout<<"vector atributes: name=";              array::show();              cout<<"  row="<<row<<endl;              cout<<" values of column-vector:"<<endl;              for(inti=0;i<row;i++) cout<<mas[i]<<endl;  }  /////////////////////////////    метод додавання матриць      ////////////////////////////////////////////////////  voidmatrix::add(matrix &other\_matr)  {                                                                                     //визначення можливості додавання              if((row==other\_matr.row)&&(column==other\_matr.column))               for(inti=0;i<row;i++)                           for(intj=0;j<column;j++)                                      matr[i][j]+=other\_matr.matr[i][j];              else     {  cout<<"adding is impossible"<<endl;                              flag=1; }                                                  //ознака неможливості додавання  }  ///////////////////////////////// метод  множення матриці на вектор /////////////////////////////////////////////  voidmatrix::mult(vector &v){              vector result(column);                                                   //об’єкт результату множення              if(v.row==column)                                           //умова можливості множення              {    for (inti=0;i<row;i++)                          {           result.mas[i]=0;                                     for(intj=0;j<column;j++)                                          result.mas[i]+=matr[i][j]\*v.mas[j];                          }              }              else     cout<<"multiplication is impossible"<<endl;            result.show();                                      //відображення результатів множення  }  /////////////////////////////////////   конструктор матриці   ////////////////////////////////////////////////////////////  matrix::matrix(intnumber\_row):vector(number\_row)   {   char answer;       //символ-відповідь на питання про необхідність уведення даних        cout<<"do you want to enter row ?y/n:";        cin>>answer;        if(answer=='n')        //якщо кількість рядків не вводиться, то вона успадковується         {             cout<<"row is defined by vector"<<endl;              row= number\_row;         }       else                                            //потрібне введення нового значення кількості рядків          {  cout<<"enter row of matrix: ";              cin>>row;          }       cout<<"enter column of matrix"<<endl;       cin>>column;                                                                    //уведення кількості стовпчиків       cout<<"enter "<<row<<"\*"<<column<<" values of object's elements "<<endl;       for(inti=0;i<row;i++)                                                          //уведення елементів матриці              for(intj=0;j<column;j++)                   cin>>matr[i][j];    }   void matrix::show()   {              puts("");               for (int i = 0; i < row; i++)               {                           for (int j = 0; j < column; j++)                           {                                      printf("%3d", matr[i][j]);                           }               puts("");               }               puts("");   }  //////////////////////////////////        тестова програма    //////////////////////////////////////  void main()   {   puts("inheritance -lab9\_2");       vector vec;   vec.show();//конструювання вектора та відображення його атрибутів       int size\_row=vec.row;  //формування аргументу для конструктора об’єкту матриці       cout<<"matrix1: ";       matrix mat1(size\_row);                                                       //конструювання матриці       mat1.show();                                                                           //відображення матриці       cout<<"matrix2: ";       matrix mat2(size\_row);       mat2.show();                                         //конструювання матриці та її відображення       mat1.add(mat2);                                                                         //додавання матриць       if(flag==0)         {    cout<<"result of matrix adding"<<endl;               mat1.show();          }       cout<<"result of  multiplication matrix by vector"<<endl;       mat1.mult(vec);   getchar();   //множення матриці на вектор, затримка зображення  } |

**Порядок виконання роботи.**

1.Визначити ієрархію класів (відповідно до варіанта).

2.Визначити в класі статичний компонент - покажчик на початок зв'язаного списку об'єктів і статичну функцію для перегляду списку.

3.Реалізувати класи.

4.Написати демонстраційну програму, у якій створюються об'єкти різних класів і поміщаються в список, після чого список проглядається.

5.Зробити відповідні методи не віртуальними і подивитися, що буде.

6.Реалізувати варіант, коли об'єкт додається в список при створенні, тобто в конструкторі (дивися пункт 6 наступного розділу).

**Методичні вказівки.**

1.Для визначення ієрархії класів зв'язати відношенням спадкування класи, приведені в додатку (для заданого варіанта). З перерахованих класів вибрати один, який буде стояти на чолі ієрархії. Це абстрактний клас.

2.Визначити у класах усі необхідні конструктори і деструктор.

3.Компонентні дані класу потрібно специфікувати як **protected**.

4.Приклад визначення статичних компонентів:

static person\* begin; // покажчик на початок списку

static void print(void); // перегляд списку

5.Статичну компоненту-дане потрібно ініціювати поза визначенням класу, у глобальній області.

6.Для додавання об'єкта в список варто передбачити метод класу, тобто об'єкт сам додає себе в список. Наприклад, a.Add() - об'єкт **a** додає себе в список.

Включення об'єкта в список можна виконувати при створенні об'єкта, тобто помістити оператори включення в конструктор. У випадку ієрархії класів, включення об'єкта в список повинний виконувати тільки конструктор базового класу. Ви повинні продемонструвати обидва ці способи.

7.Список проглядається шляхом виклику віртуального методу **Show** кожного об'єкта.

8.Статичний метод перегляду списку викликайте не через об'єкт, а через клас.

9.Визначення класів, їхню реалізацію, демонстраційну програму помістити в окремі файли.

**Зміст звіту.**

1.Титульний лист: назва дисципліни, номер і найменування роботи, прізвище, ім'я, по батькові студента, дата виконання.

2.Постановка задачі. Варто дати конкретну постановку, тобто вказати які класи повинні бути реалізовані, які повинні бути в них конструктори, компоненти-функції і т.д.

3.Ієрархія класів у виді графа.

4.Визначення класів користувача з коментарями.

5.Реалізація конструкторів з параметрами і деструктора.

6.Реалізація методів для додавання об'єктів в список .

7. Реалізація методів для перегляду списку.

8.Лістінг демонстраційної програми.

9.Пояснення необхідності віртуальних функцій. Варто показати, які результати будуть у випадку віртуальних і не віртуальних функцій.

**Приклади виконаних завдань**

**Варіант 10.**

**PZ.h**

#pragma once

#include <string.h>

class PZ

{

public:

PZ(void);

~PZ(void);

virtual void show()=0;

virtual void use()=0;

virtual void Input()=0;

};

**PZ.cpp**

#include "StdAfx.h"

#include "PZ.h"

PZ::PZ(void)

{

}

PZ::~PZ(void)

{

}

**Free.h**

#pragma once

#include "PZ.h"

#include "DataTimes.h"

#include <string>

#include <cstdlib>

using namespace std;

class Free :

virtual public PZ

{

public:

string name,dev;

Free(string name,string dev);

Free();

void Input();

~Free(void);

void show();

void use();

};

**Free.cpp**

#include "StdAfx.h"

#include "Free.h"

#include <string>

#include <cstdlib>

#include <iostream>

#include <string>

#include "PZ.h"

Free::Free(string n,string d)

{

name=n;

dev=d;

}

Free::Free():name("A"),dev("KhM"){};

void Free::Input() {

setlocale(LC\_ALL,"rus");

cout<<"Введите названия програмного средства:";

cin>>name;

cout<<"Введите названия студии разработчика:";

cin>>dev;

}

void Free::use(){

setlocale(LC\_ALL,"rus");

cout<<"Вы можете всегда использывать даное програмоное обеспечение"<<"\n";

}

void Free::show() {

setlocale(LC\_ALL,"rus");

cout<<"Прoграмное средство:\""<<name<<"\" производитель:\""<<dev<<"\"\n";

}

Free::~Free(void)

{

}

**Shareware.h**

#pragma once

#include "PZ.h"

#include "DataTimes.h"

#include <string>

#include <cstdlib>

#include <iostream>

#include <string>

using namespace std;

class Shareware :

virtual public PZ

{

public:

string name,dev;

DataTimes installdata;

int dayoffree;

void use();

void show();

Shareware(string s1,string s2,DataTimes intsd,int countday);

Shareware();

void Input();

~Shareware(void);

};

**Shareware.cpp**

#include "StdAfx.h"

#include "Shareware.h"

#include <string>

#include <cstdlib>

#include <iostream>

#include <string>

#include <time.h>

#include "DataTimes.h"

#include "PZ.h"

Shareware::Shareware(string s1,string s2,DataTimes intsd,int countday)

{

name=s1;

dev=s2;

installdata=intsd;

dayoffree=countday;

}

void Shareware::Input()

{

setlocale(LC\_ALL,"rus");

cout<<"Введите названия програмного средства:";

cin>>name;

cout<<"Введите названия студии разработчика:";

cin>>dev;

cout<<"Введите дату установки:";

installdata.Input();

cout<<"Введите количесвто дней бесплатного использивания";

cin>>dayoffree;

}

void Shareware::use() {

setlocale(LC\_ALL,"rus");

DataTimes dcon;

dcon=installdata;

dcon.plusDay(dayoffree);

int dn,mn,yyyn;

time\_t t;

tm \*tk;

time(&t);

tk=localtime(&t);

dn=tk->tm\_mday;

mn=1+tk->tm\_mon;

yyyn=1900+tk->tm\_year;

bool flag=true;

if(yyyn>dcon.yyyy) flag=false;

else

{

if(mn>dcon.mm) flag=false;

else

{

if(dn>dcon.dd) flag=false;

}

}

if(flag)

{

cout<<"Вы можете использовать даную програму, дата окончания пробной версии ";

dcon.Print();

}

else

{

cout<<"Вы не можете использовать даную програму, лицензия истекла ";

dcon.Print();

}

}

Shareware::Shareware(){

name="A";

dev="B";

installdata.setAll(1,1,2012);

dayoffree=365;

}

void Shareware::show(){

setlocale(LC\_ALL,"rus");

cout<<"Прoграмное средство:\""<<name<<"\" производитель:\""<<dev<<"\"\n";

cout<<"Срок действия триала "<<dayoffree<<"\n";

cout<<"Дата инсталяции";

installdata.Print();

}

Shareware::~Shareware(void)

{

}

**Busines.h**

#pragma once

#include "pz.h"

#include "DataTimes.h"

#include <string>

#include <cstdlib>

using namespace std;

class Busines :

virtual public PZ

{

public:

string name,dev;

DataTimes installdata;

int dayoffree;

void Input();

void use();

void show();

Busines(string s1,string s2,DataTimes intsd,int countday);

Busines(void);

~Busines(void);

};

**Busines.cpp**

#include "StdAfx.h"

#include "Busines.h"

#include <string>

#include <cstdlib>

#include <iostream>

#include <string>

#include <time.h>

#include "DataTimes.h"

#include "PZ.h"

using namespace std;

void Busines::use() {

setlocale(LC\_ALL,"rus");

DataTimes dcon;

dcon=installdata;

dcon.plusDay(dayoffree);

int dn,mn,yyyn;

time\_t t;

tm \*tk;

time(&t);

tk=localtime(&t);

dn=tk->tm\_mday;

mn=1+tk->tm\_mon;

yyyn=1900+tk->tm\_year;

bool flag=true;

if(yyyn>dcon.yyyy) flag=false;

else

{

if(mn>dcon.mm) flag=false;

else

{

if(dn>dcon.dd) flag=false;

}

}

if(flag)

{

cout<<"Вы можете использовать даную програму, дата окончания лицензии";

dcon.Print();

}

else

{

cout<<"Вы не можете использовать даную програму, лицензия истекла ";

dcon.Print();

}

}

void Busines::Input()

{

setlocale(LC\_ALL,"rus");

cout<<"Введите названия програмного средства:";

cin>>name;

cout<<"Введите названия студии разработчика:";

cin>>dev;

cout<<"Введите дату установки:";

installdata.Input();

cout<<"Введите количесвто дней лицензии";

cin>>dayoffree;

}

void Busines::show(){

setlocale(LC\_ALL,"rus");

cout<<"Прoграмное средство:\""<<name<<"\" производитель:\""<<dev<<"\"\n";

cout<<"Срок действия "<<dayoffree<<"\n";

cout<<"Дата инсталяции";

installdata.Print();

}

Busines::Busines(void)

{

}

Busines::Busines(string s1,string s2,DataTimes intsd,int countday)

{

name=s1;

dev=s2;

installdata=intsd;

dayoffree=countday;

}

Busines::~Busines(void)

{

}

**DataTimes.h**

#pragma once

class DataTimes

{

public:

unsigned short int dd;

unsigned short int mm;

unsigned int yyyy;

public:

DataTimes();

DataTimes(unsigned short int d,unsigned short int m,unsigned int y);

~DataTimes();

void setAll(unsigned short int d,unsigned short int m,unsigned int y);

int getYear();

int getMounth();

int getDay();

DataTimes Input();

void Print();

DataTimes NextDay(DataTimes ob);

DataTimes PrevDay(DataTimes ob);

int CountDay();

void plusDay(int count);

void DayToDate();

bool operator&(DataTimes ob1);

bool operator!();

void DateTrue();

bool DataValid();

};

**DataTimes.cpp**

#include "StdAfx.h"

#include "DataTimes.h"

#include <iostream>

#include <string>

using namespace std;

DataTimes::DataTimes():dd(1),mm(1),yyyy(2009){};

DataTimes::DataTimes(unsigned short int d = 1,unsigned short int m = 1,unsigned int y = 2009)

{

}

DataTimes::~DataTimes(void)

{

}

void DataTimes::setAll(unsigned short int d,unsigned short int m,unsigned int y)

{

dd=d;

mm=m;

yyyy=y;

}

int DataTimes::getYear()

{

return yyyy;

}

int DataTimes::getMounth()

{

return mm;

}

int DataTimes::getDay()

{

return dd;

}

DataTimes DataTimes::Input()

{

setlocale(LC\_ALL,"rus");

cout << "Введите год:";

cin >> yyyy;

cout << "Введите месяц:";

cin >> mm;

cout << "Введите день:";

cin >> dd;

setlocale(LC\_ALL,"");

return DataTimes(dd,mm,yyyy);

}

DataTimes DataTimes::NextDay(DataTimes ob)

{

this->dd=ob.dd;

this->mm=ob.mm;

this->yyyy=ob.yyyy;

this->dd++;

if((mm==1)||(mm==3)||(mm==5)||(mm==7)||(mm==8)||(mm==10)||(mm==12))

{

if(dd>31) {dd=1;mm++;}

} else

{

if((mm==4)||(mm==6)||(mm==9)||(mm==11))

{

if(dd>30) {dd=1;mm++;}

} else

{

if(((yyyy%4==0) && (yyyy%100!=0))||((yyyy%4==0) && (yyyy%100!=0) && (yyyy%400==0)))

{

if(dd>29)

{

dd=1;mm++;

}

} else

{

if(dd>28)

{

dd=1;mm++;

}

}

}

}

if(mm>12) {

mm=1;

yyyy++;

}

return DataTimes(dd,mm,yyyy);

}

DataTimes DataTimes::PrevDay(DataTimes ob)

{

this->dd=ob.dd;

this->mm=ob.mm;

this->yyyy=ob.yyyy;

this->dd--;

if((mm==1)||(mm==3)||(mm==5)||(mm==7)||(mm==8)||(mm==10)||(mm==12))

{

if(dd<1) {dd=30;mm--;

if(((yyyy%4==0) && (yyyy%100!=0)&& (mm==2))||((yyyy%4==0) && (yyyy%100!=0) && (yyyy%400==0) && (mm==2)))

{

if(dd<1)

{

dd=29;

}

} else

{

if(dd<1)

{

dd=28;

}

}}

} else

{

if((mm==2)||(mm==4)||(mm==6)||(mm==9)||(mm==11))

{

if(dd<1) {dd=31;mm--;}

}

}

if((mm==0)) {

mm=12;

yyyy--;

}

return DataTimes(dd,mm,yyyy);

}

int DataTimes::CountDay()

{

switch (mm)

{

case 1:case 3:case 5:case 7:case 8:case 10:case 12: { return 31-dd;break;};

case 4:case 6:case 9:case 11: {return 30-dd;break;}

case 2:

{

if(((yyyy%4==0) && (yyyy%100!=0)&& (mm==2))||((yyyy%4==0) && (yyyy%100!=0) && (yyyy%400==0) && (mm==2)))

return 29-dd; else return 28-dd;

break;

}

default: return 0; break;

}

}

void DataTimes::Print() {

printf("%02d.%02d.%04d\n",dd,mm,yyyy);

}

bool DataTimes::operator&(DataTimes ob) {

if((ob.dd==dd)&&(ob.mm==mm)&&(ob.yyyy==yyyy)) return 1; else return 0;

}

bool DataTimes::operator!(){

if(dd<28) return true; else

{

if((mm==1)||(mm==3)||(mm==5)||(mm==7)||(mm==8)||(mm==10)||(mm==12)) {

if(dd==31) return false; else return true;

} else

{

if((mm==4)||(mm==6)||(mm==9)||(mm==11))

{

if(dd==30) return false; else return true;

} else

{

if(((yyyy%4==0) && (yyyy%100!=0)&& (mm==2))||((yyyy%4==0) && (yyyy%100!=0) && (yyyy%400==0) && (mm==2)))

{

if(dd==29) return false; else return true;

} else if(dd==28) return false; else return true;

}

}

}

}

void DataTimes::plusDay(int count) {

dd=dd+count;

while(!this->DataValid())

{

if((mm==1)||(mm==3)||(mm==5)||(mm==7)||(mm==8)||(mm==10)||(mm==12))

{

if(dd>31) {dd-=31;mm++;}

} else

{

if((mm==4)||(mm==6)||(mm==9)||(mm==11))

{

if(dd>30) {dd-=30;mm++;}

} else

{

if(((yyyy%4==0) && (yyyy%100!=0))||((yyyy%4==0) && (yyyy%100!=0) && (yyyy%400==0)))

{

if(dd>29)

{

dd-=29;mm++;

}

} else

{

if(dd>28)

{

dd-=28;mm++;

}

}

}

}

if(mm>12) {

mm=1;

yyyy++;

}

}

}

bool DataTimes::DataValid(){

if((mm==1)||(mm==3)||(mm==5)||(mm==7)||(mm==8)||(mm==10)||(mm==12))

{

if(dd>31) return false;

} else

{

if((mm==4)||(mm==6)||(mm==9)||(mm==11))

{

if(dd>30) return false;

} else

{

if(((yyyy%4==0) && (yyyy%100!=0))||((yyyy%4==0) && (yyyy%100!=0) && (yyyy%400==0)))

{

if(dd>29)

{

return false;

}

} else

{

if(dd>28)

{

return false;

} else return true;

}

}

}

}

**Project\_1.cpp**

// Project\_1.cpp: определяет точку входа для консольного приложения.

//

#include "stdafx.h"

#include "PZ.h"

#include "Free.h"

#include "Shareware.h"

#include "DataTimes.h"

#include "Busines.h"

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

{

/\*Free ob1;

ob1.Input();

ob1.show();

ob1.use();

Shareware ob2;

ob2.Input();

ob2.show();

ob2.use();

system("pause");

Busines ob3;

ob3.Input();

ob3.show();

ob3.use(); \*/

PZ \*\*pp;

pp = new PZ\*[5];

pp[0]= new Free();

pp[1]= new Shareware();

pp[2]= new Busines();

for(int i=0;i<3;i++) pp[i]->Input();

for(int i=0;i<3;i++) pp[i]->show();

for(int i=0;i<3;i++) pp[i]->use();

system("pause");

return 0;

}

**Варіант 1**

1. Створити абстрактний клас Figure з методами обчислення площі і периметра, а також методом, що виводить інформацію про фігуру на екран.
2. Створити похідні класи: Rectangle (прямокутник), Circle (круг), Triangle (трикутник) з своїми методами обчислення площі і периметра.
3. Створити масив n фігур і вивести повну інформацію про фігури на екран.

**Варіант 2**

1. Створити абстрактний клас Function з методом обчислення значення функції y=f(x) в заданій точці
2. Створити похідні класи: Line (y=ax+b), Kub (y=ax2+bx+c), Hyperbola з своїми методами обчислення значення в заданій точці.
3. Створити масив n функцій і вивести повну інформацію про значення даних функцій в точці х.

**Варіант 3**

1. Створити абстрактний клас Видання з методами дозволяючими вивести на екран інформацію про видання, а також визначити чи є дане видання шуканим.
2. Створити похідні класи: Книга (назва, прізвище автора, рік видання, видавництво), Стаття (назва, прізвище автора, назва журналу, його номер і рік видання), Електронний ресурс (назва, прізвище автора, посилання, анотація) з своїми методами виведення інформації на екран.
3. Створити каталог (масив) з n видань, вивести повну інформацію з каталога, а також організувати пошук видань по прізвищу автора.

**Варіант 4**

1. Створити абстрактний клас Trans з методами дозволяючими вивести на екран інформацію про транспортний засіб, а також визначити вантажопідйомність транспортного засобу.
2. Створити похідні класи: Легковая\_машина (марка, номер, швидкість, вантажопідйомність), Мотоцикл (марка, номер, швидкість, вантажопідйомність, наявність коляски, при цьому якщо коляска відсутня, то вантажопідйомність рівна 0), Грузовик (марка, номер, швидкість, вантажопідйомність, наявність причепа, при цьому якщо є причіп, то вантажопідйомність збільшується в два рази) з своїми методами виведення інформації на екран, і визначення вантажопідйомності.
3. Створити базу (масив) з n машин, вивести повну інформацію з бази на екран, а також організувати пошук машин, що задовольняють вимогам вантажопідйомності.

**Варіант 5**

1. Створити абстрактний клас Persona з методами, дозволяючим вивести на екран інформацію про персону, а також визначити її вік (на момент поточної дати).
2. Створити похідні класи: Абітурієнт (прізвище, дата народження, спеціальність), Студент (прізвище, дата народження, ВІДДІЛЕННЯ, курс), Викладач (прізвище, дата народження, посада, стаж), з своїми методами виведення інформації на екран, і визначення віку.
3. Створити базу (масив) з n персон, вивести повну інформацію з бази на екран, а також організувати пошук персон, чий вік потрапляє в заданий діапазон.

**Варіант 6**

1. Створити абстрактний клас Товар з методами, дозволяючим вивести на екран інформацію про товар, а також визначити, чи відповідає вона терміну придатності на поточну дату.
2. Створити похідні класи: Продукт (назва, ціна, дата виробництва, термін придатності), Партія (назва, ціна, кількість шт, дата виробництва, термін придатності), Комплект (назви, ціна, перелік продуктів) з своїми методами виведення інформації на екран, і визначення відповідності терміну придатності.
3. Створити базу (масив) з n товарів, вивести повну інформацію з бази на екран, а також організувати пошук простроченого товару (на момент поточної дати).

**Варіант 7**

1. Створити абстрактний клас Товар з методами, що дозволяють вивести на екран інформацію про товар, а також визначити, чи відповідає вона шуканому типу.
2. Створити похідні класи: Іграшка (назва, ціна, виробник, матеріал, вік, на який розрахована), Книга (назва, автор, ціна, видавництво, вік, на який розрахована), Спорт-інвентар (назва, ціна, виробник, вік, на який розрахована), з своїми методами виведення інформації на екран, і визначення відповідності шуканому типу.
3. Створити базу (масив) з n товарів, вивести повну інформацію з бази на екран, а також організувати пошук товарів певного типу.

**Варіант 8**

1. Створити абстрактний клас Телефонний\_довідник з методами, що дозволяють вивести на екран інформацію про записи в телефонному довіднику, а також визначити відповідність запису критерію пошуку.
2. Створити похідні класи: Персона (прізвище, адреса, номер телефону), Організація (назва, адреса, телефон, факс, контактна особа), Друг (прізвище, адреса, номер телефону, дата народження) з своїми методами виведення інформації на екран, і визначення відповідності шуканому типу.
3. Створити базу (масив) з n товарів, вивести повну інформацію з бази на екран, а також організувати пошук в базі по прізвищу.

**Варіант 9**

1. Створити абстрактний клас Клієнт з методами, що дозволяють вивести на екран інформацію про клієнтів банку, а також визначити відповідність клієнта критерію пошуку.
2. Створити похідні класи: Вкладник (прізвище, дата відкриття внеску, розмір внеску, відсоток по внеску), Кредитор (прізвище, дата видачі кредиту, розмір кредиту, відсоток по кредиту, залишок боргу), Організація (назва, дата відкриття рахунку, номер рахунку, сума на рахунку) з своїми методами виведення інформації на екран, і визначення відповідності даті (відкриття внеску, видачі кредиту, відкриття рахунку).
3. Створити базу (масив) з n клієнтів, вивести повну інформацію з бази на екран, а також організувати пошук клієнтів, що почали співробітничати з банком в задану дату.

**Варіант 10**

1. Створити абстрактний клас Программне\_забезпечення з методами, що дозволяють вивести на екран інформацію про програмне забезпечення, а також визначити відповідність можливості використовування (на момент поточної дати).
2. Створити похідні класи: Вільне (назва, виробник), Умовно-безкоштовне (назва, виробник, дата установки, термін безкоштовного використовування), Комерційне (назва, виробник, ціна, дата установки, термін використовування) з своїми методами виведення інформації на екран, і визначення можливості використовування на поточну дату.
3. Створити базу (масив) з n видів програмного забезпечення, вивести повну інформацію з бази на екран, а також організувати пошук програмного забезпечення, яке допустимо використовувати на поточну дату.

**Приклад коду в консольному режимі VS С++2008**

**ПРИКЛАД 1 Реалізація простого (одиничного) однорівневого успадкування.**

// p1.cpp : Defines the entry point for the console application.

#include "stdafx.h"

#include <iostream>

#include <conio.h>

using namespace std;

class B {

int b;

public:

void setb(int n){b=n;}

void showb(){cout<<"in B b="<<b<<endl;}

};

class D: public B {

int d;

public:

void setd(int n){d=n;}

void showd(){cout<<"in D d="<<d<<endl;}

};

//void main()

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

{ D qq; //объявление объекта порожденного класса

qq.setb(1); //доступ к члену базового класса qq.x

qq.showb(); //доступ к члену базового класса

qq.setd(2); //доступ к члену производного класса qq.y

qq.showd(); //доступ к члену производного класса

qq.showb(); //доступ к члену базового класса

getch();

return 0;

}

//=== Результат работы ===

in B b=1 //qq.x

in D d=2 //qq.y

in B b=1 //qq.x

**2. Завдання для самостійниго виконання**

**Приклад виконаного ззавдання**

**Варіант 11.**

**tz.h**

#pragma once

#include <string>

#include <iostream>

#include <cstdlib>

using namespace std;

class tz

{

public:

std::string name,mark,model;

void show();

void input();

tz(std::string n,std::string ma,std::string mo);

tz(void);

~tz(void);

};

**tz.cpp**

#include "StdAfx.h"

#include "tz.h"

#include <string>

#include <iostream>

#include <cstdlib>

using namespace std;

tz::tz(void)

{

}

tz::tz(string na,string ma,string mo){

name=na;

mark=ma;

model=mo;

}

void tz::input(){

setlocale(LC\_ALL,"rus");

cout<<"Введите название транспортного средства:";

cin>>name;

cout<<"Введите марку транспортного средства:";

cin>>mark;

cout<<"Введите модель транспортного средства:";

cin>>model;

}

void tz::show(){

cout<<"название транспортного средства:"<<name<<endl;

cout<<"марка транспортного средства:"<<mark<<endl;

cout<<"модель транспортного средства:"<<model<<endl;

}

tz::~tz(void)

{

}

**train.h**

#pragma once

#include "tz.h"

#include <string>

#include <iostream>

#include <cstdlib>

using namespace std;

class train :

public tz

{

public:

string gosnum,poch,kin;

int klvag;

int godPuti;

train(string num,string poch1,string kin1,int kl,int god,std::string na,std::string ma,std::string mo);

void input();

void show();

train(void);

~train(void);

};

**train.cpp**

#include "StdAfx.h"

#include "train.h"

#include <iostream>

#include <cstdlib>

using namespace std;

train::train(string num,string poch1,string kin1,int kl,int god,std::string na,std::string ma,std::string mo) {

gosnum=num;

poch=poch1;

kin=kin1;

klvag=kl;

godPuti=god;

name=na;

mark=ma;

model=mo;

}

void train::input() {

setlocale(LC\_ALL,"rus");

cout<<"Введите название поезда:";

cin>>name;

cout<<"Введите марку поезда:";

cin>>mark;

cout<<"Введите модель поезда:";

cin>>model;

cout<<"Введите госномер поезда:";

cin>>gosnum;

cout<<"Введите начальную:";

cin>>poch;

cout<<"Введите конечную:";

cin>>kin;

cout<<"Введите количество вагонов:";

cin>>klvag;

cout<<"Введите часов в пути:";

cin>>godPuti;

}

void train::show() {

setlocale(LC\_ALL,"rus");

cout<<"название поезда:"<<name<<endl;

cout<<"маркa поезда:"<<mark<<endl;

cout<<"модель поезда:"<<model<<endl;

cout<<"госномер поезда:"<<gosnum<<endl;

cout<<"начальная:"<<poch<<endl;

cout<<"конечная"<<kin<<endl;

cout<<"количество вагонов:"<<klvag<<endl;

cout<<"chасов в пути:"<<godPuti<<endl;

}

train::train(void)

{

}

train::~train(void)

{

}

**expess.h**

#pragma once

#include "train.h"

#include <iostream>

#include <cstdlib>

using namespace std;

class expess :

public train

{

public:

int cina;

void input();

void show();

expess(string num,string poch1,string kin1,int kl,int god,int c,std::string na,std::string ma,std::string mo);

expess(void);

~expess(void);

};

**expess.cpp**

#include "StdAfx.h"

#include "expess.h"

#include <iostream>

#include <cstdlib>

using namespace std;

expess::expess(string num,string poch1,string kin1,int kl,int god,int c,std::string na,std::string ma,std::string mo){

gosnum=num;

poch=poch1;

kin=kin1;

klvag=kl;

godPuti=god;

name=na;

mark=ma;

model=mo;

cina=c;

}

void expess::input(){

setlocale(LC\_ALL,"rus");

cout<<"Введите название експреса:";

cin>>name;

cout<<"Введите марку поезда:";

cin>>mark;

cout<<"Введите модель поезда:";

cin>>model;

cout<<"Введите госномер поезда:";

cin>>gosnum;

cout<<"Введите начальную:";

cin>>poch;

cout<<"Введите конечную:";

cin>>kin;

cout<<"Введите количество вагонов:";

cin>>klvag;

cout<<"Введите часов в пути:";

cin>>godPuti;

cout<<"Введите цену билета експреса";

cin>>cina;

}

void expess::show() {

setlocale(LC\_ALL,"rus");

cout<<"название поезда:"<<name<<endl;

cout<<"маркa поезда:"<<mark<<endl;

cout<<"модель поезда:"<<model<<endl;

cout<<"госномер поезда:"<<gosnum<<endl;

cout<<"начальная:"<<poch<<endl;

cout<<"конечная"<<kin<<endl;

cout<<"количество вагонов:"<<klvag<<endl;

cout<<"chасов в пути:"<<godPuti<<endl;

cout<<"цена билета експреса"<<cina<<endl;

}

expess::expess(void)

{

}

expess::~expess(void)

{

}

**avto.h**

#pragma once

#include "tz.h"

#include <string>

#include <iostream>

#include <cstdlib>

using namespace std;

class avto :

public tz

{

public:

string zvet,gosnumb;

int vaga;

avto(std::string n,std::string ma,std::string mo,string zv, string gosn);

avto(void);

void input();

void show();

~avto(void);

};

**avto.cpp**

#include "StdAfx.h"

#include "avto.h"

#include <iostream>

#include <cstdlib>

using namespace std;

avto::avto(std::string n,std::string ma,std::string mo,string zv, string gosn) {

name=n;

mark=ma;

model=mo;

zvet=zv;

gosnumb=gosn;

}

void avto::input()

{

setlocale(LC\_ALL,"rus");

cout<<"Введите название автомобиля:";

cin>>name;

cout<<"Введите марку автомобиля:";

cin>>mark;

cout<<"Введите модель автомобиля:";

cin>>model;

cout<<"Введите цвет автомобиля:";

cin>>zvet;

cout<<"Введите государствений автомобиля:";

cin>>gosnumb;

}

void avto::show() {

cout<<"название автомобиля:"<<name<<endl;

cout<<"марка автомобиля:"<<mark<<endl;

cout<<"модель автомобиля:"<<model<<endl;

cout<<"цвет автомобиля:"<<zvet<<endl;

cout<<"государствений автомобиля:"<<gosnumb<<endl;

}

avto::avto(void)

{

}

avto::~avto(void)

{

}

**Project\_2.cpp**

// Project\_2.cpp: определяет точку входа для консольного приложения.

//

#include "stdafx.h"

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

{

return 0;

}

**Задача 2**

**Додаток. Варіанти завдань**

*Дано перелік класів. Побудуйте обєктну модель предметної області, враховуючи, види залежностей між класами. Реалізуйте проект за ООП.*

**1.** **студент, викладач, персона, зав. кафедрою**

**2. службовець, персона, робітник, інженер**

**3. робітник, кадри, інженер, адміністрація**

**4. деталь, механізм, виріб, вузол**

**5. організація, страхова компанія, суднобудівна компанія, завод**

**6. журнал, книга, друковане видання, підручник**

**7. тест, іспит, випускний іспит, випробовування**

**8. місце, область, місто, мегаполіс**

**9. іграшка, продукт, товар, молочний продукт**

**10. квитанція, накладна, документ, чек**

**11. автомобіль, потяг, транспортний засіб, експрес**

**12. двигун, двигун внутрішнього згоряння, дизель, турбореактивний двигун**

**13. республіка, монархія, королівство, держава**

**14. ссавці, парнокопитні, птаха, тварина**

**15. корабель, пароплав, вітрильник, корвет**

**Варіанти завдань на мові С#**

***Завдання 3.***  Створити базовий клас для обробки записів бази даних у відповід­ності з вказаною предметною областю. Розробити програму, дотримуючись таких вимог:

1. Розробити інтерфейс класу, та визначення функцій,головну функцію програми.
2. Передбачити можливість роботи з довільним числом записів, а також реалізувати окремими функціями класу:
   * конструктори без параметрів та з параметрами;
   * додавання об’єктів;
   * виведення інформації на екран;
   * пошук потрібної інформації за конкретною ознакою;
   * редагування записів;
   * сортування за різними полями.
3. При розробці програми слід здійснити захищення даних (опис з моди­фікатором *private*) для ізоляції елементів-даних класу від підпрограм, в яких цей клас використовується.
4. Програма повинна містити меню для перевірки всіх методів класу. Бажано для реалізації меню розробити окрему функцію, яка повертає номер вибраного пункту меню.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Предметна область** | **Реквізити об’єкту** | **Параметр сортування** | **Параметр пошуку** |
| 1. | Бібліотека | інвентарний номер, автор, назва, кількість сторінок, рік видання | Рік видання | Автор |
| 2. | Телефонний довідник | Прізвище, ім'я, по батькові, домашня адреса, телефон. | Телефон | Прізвище |
| 3. | Розклад руху літаків | Номер рейсу, тип літака, напрямок руху, періодичність вильоту. | Номер рейсу | Тип літака |
| 4. | Колекція компакт-дисків | Інвентарний номер, назва альбому, об'єм диску, тип, дата запису. | Дата запису | Назва альбому |
| 5. | Записна книжка | Прізвище, ім'я, домашня адреса, телефон, електронна пошта. | Прізвище | Електронна пошта |
| 6. | Предметний покажчик | Слово; номера сторінок, де це слово зустрічається; кількість цих слів на даній сторінці | Номер сторінки | Слово |
| 7. | Розклад пар | Номер пари, предмет, прізвище викладача, форма заняття. | Предмет | Номер пари |
| 8. | Список файлів | ім'я файла, розширення, розмір, дата створення, атрибути. | Розширен­ня | Дата створення |
| 9. | Архів програм | Назва програми, операційна система, розмір програми, дата запису | Назва програми | Операційна система |
| 10. | Рахунки банку | Прізвище, ім'я, дата останньої операції, сума вкладу | Сума вкладу | Дата операції |
| 11. | Користувачі локальної мережі | Прізвище, група, обліковий запис, тип облікового запису. | Тип облікового запису | Прізвище |
| 12. | Камера схову | Прізвище, дата здачі, термін зберігання, інвентарний номер та назва предмета | Інвентар­ний номер | Дата здачі |
| 13. | Склад товарів | інвентарний номер, назва товару, вага, ціна, кількість | Вага | Назва товару |
| 14. | Каса продажу квитків | Назва пункту, час відправлення, дата відправлення, час прибуття, дата прибуття, ціна квитка | Час від­правлення | Назва пункту |
| 15. | Успішність студентів | Прізвище, номер групи, оцінки з трьох предметів | Прізвище | Номер групи |

***Завдання4.*** Утворити похідний клас, залучивши до нього як мінімум два додаткових поля таким чином, щоб клас набув більшої спеціалізованості. Для похідного класу використати кон­структор, щоб він містив усі аргументи, необхідні для ініціалізації об'єкту похідного класу. Створити додаткові необхідні функції, що дозволяють перевірити роботу похідних класів.

**Контрольні питання**

1. Що таке об’єктно-орієнтоване програмування?
2. Яка різниця між об'єктно-орієнтованним та функціональним програму­ванням?
3. Що таке класи? Який синтаксис їх опису?
4. Що таке об’єкти та екземпляри класу?
5. Які основні принципи в об’єктно-орієнтованому програмуванні? Назвіть їх і дайте коротку характеристику сутностей цих принципів.
6. Наведіть приклад опису будь-якого класу та приклад створення екземпляру цього класу.
7. Для чого потрібні *set-* і *get-*функції?
8. Які специфікатори доступу використовуються в описах класів? Поясніть їх значення.
9. Дайте поняття закритих полів і методів класу.
10. Що таке конструктори і деструктори класу? Для чого їх викорис­то­вують?
11. Які особливості конструкторів у класах?
12. Які особливості деструкторів у класах?
13. Поясніть на прикладах сутність успадкування класів.
14. Наведіть синтаксис опису успадкованого класу. Поясніть на прикладах.
15. Що Ви можете сказати про модифікатори доступу в успадкованих класах?
16. Як успадковуються конструктори у похідних класах?
17. Які аиди успадкування Ви знєте? На ведіть приклади.
18. Що таке віртуальні функції? Наведіть приклади використання.
19. Що таке дружні функції і дружні класи?

**Лабораторна робота № 4 Побудова множинного наслідування класів.(С++ ) Інтерфейси- С#**

*Мета:* навчитися організовувати класи та похідні класи з своїми даними та властивостями: визначати структуру класу, типи даних та методів, організовувати необхідні конструктори та деструктори, використовувати специфікатори доступу; будувати ієрархію класів, використовуючи множинне наслідування; визначати та оперувати об’єктами цих класів; отримати практичні вміння та навички проектування та побудови ієрархії класів. Отримати практичні навички використання інтерфейсів.  
  
*Обладнання:* ПК, програмне забезпечення Visual Studio 2008 (10,12,13)C++, методичні вказівки та завдання до лабораторної роботи.  
  
 **Короткі теоретичні зведення**  
  
Принципи множинного наслідування. В С++ клас може мати не лише один батьківський клас. Якщо характеристики наслідуються від декількох батьківських класів, то говорять про множинне наслідування. Визначальним для похідного класу породженого множинним наслідуванням є те, що він явно чи неявно повинен успадковувати характеристики декількох базових класів. Неявним множинним наслідуванням можуть бути випадки змішаного наслідування. Як приклад приведемо ситуацію, коли похідний клас породжений від проміжного (просте наслідування), який, у свою чергу, породжений від двох базових (множинне наслідування). Результатом цих наслідувань є ієрархія, в якій похідний клас неявно (через один проміжний) успадкував характеристики двох базових класів.  
  
Якщо похідний клас породжується від декількох базових, то в декларації класу треба вказати усі базові класи, розділяючи їх комою, разом зі специфікаторами доступу до них. Загальний синтаксис множинного наслідування має вигляд  
  
class : [<модифікатор>] , ..., [<модифікатор>] { <тіло класу> };  
  
Наприклад  
  
class A { <тіло класу A> };  
  
class B { <тіло класу B> };  
  
class C : public A, B { }; // клас C породжений від A i B  
  
Основні принципи простого наслідування, зокрема наслідування членів, модифікатори доступу до членів базових класів, розширення та обмеження характеристик, без жодних доповнень можуть бути перенесені на наслідування множинне. Тому відразу наведемо приклад програми, яка ілюструє множинне наслідування  
  
Приклад  
  
#include   
  
output (char\* c) { std::cout<  
  
class A { public:  
  
show () { output ("Клас A "); }  
  
};  
  
class B { public:  
  
show () { output("Клас B "); }  
  
};  
  
class C : public A, public B // клас C породжений від A i B  
  
{  
  
public:  
  
void show () { A::show (); B::show (); output ("Клас C"); }  
  
};  
  
main()  
  
{ C c;  
  
c.show ();  
  
}  
  
В наведеній програмі використана технологія перевизначення функцій при наслідуванні. Як метод доступу до функцій-членів перевизначених в множинній ієрархії використано ОДД з іменем класу.  
  
**^ Констуктори та деструктори при множинному наслідуванні.** Виклики і виконання конструкторів та деструкторів при множинному наслідуванні є подібними до випадку одинарного наслідування. Якщо множинне наслідування розглядати як суму простих в порядку декларацій батьківських класів в оголошенні похідного класу, то легко зрозуміти черговість виклику і виконання функцій побудови і знищення об'єктів похідного класу.  
  
При створенні об’єкта класу множинно породженого після виклику конструктора похідного класу викликатиметься конструктор найпершого батьківського класу. Якщо він є похідним від ще одного класу, то викликатиметься і буде виконуватись конструктор останнього. По закінченню роботи усіх конструкторів по вітці дерева від найпершого класу, розпочне виконуватись вітка від другого батьківського класу і т.д. Після того, як відпрацюють конструктори усіх батьківських класів, виконається тіло конструктора похідного класу.  
  
Порядок виклику деструкторів є таким як у конструкторів, а виконання – зворотнім. Найпершим почне виконуватись деструктор похідного класу, а далі – деструктори вітки породженої останнім в оголошенні батьківським класом. У порядку зворотньому до декларації батьківських класів відпрацють деструктори класів усіх віток від них породжених. Лише по закінченню роботи і закритття батьківських деструкторів закриється деструктор похідного класу. Порядок роботи конструкторів та деструкторів ілюструє наступний приклад.  
  
Приклад  
  
#include   
  
using namespace std;  
  
Show (const char\* mes, const char a=' ') { cout<<MES<<A<<"\N"; }  
  
const char\* mes\_cosntr = "Конструктор класу";  
  
const char\* mes\_destr = "Деструктор класу";  
  
class D { public:  
  
char d\_pub;  
  
D () { // виконується першим  
  
Show ("Порядок виконання конструкторів");  
  
d\_pub = 'D'; Show (mes\_cosntr,d\_pub);  
  
}  
  
~D() { Show (mes\_destr, d\_pub); } // виконується останнім  
  
};  
  
class A : public D { public:  
  
char a\_pub;  
  
A (char c) { a\_pub = c; Show (mes\_cosntr, a\_pub); }  
  
~A() { Show (mes\_destr,a\_pub); }  
  
};  
  
class B { public:  
  
char b\_pub;  
  
~B() { Show (mes\_destr,b\_pub); }  
  
B (char c) { b\_pub = c; Show (mes\_cosntr, b\_pub); }  
  
};  
  
class C : public A, public B { public:  
  
char c\_pub;  
  
C (char c1, char c2, char c3 ) : B ( c2 ), A ( c3 )  
  
{ // виконується останнім  
  
c\_pub = c1; Show (mes\_cosntr, c\_pub);  
  
}  
  
~C() // виконується першим  
  
{ Show("Порядок виконання деструкторів");  
  
Show(mes\_destr,c\_pub);  
  
}  
  
};  
  
Show (C c) { cout<<"\n"; }  
  
main()  
  
{ C\* c = new C ('C', 'B', 'A');  
  
Show (\*c);  
  
delete c;  
  
}  
  
Механізм передаваня аргументів конструкторам базових класів повністю підтримується при множинному наслідуванні. Так у коді  
  
C :: C (char c1, char c2, char c3) : B (c2), A(c3) { c\_pub=c1; Show (mes\_cosntr, c\_pub); }  
  
аргументи с2 і с3 передаються конструкторам базового класу В і проміжного класу А відповідно. Порядок передавання аргументів конструкторам в оголошенні конструктора похідного класу може бути довільним, оскільки виклики і виконання конструкторів визначаються порядком наслідування в оголошенні класу.  
  
Як у випадку простого наслідування при визначенні конструктора можна передавати аргументи лише конструкторам вищих сусідніх класів. Конструктором класів, які не є сусідніми в ієрархії, аргументи передаються лише через визначення конструкторів проміжних класів. Наприклад, наступне визначення конструктора є неправильним  
  
C :: C (char c1, char c2, char c3) : B (c2), A(c3), D()  
  
{ c\_pub=c1; Show (mes\_cosntr, c\_pub); }  
  
Завдання до лабораторної роботи № 4  
Необхідно побудувати ієрархію класів що відповідає схемі наслідування, наведеній у варіанті завдання. Кожен клас повинен містити конструктор-ініціалізатор, і функцію *show()* для виведення значень.

|  |  |  |
| --- | --- | --- |
| **Варіант 1.** | **Варіант 2.** | **Варіант 3.** |
| 12842_html_m1a3eb252 | 12842_html_m332080c8 | 12842_html_39cae608 |
| **Варіант 4.** | **Варіант 5.** | **Варіант 6.** |
| 12842_html_698847bd | 12842_html_ee86689 | 12842_html_e0f07e |
| **Варіант 7.** | **Варіант 8.** | **Варіант 9.** |
| 12842_html_59d0a0e5 | 12842_html_m7fa2f05e | 12842_html_17e05fd0 |
| **Варіант 10.** | **Варіант 11.** | **Варіант 12.** |
| 12842_html_m7738a28f | 12842_html_m443dbf0 | 12842_html_1051a827 |

**Приклади виконаних завдань**

**B1.h**

#pragma once

#include <iostream>

using namespace std;

class B1

{

public:

string n;

B1(void);

~B1(void);

};

**B1.cpp**

#include "StdAfx.h"

#include "B1.h"

#include <iostream>

using namespace std;

B1::B1(void)

{

cout<<"Class B1, predkiv nema\n";

}

B1::~B1(void)

{

}

**B2.h**

#pragma once

#include <iostream>

using namespace std;

class B2

{

public:

string n;

B2(void);

~B2(void);

};

**B2.cpp**

#include "StdAfx.h"

#include "B2.h"

#include <iostream>

using namespace std;

B2::B2(void)

{

cout<<"Class B2, predkiv nema\n";

}

B2::~B2(void)

{

}

**D1.h**

#pragma once

#include "B1.h"

#include "B2.h"

class D1:public B2,private B1

{

public:

D1(void);

~D1(void);

};

**D1.cpp**

#include "StdAfx.h"

#include "D1.h"

#include <iostream>

using namespace std;

D1::D1(void)

{

cout<<"Class D1, predok public B2, private B1\n";

}

D1::~D1(void)

{

}

**D2.h**

#pragma once

#include "B1.h"

#include "B2.h"

class D2:public B1,private B2

{

public:

D2(void);

~D2(void);

};

**D2.cpp**

#include "StdAfx.h"

#include "D2.h"

#include <iostream>

using namespace std;

D2::D2(void)

{

cout<<"Class D2, predok public B1, private B2\n";

}

D2::~D2(void)

{

}

**D3.h**

#pragma once

#include "D1.h"

#include "D2.h"

class D3:private D1, public D2

{

public:

D3(void);

~D3(void);

};

**D3.cpp**

#include "StdAfx.h"

#include "D3.h"

#include <iostream>

using namespace std;

D3::D3(void)

{

cout<<"Class D3, predok public D2, private D1\n";

}

D3::~D3(void)

{

}

Функція *main()* повинна ілюструвати ієрархію наслідування.

**Інтерфейс** - це контракт, що забезпечує певну поведінку класу або структури. Коли клас реалізує інтерфейс, він як би говорить потенційному користувачеві: "Я гарантую, що підтримую методи, властивості, події і індексатори цього інтерфейсу".

**Інтерфейс** - сімейство явно описаних як public - методів і властивостей, які згруповані в єдине ціле і інкапсулюють якусь певну функціональну можливість.

Після того як інтерфейс визначено, його можна реалізувати в деякому класі. Це означає, що клас буде підтримувати всі властивості і члени, які визначаються даним інтерфейсом. Інтерфейси не можуть існувати самі по собі. таким же чином, як створюється екземпляр класу. У інтерфейсу немає коду, який би реалізував його члени; він просто описує ці члени. Їх реалізація повинна перебувати в класах, в яких реалізований даний інтерфейс.

Отже, інтерфейс являє собою альтернативу абстрактного класу в сенсі створення контрактів між класами і їх користувачами. Ці контракти створюються з використанням ключового слова interface, яке оголошує контрольний тип, що інкапсулює контракт. Синтаксично інтерфейс подібний до класу, що має тільки абстрактні методи. Абстрактний клас є базовим для сімейства похідних класів, в той час як інтерфейси задумані для "змішування" їх з іншими деревами успадкування. Коли клас реалізує інтерфейс, він повинен реалізувати всі методи цього інтерфейсу. По суті справи, він погоджується виконати контракт, визначений інтерфейсом.

**Правила використання інтерфейсів:**

* Не можна "створити екземпляр інтерфейсу"
* Интерфейс не може містити в собі ніякого коду
* У класа може бути декілька інтерфейсів, і декілька класів можуть підтримувати один і той же інтерфейс.
* У інтерфейсів немає ніяких конструкторів
* Визначенню інтерфейса не дозволено містити перегрузки операцій
* Не допускаєтся вказувати будь-які модифікатори доступу, а також модифікатори virtual,static

Розглянемо інтерфейси за рахунок подання повного визначення одного з інтерфейсів, визначеного Microsoft - **System.IDisposable**. Інтерфейс IDisposable містить один метод Dispose (), призначений для деініціалізації класами, які виконують очистку коду:

public interface IDisposable

{

void Dispose ();

}

Цей фрагмент показує, що оголошення інтерфейсу синтаксично дуже схоже на оголошення абстрактного класу. Однак ви повинні пам'ятати, що не допускається вказувати ніякої реалізації ні для одного з членів інтерфейсу.

Інтерфейс може містити тільки оголошення методів, властивостей, індексів і подій. Він містить тільки сигнатури членів. Інтерфейс не має ніяких конструкторів, так само як і ніяких полів.

Визначенню інтерфейсу також заборонено містити перевантаження операцій.

Також не допускається вказувати модифікатори при визначенні членів інтерфейсу. Члени інтерфейсу завжди неявно є **public** і не можуть бути **virtual** або **static**.

Повернемося до визначення інтерфейсу IDisposable. Якщо клас оголошує, що він реалізує метод Dispose (), то він повинен буде реалізувати інтерфейс IDisposable, що в термінах С # означає, що він успадковується від IDisposable.

// Цей клас ПОВИНЕН містити реалізацію методу IDisposable.Dispose (), інакше-ERROR!

**class my: IDisposable**

{

public void Dispose () {// реалізація методу Dispose ()}

// решта класу ...

}

У цьому прикладі якщо клас my буде успадковувати IDisposable, але не буде містити реалізації Dispose (), в точності збігається з сигнатурою, визначеної в інтерфейсі IDisposable, буде видана помилка компіляції, оскільки в цьому випадку клас порушить договір продажу інтерфейсу.

IDisposable - порівняно простий інтерфейс, тому що визначає тільки один метод. Більшість інтерфейсів, то більша кількість методів.

Іншим хорошим прикладом інтерфейсу є той, який представляє цикл foreach в С #. В принципі цикл foreach всередині працює так, що опитує об'єкт на предмет того, чи реалізує він інтерфейс **System. Collection. IEnumerable**. Він працює так, то компілятор C # вставляє в IL-код, який використовує методи цього інтерфейсу для проходу по всім членам колекції. Якщо ж ні, foreach генерує виняток.

**Приклад опису, реалізації та використання інтерфейсу**

Приклад описує банківський рахунок. Припустимо, програмується код, який в кінцевому підсумку забезпечить комп'ютеризований переклад грошей між банківськими та особистими рахунками. Припустимо, що існує безліч компаній, які можуть peaлізовать банківські рахунки, але всі вони погодилися з тим, що будь-які класи, що представляють банківські рахунки, повинні реалізовувати інтерфейс IBankAccount, має методи для внесення і знімання грошових сум, а також властивість Balance-сума на рахунку . Для початку потрібно визначити інтерфейс **IBankAccount:**

**public interface IBankAccount**

**{**

**void PayIn (decimal amount); // додати гроші на рахунок**

**bool Withdraw (decimal amount); //зняти гроші з рахунку**

**decimal Balance // властивість баланс, пов'язане з даними balance- сума на рахунку**

**{**

**get; // У властивості описуються тільки блоки get set, але не реалізовуються!**

**}**

**}**

Ідея полягає в тому, що тепер ви маєте можливість писати класи, що представляють банківські рахунки. Ці класи не повинні бути якось пов'язаними один з одним; вони можуть бути абсолютно різними. Однак всі вони декларують своє уявлення банківських рахунків тим, що реалізують інтерфейс IBankAccount.

Припустимо, перший клас описує ощадний рахунок в Беларусбанке:

**public class BelarusAccount: IBankAccount // зверніть увагу на оголошення**

**{**

**private decimal balance; // дане класу - тільки в класі!**

**public void PayIn (decimal amount)**

**{**

**balance + = amount;**

**}**

**public bool Withdraw (decimal amount)**

**{**

**if (balance> = amount)**

**{Balance - = amount; return true; } else**

**Console.WriteLine ( "Спроба перекладу грошей не вдалася!");**

**return false;**

**}**

**public decimal Balance {**

**get**

**{**

**return balance;**

**}**

**}**

**public override string ToString ()**

**{**

**return String.Format ( "Беларусбанк: Баланс = {0,6: З}", balance);**

**}**

**}**

У класі створюється приватне поле balance і змінюється сума залишку, зазначена в ньому, при зніманні та зарахування грошей на рахунок. Якщо робиться спроба зняти більше грошей, ніж залишилося на рахунку, видається повідомлення про помилку. Відзначимо, що для простоти коду тут не реалізуються додаткові властивості, такі як ім'я власника рахунку та ін.

Щоб проілюструвати, як різні класи можуть реалізувати один і той же інтерфейс, припустимо, що Пріорбанк також реалізує власний клас, що представляє банківські рахунки - PriorAccount:

public class PriorAccount: IBankAccount {...}

// Реалізація основної програми

...

static void Main ()

{

**// зверніть увагу на оголошення примірників класів!**

**// Це посилання на інтерфейс !!!**

**IBankAccount Bel = new BelarusAccount ();**

**IBankAccount Prior = new PriorAccount ();**

Bel.PayIn (200);

Bel.Withdraw (100);

Console.WriteLine (Bel.ToString ());

Prior.PayIn (500);

Prior.Withdraw (600);

Prior.Withdraw (100);

Console.WriteLine (Prior.ToString ());

    }

Головний момент, який слід тут зазначити - спосіб оголошення примірників **як посилань** на IBankAccount. Це означає, що вони можуть вказувати на будь-який екземпляр будь-якого класу, що реалізовує інтерфейс. Це також означає, що через ці посилання ви можете викликати тільки ті методи, які є частиною інтерфейсу.

Якщо вам знадобиться викликати будь-які методи, реалізовані класом, але не є частиною інтерфейсу, то доведеться виконати **приведення посилання до відповідного типу**.

**Приклад:**

Припустимо, клас BelarusAccount має ще метод:

public void f () {...}

Якщо в основній програмі записати рядок:

**Bel.f (); // Компілятор видасть помилку !!!, так як Bel-посилання на інтерфейс!**

Для того, щоб викликати метод f () класу BelarusAccount необхідно привести змінну Bel до типу класу BelarusAccount через нову змінну х:

**BelarusAccount x = (BelarusAccount) Bel; // явне приведення типів**

Тепер можна викликати метод f (), використовуючи цю нову змінну:

x.f ();

Для методу ToString () (Не оголошений в інтерфейсі IBankAccount) не потрібно ніякого явного приведення типів, просто тому, що ToString () - це метод System.Object, тому компілятор С # знає про те, що він підтримується будь-яким класом.

Можна також створювати **масиви інтерфейсів**, елементи яких є екземплярами різних класів:

**IBankAccount** [] mas = new IBankAccount [2];

mas [0] = new BelarusAccount ();

mas [1] = new PriorAccount (); ...

Інтерфейси можуть бути успадкованими один від одного —так само, як це роблять класи.

Розглянемо застосування інтерфейсів на прикладі класу Monster. Нехай в класі Monster буде реалізований інтерфейс IAction, в якому зібрані всі дії, які може зробити монстр в грі.

namespace ConsoleApplication1

{

    interface IAction

    {

        void Draw (); // простий метод виведення імені монстра

        int Attack (int a); // реалізація атаки із заданою силою

        void Die (); // метод виведення на екран повідомлення про смерть монстра

        int Power {get; } // властивість, обчислює силу монстра

     }

    class Monster: IAction

    {

        public Monster ()

        {

            this.health = 100;

            this.ammo = 100;

            this.name = "Noname";

        }

        public Monster (string name)

            : This ()

        {

            this.name = name;

        }

        public Monster (int health, int ammo, string name)

        {

            this.health = health;

            this.ammo = ammo;

            this.name = name;

        }

        public int Health

        {

            get

            {

                return health;

            }

            set

            {

                if (value> 0) health = value;

                else health = 0;

            }

        }

        public int Ammo

        {

            get

            {

                return ammo;

            }

            set

            {

                if (value> 0) ammo = value;

                else ammo = 0;

            }

        }

        public string Name

        {

            get

            {

                return name;

            }

        }

        public void Passport ()

        {

            Console.WriteLine ( "Monster {0} \ t health = {1} ammo = {2}", name, health, ammo);

        }

// ДАЛІ РЕАЛІЗУЮТЬСЯ ВСІ МЕТОДИ ІНТЕРФЕЙСУ

         public void Draw ()

        {Console.WriteLine ( "Тут був монстр" + name); }

        public int Attack (int a)

        {

            ammo - = a;

            if (ammo> 0) Console.WriteLine ( "ба-бах!");

            else ammo = 0;

            return ammo;

         }

        public void Die ()

        {Console.WriteLine ( "Монстр" + name + "помер"); health = 0; }

        public int Power

        {

            get {return ammo \* health; }

        }

        string name;

        int health, ammo;

    }

    class Program

    {

        static void Main (string [] args)

        {

            Monster Вася = new Monster (50, 50, "Вася як клас");

            Вася.Draw ();

            IAction IВася = new Monster (10, 10, "Вася як інтерфейс");

            IВася.Draw ();

            Вася.Passport ();

            ((Monster) IВася) .Passport ();

            IВася.Attack (10);

            Вася.Attack (20);

            Console.WriteLine ( "У монстра" + Вася.Name + "сила дорівнює" + Convert.ToString (((IAction) Вася) .Power));

            Console.WriteLine ( "У монстра" + ((Monster) IВася) .Name + "сила дорівнює" + Convert.ToString (IВася.Power));

            if (Вася.Ammo <= 0) Вася.Die ();

            if (((Monster) IВася) .Ammo <= 0) IВася.Die ();

            Console.ReadLine ();

        }

    }

}

|  |
| --- |
| **Контрольні питання**   1. Що таке успадкування? 2. Описати правила доступу для різних типів успадкування. 3. Чим відрізняються два види ієрархії? Навести приклади із свого варіанта. 4. Наведіть приклади множинного успадкування. Поясніть порядок виклику конструкторів. 5. Що таке статичні члени класу і в яких випадках їх зручно використовувати? |

## Лабораторна робота №5 Ієрархія об'єктів і групи. Агрегація.Ітератори.

**Мета.** Одержати практичні навички створення об'єктів-груп (агрегація, композиція) і використання методів-ітераторів.

*Обладнання:* ПК, програмне забезпечення Visual Studio 2008 (10,12,13)C++, методичні вказівки та завдання до лабораторної роботи.

**Основні теоретичні відомості.**

**Група.**

Група — це об'єкт, у який включені інші об'єкти. Об'єкти, що входять у групу, називаються *елементами групи*. Елементи групи, у свою чергу, можуть бути групою.

Приклади груп:

1)Вікно в інтерактивній програмі, що володіє такими елементами, як поля вводу і редагування даних, кнопки, списки вибору, діалогові вікна і т.д. Прикладами таких вікон є об'єкти класів, породжених від абстрактного класу *TGroup (TDeskTop, TWindow, TDialog*) в ієрархії класів бібліотеки **Turbo Vision** і об'єкти класів, породжених від *TWindowObject* в ієрархії класів бібліотеки **OWL.**

2)Агрегат, що складається з більш дрібних вузлів.

3)Город, що складається з рослин, системи поливу і плану вирощування.

4)Деяка організаційна структура (наприклад, ВІДДІЛЕННЯ, КАФЕДРА, СТУДЕНТСЬКА ГРУПА).

Ми відрізняємо “групу” від “контейнера”. Контейнер використовується для збереження інших даних. Приклади контейнерів: об'єкти класу *TCollection* бібліотеки і об'єкти контейнерних класів бібліотеки **STL** у **C++** (масиви, списки, черги).

На відміну від контейнера ми розуміємо групу, як клас, що не тільки зберігає об'єкти інших класів, але і має власні властивості, що не випливають із властивостей його елементів.

Група дає другий вид ієрархії (перший вид — *ієрархія класів*, побудована на основі успадкування) — *ієрархію об'єктів* (ієрархію типу *ціле/частина*), побудовану на основі агрегування ( *агрегація*).

Реалізувати групу можна декількома способами.

1.Клас “група” містить поля даних об'єктного типу. Таким чином, об'єкт “група” як дані містить або безпосередньо свої елементи, або покажчики на них

*class TWindowDialog: public TGroup*

*{*

*protected:*

*TInputLine input1;*

*TEdit edit1;*

*TButton button1;*

*/\*інші члени класу\*/*

};

Такий спосіб реалізації групи використовується в **C++Builder.**

2.Група містить член-дане *last* типу *TObject\**, що вказує на початок зв'язаного списку об'єктів, включених у групу. У цьому випадку об'єкти повинні мати поле *next* типу *TObject\*,* що вказує на наступний елемент у списку. Такий спосіб використовується при реалізації груп у **Turbo Vision.**

3.Створюється зв'язаний список структур типу *TItem:*

*struct TItem*

*{TObject\* item;*

*TItem\* next;};*

Поле *item* указує на об'єкт, включений у групу. Група містить поле *last* типу *TItem \**, що вказує на початокзв'язаного списку структур типу *TItem.*

Якщо необхідний доступ елементів групи до її полів і методів, об'єкт типу *TObject* повинний мати поле *owner* типу *TGroup\**, що вказує на власника цього елемента.

Методи групи.

Є два методи, що необхідні для функціонуваннягрупи:

1)*void Insert(TObject\* p);*

Вставляє елемент у групу.

2)*void Show();*

Дозволяє переглянути групу.

Крім цього група може містити наступні методи:

1) *int Empty();*

Показує, чи є хоча б один елемент у групі.

2) *TObject\* Delete(TObject\* p);*

Видаляє елемент із групи, але зберігає його в пам'яті.

3)*void DelDisp(TObject\* p);*

Видаляє елемент із групи і з пам'яті.

**Ієрархія об'єктів**

**Ієрархія класів** є ієрархія за принципом успадкування, тобто типу "це є різновид того". Наприклад, "робітник є різновид персони", “автомобіль” є різновид “транспортного засобу”. На відміну від цього **ієрархія об'єктів —** це ієрархія за принципом входження, тобто типу "це є частиною того". Наприклад, "установа - частина заводу", “двигун” частина “автомобіля”. Таким чином об'єкти нижнього рівня ієрархії включаються в об'єкти більш високого рівня, що є для них групою.

**Ітератор.**

Ітератори дозволяють виконувати деякі дії для кожного елемента визначеного набору даних.

***For all елементів набору { дії}***

Такий цикл міг би бути виконаний для всього набору, наприклад, щоб надрукувати всі елементи набору. Чи міг би шукати деякий елемент, що задовольняє визначеній умові, і в цьому випадку такий цикл може закінчитися, як тільки буде знайдений необхідний елемент.

Ми будемо розглядати ітератори як спеціальні методи класу-групи, що дозволяють виконувати деякі дії для всіх об'єктів, включених у групу. Прикладом ітератора є метод *Show.*

Нам би хотілося мати такий ітератор, що дозволяв би виконувати над всіма елементами групи дії, задані не одним з методів об'єкта, а довільною функцією користувача. Такий ітератор можна реалізувати, якщо цю функцію передавати йому через покажчик на функцію.

Визначимо тип покажчика на функцію в такий спосіб:

*typedef void(\*PF)(TObject\*,< додаткові параметри>);*

Функція має обов'язковий параметр типу *TObject* чи *TObject\**, через який їй передається об'єкт, для якого необхідно виконати визначені дії.

Метод - ітератор з'являється в такий спосіб:

*void TGroup::ForEach(PF action,< додаткові параметри >);*

де

*action-єдиний* обов'язковий параметр-покажчик на функцію, що викликається для кожного елемента групи;

*додаткові параметри* – параметри, що передаються функції, яка викликається.

Потім визначається покажчик на функцію і ініціюється переданою ітератору функцією.

*PF pf=myfunc;*

Тоді ітератор буде викликатися, наприклад для додаткового параметра типу int, так:

*gr.ForEach(pf,25);*

### Тут *gr-об'єкт-група*.

**Динамічна ідентифікація типів.**

Динамічна ідентифікація типу характерна для мов, у яких підтримується поліморфізм. У цих мовах можливі ситуації, у яких тип об'єкта на етапі компіляції невідомий.

У С++ поліморфізм підтримується через ієрархії класів, віртуальні функції і покажчики базових класів. При цьому покажчик базового класу може використовуватися або для вказівки на об'єкт базового класу, або для вказівки на об'єкт будь-якого класу похідного від цього базового.

Нехай група містить об'єкти різних класів і необхідно виконати деякі дії тільки для об'єктів визначеного класу. Тоді в ітераторі ми повинні розпізнавати тип чергового об'єкта.

У стандарт мови С++ включені засоби **RTTI** (Run-Time Type Idendification)- динамічна ідентифікація типів. Ці засоби реалізовані в останніх системах Borland C++ (версій 4.0 і вище).

Інформацію про тип об'єкта одержують за допомогою оператора typeid, визначення якого містить заголовний файл <typeinfo.h>.

Є дві форми оператора typeid:

typeid (об'єкт)

typeid (ім'я\_типу)

Оператор typeid повертає посилання на об'єкт типу type\_info.

У класі type\_info перевантажені операції == і != забезпечують порівняння типів.

Функція name() повертає покажчик на ім'я типу.

Є одне обмеження. Оператор typeid працює коректно тільки з об'єктами, у яких визначені віртуальні функції. Більшість об'єктів мають віртуальні функції, хоча б тому, що звичайно деструктор є віртуальним для усунення потенційних проблем з похідними класами. Коли оператор typeid застосовують до неполіморфного класу (у класі немає віртуальної функції), які одержують покажчик чи посилання базового типу.

**Приклади.**

**1**.

#include<iostream.h>

#include<typeinfo.h>

class Base{

virtual void f(){};

//…

};

class Derived: public Base{

//…

};

void main()

{int i;

Base ob,\*p;

Derived ob1;

cout<<typeid(i).name(); //Виводиться **int**

p=&ob1;

cout<<typeid(\*p).name(); // Виводиться **Derived**

}

**2**.

//початок див. вище

void WhatType(Base& ob)

{cout<< typeid(ob).name()<<endl;

}

void main()

{

Base ob;

Derived ob1;

WhatType(ob); //Виводиться **Base**

WhatType(ob1); //Виводиться **Derived**

}

**3.**

//початок див. вище

void main()

{

Base \*p;

Derived ob;

p=&ob;

if(typeid(\*p)==typeid(Derived)) cout<<“p указує на об'єкт типу Derived”;

…

}

Якщо при звертанні typeid(\*p), p=NULL, то збуджується виняткова ситуація bad\_typeid

**Порядок виконання роботи.**

1. Доповнити ієрархію класів лабораторної роботи №3 класами “група”.

Наприклад, для предметної області ВІДДІЛЕННЯ можна запропонувати класи “*ВІДДІЛЕННЯ*”, “*студентська група*”, “*кафедра*”. Рекомендується створити абстрактний клас - “*підрозділ*”, що буде предком усіх груп і абстрактний клас *TObject*, що знаходиться на чолі всієї ієрархії.

2.Написати для класу-групи метод - ітератор.

3.Написати процедуру функції, що виконується для всіх об'єктів, що входять у групу (дивися приклади в додатку).

4.Написати демонстраційну програму, у якій створюються, показуються і руйнуються об'єкти-групи, а також демонструється використання ітератора.

**Методичні вказівки.**

1.Клас-група повинен відповідати ієрархії класів лабораторної роботи №2, тобто об'єкти цих класів повинні входити в групу. Наприклад, для варіанта 1 може бути запропонована наступна ієрархія клас

**TObject**(абстр. клас)

**Персона**(абстр. клас) **Підрозділ** (абстрактна група)

###### Студент Викладач Кафедра Студентська група

###### Зав. кафедрою ВІДДІЛЕННЯ

При цьому ієрархія об'єктів буде мати наступний вид:

**ВІДДІЛЕННЯ**

**Студ\_ група\_1 … Студ\_ група\_N Кафедра\_1 … Кафедра\_N**

**Студенти Студенти Викладачі Викладачі**

**Зав.кафедрою Зав.кафедрою**

2. Для включення об'єктів у групу використовувати третій спосіб (через зв'язаний список структур типу TItem).

3. Приклад визначення доданих абстрактних класів:

*class TObject*

*{*

*public:*

*virtual void Show()=0;};*

*class TDepartment:public TObject // абстрактний клас-група*

*{*

*protected:*

*char name[20]; // найменування*

*TPerson\* head; // керівник*

*TItem\* last; // покажчик на початок зв'язаного списку структур TItem*

*public:*

*TDepartment(char\*,TPerson\*);*

*TDepardment(TDepartment&);*

*~ TDepartment();*

*char\* GetName();*

*TPerson\* GetHead();*

*void SetName(char\* NAME);*

*void SetHead(TPerson\* p);*

*void Insert(TObject\* p);*

*virtual void Show()=0;*

*};*

4.Ієрархія об'єктів створюється в такий спосіб (на прикладі ВІДДІЛЕННЯ):

а)Створюється порожній ВІДДІЛЕННЯ.

б)Створюється порожня КАФЕДРА.

в)Створюються ВИКЛАДАЧІ і включаються в КАФЕДРУ.

г)КАФЕДРА включається у ВІДДІЛЕННЯ.

д) теж повторюється для іншої кафедри.

е)Створюється порожня СТУДЕНТСЬКА ГРУПА.

ж)Створюються СТУДЕНТИ і включаються в СТУДЕНТСЬКУ ГРУПУ.

з) СТУДЕНТСЬКА ГРУПА включається у ВІДДІЛЕННЯ.

и) теж повторюється для іншої студентської групи.

5.Видаляється ВІДДІЛЕННЯ (при виклику деструктора) у зворотному порядку.

6.Метод-ітератор визначається в неабстрактних класах-групах на основі обраних запитів.

Наприклад, для класу *TStudentGroup* може бути запропонований ітератор

*void TStudentGroup::ForEach(PF action, float parametr);*

де *action* - покажчик на функцію, що повинна бути виконана для всіх об'єктів, включених у групу(у даному випадку для всіх СТУДЕНТІВ), *parametr-передана* процедурі додаткова інформація..

У якості переданої методу функції може бути запропонована, наприклад, така: “вивести список студентів, що мають рейтинг не нижче заданого”

*void MyProc(TObject\* p,float rate)*

*{*

*if (((TStudent\*)p) ->GetGrade()>=rate) cout<<(((TStudent\*)p) ->GetName());*

*}*

7. Студент визначає передану ітератору функцію на основі запитів, що повинні бути виконані викликом ітератора. Варіанти запитів приведені в додатку.

**Зміст звіту.**

1.Постановка задачі.

2.Ієрархія класів.

3.Ієрархія об'єктів.

4.Визначення класів( доданих чи змінених у порівнянні з лабораторною роботою №3).

5.Реалізація для одного не абстрактного класу-групи всіх методів.

6.Реалізація ітератора*.*

7.Реалізація переданої ітератору функції.

8.Листинг демонстраційної програми.

**Завдання.**

Створити агрегативну залежність між класами створеними на попередніх лабораторних роботах.

**Приклад виконаного завдання(**Для нащадків віртуального класу програмне забезпечення, поля є об’єктами класу для роботи з датою, та класу студія розробника**).**

**PZ.h**

#pragma once

#include <string.h>

class PZ

{

public:

PZ(void);

~PZ(void);

virtual void show()=0;

virtual void use()=0;

virtual void Input()=0;

};

**PZ.cpp**

#include "StdAfx.h"

#include "PZ.h"

PZ::PZ(void)

{

}

PZ::~PZ(void)

{

}

**Free.h**

#pragma once

#include "PZ.h"

#include "DataTimes.h"

#include <string>

#include <cstdlib>

#include "DevStd.h"

using namespace std;

class Free :

virtual public PZ

{

public:

string name;

DevStd dev;

Free(string name,DevStd dev);

Free();

void Input();

~Free(void);

void show();

void use();

};

**Free.cpp**

#include "StdAfx.h"

#include "Free.h"

#include <string>

#include <cstdlib>

#include <iostream>

#include <string>

#include "PZ.h"

#include "DevStd.h"

Free::Free(string n,DevStd d)

{

name=n;

dev=d;

}

Free::Free(){};

void Free::Input() {

setlocale(LC\_ALL,"rus");

cout<<"Введите названия програмного средства:";

cin>>name;

cout<<"Введите студию разработчика:\n";

dev.input();

}

void Free::use(){

setlocale(LC\_ALL,"rus");

cout<<"Вы можете всегда использывать даное програмоное обеспечение"<<"\n";

}

void Free::show() {

setlocale(LC\_ALL,"rus");

cout<<"Прoграмное средство:\""<<name<<"\" производитель:"<<"\n";

dev.show();

}

Free::~Free(void)

{

}

**Shareware.h**

#pragma once

#include "PZ.h"

#include "DataTimes.h"

#include <string>

#include <cstdlib>

#include <iostream>

#include <string>

#include "DevStd.h"

using namespace std;

class Shareware :

virtual public PZ

{

public:

string name;

DevStd dev;

DataTimes installdata;

int dayoffree;

void use();

void show();

Shareware(string s1,DevStd d,DataTimes intsd,int countday);

Shareware();

void Input();

~Shareware(void);

};

**Shareware.cpp**

#include "StdAfx.h"

#include "Shareware.h"

#include <string>

#include <cstdlib>

#include <iostream>

#include <string>

#include <time.h>

#include "DataTimes.h"

#include "PZ.h"

#include "DevStd.h"

Shareware::Shareware(string s1,DevStd d,DataTimes intsd,int countday)

{

name=s1;

dev=d;

installdata=intsd;

dayoffree=countday;

}

void Shareware::Input()

{

setlocale(LC\_ALL,"rus");

cout<<"Введите названия програмного средства:";

cin>>name;

cout<<"Введите студию разработчика:";

dev.input();

cout<<"Введите дату установки:";

installdata.Input();

cout<<"Введите количесвто дней бесплатного использивания";

cin>>dayoffree;

}

void Shareware::use() {

setlocale(LC\_ALL,"rus");

DataTimes dcon;

dcon=installdata;

dcon.plusDay(dayoffree);

int dn,mn,yyyn;

time\_t t;

tm \*tk;

time(&t);

tk=localtime(&t);

dn=tk->tm\_mday;

mn=1+tk->tm\_mon;

yyyn=1900+tk->tm\_year;

bool flag=true;

if(yyyn>dcon.yyyy) flag=false;

else

{

if(mn>dcon.mm) flag=false;

else

{

if(dn>dcon.dd) flag=false;

}

}

if(flag)

{

cout<<"Вы можете использовать даную програму, дата окончания пробной версии ";

dcon.Print();

}

else

{

cout<<"Вы не можете использовать даную програму, лицензия истекла ";

dcon.Print();

}

}

Shareware::Shareware(){

name="A";

dev.name="A";

dev.strana="UA";

installdata.setAll(1,1,2012);

dayoffree=365;

}

void Shareware::show(){

setlocale(LC\_ALL,"rus");

cout<<"Прoграмное средство:\""<<name<<"\" производитель:"<<"\n";

dev.show();

cout<<"Срок действия триала "<<dayoffree<<"\n";

cout<<"Дата инсталяции";

installdata.Print();

}

Shareware::~Shareware(void)

{

}

**Busines.h**

#pragma once

#include "pz.h"

#include "DataTimes.h"

#include <string>

#include <cstdlib>

#include "DevStd.h"

using namespace std;

class Busines :

virtual public PZ

{

public:

string name;

DevStd dev;

DataTimes installdata;

int dayoffree;

void Input();

void use();

void show();

Busines(string s1,DevStd d,DataTimes intsd,int countday);

Busines(void);

~Busines(void);

};

**Busines.cpp**

#include "StdAfx.h"

#include "Busines.h"

#include <string>

#include <cstdlib>

#include <iostream>

#include <string>

#include <time.h>

#include "DataTimes.h"

#include "PZ.h"

#include "DevStd.h"

using namespace std;

void Busines::use() {

setlocale(LC\_ALL,"rus");

DataTimes dcon;

dcon=installdata;

dcon.plusDay(dayoffree);

int dn,mn,yyyn;

time\_t t;

tm \*tk;

time(&t);

tk=localtime(&t);

dn=tk->tm\_mday;

mn=1+tk->tm\_mon;

yyyn=1900+tk->tm\_year;

bool flag=true;

if(yyyn>dcon.yyyy) flag=false;

else

{

if(mn>dcon.mm) flag=false;

else

{

if(dn>dcon.dd) flag=false;

}

}

if(flag)

{

cout<<"Вы можете использовать даную програму, дата окончания лицензии";

dcon.Print();

}

else

{

cout<<"Вы не можете использовать даную програму, лицензия истекла ";

dcon.Print();

}

}

void Busines::Input()

{

setlocale(LC\_ALL,"rus");

cout<<"Введите названия програмного средства:";

cin>>name;

cout<<"Введите студию разработчика:";

dev.input();

cout<<"Введите дату установки:";

installdata.Input();

cout<<"Введите количесвто дней лицензии";

cin>>dayoffree;

}

void Busines::show(){

setlocale(LC\_ALL,"rus");

cout<<"Прoграмное средство:\""<<name<<"\" производитель:\n";

dev.show();

cout<<"Срок действия "<<dayoffree<<"\n";

cout<<"Дата инсталяции";

installdata.Print();

}

Busines::Busines(void)

{

}

Busines::Busines(string s1,DevStd d,DataTimes intsd,int countday)

{

name=s1;

dev=d;

installdata=intsd;

dayoffree=countday;

}

Busines::~Busines(void)

{

}

**DataTimes.h**

#pragma once

class DataTimes

{

public:

unsigned short int dd;

unsigned short int mm;

unsigned int yyyy;

public:

DataTimes();

DataTimes(unsigned short int d,unsigned short int m,unsigned int y);

~DataTimes();

void setAll(unsigned short int d,unsigned short int m,unsigned int y);

int getYear();

int getMounth();

int getDay();

DataTimes Input();

void Print();

DataTimes NextDay(DataTimes ob);

DataTimes PrevDay(DataTimes ob);

int CountDay();

void plusDay(int count);

void DayToDate();

bool operator&(DataTimes ob1);

bool operator!();

void DateTrue();

bool DataValid();

};

**DataTimes.cpp**

#include "StdAfx.h"

#include "DataTimes.h"

#include <iostream>

#include <string>

using namespace std;

DataTimes::DataTimes():dd(1),mm(1),yyyy(2009){};

DataTimes::DataTimes(unsigned short int d = 1,unsigned short int m = 1,unsigned int y = 2009)

{

}

DataTimes::~DataTimes(void)

{

}

void DataTimes::setAll(unsigned short int d,unsigned short int m,unsigned int y)

{

dd=d;

mm=m;

yyyy=y;

}

int DataTimes::getYear()

{

return yyyy;

}

int DataTimes::getMounth()

{

return mm;

}

int DataTimes::getDay()

{

return dd;

}

DataTimes DataTimes::Input()

{

setlocale(LC\_ALL,"rus");

cout << "Введите год:";

cin >> yyyy;

cout << "Введите месяц:";

cin >> mm;

cout << "Введите день:";

cin >> dd;

setlocale(LC\_ALL,"");

return DataTimes(dd,mm,yyyy);

}

DataTimes DataTimes::NextDay(DataTimes ob)

{

this->dd=ob.dd;

this->mm=ob.mm;

this->yyyy=ob.yyyy;

this->dd++;

if((mm==1)||(mm==3)||(mm==5)||(mm==7)||(mm==8)||(mm==10)||(mm==12))

{

if(dd>31) {dd=1;mm++;}

} else

{

if((mm==4)||(mm==6)||(mm==9)||(mm==11))

{

if(dd>30) {dd=1;mm++;}

} else

{

if(((yyyy%4==0) && (yyyy%100!=0))||((yyyy%4==0) && (yyyy%100!=0) && (yyyy%400==0)))

{

if(dd>29)

{

dd=1;mm++;

}

} else

{

if(dd>28)

{

dd=1;mm++;

}

}

}

}

if(mm>12) {

mm=1;

yyyy++;

}

return DataTimes(dd,mm,yyyy);

}

DataTimes DataTimes::PrevDay(DataTimes ob)

{

this->dd=ob.dd;

this->mm=ob.mm;

this->yyyy=ob.yyyy;

this->dd--;

if((mm==1)||(mm==3)||(mm==5)||(mm==7)||(mm==8)||(mm==10)||(mm==12))

{

if(dd<1) {dd=30;mm--;

if(((yyyy%4==0) && (yyyy%100!=0)&& (mm==2))||((yyyy%4==0) && (yyyy%100!=0) && (yyyy%400==0) && (mm==2)))

{

if(dd<1)

{

dd=29;

}

} else

{

if(dd<1)

{

dd=28;

}

}}

} else

{

if((mm==2)||(mm==4)||(mm==6)||(mm==9)||(mm==11))

{

if(dd<1) {dd=31;mm--;}

}

}

if((mm==0)) {

mm=12;

yyyy--;

}

return DataTimes(dd,mm,yyyy);

}

int DataTimes::CountDay()

{

switch (mm)

{

case 1:case 3:case 5:case 7:case 8:case 10:case 12: { return 31-dd;break;};

case 4:case 6:case 9:case 11: {return 30-dd;break;}

case 2:

{

if(((yyyy%4==0) && (yyyy%100!=0)&& (mm==2))||((yyyy%4==0) && (yyyy%100!=0) && (yyyy%400==0) && (mm==2)))

return 29-dd; else return 28-dd;

break;

}

default: return 0; break;

}

}

void DataTimes::Print() {

printf("%02d.%02d.%04d\n",dd,mm,yyyy);

}

bool DataTimes::operator&(DataTimes ob) {

if((ob.dd==dd)&&(ob.mm==mm)&&(ob.yyyy==yyyy)) return 1; else return 0;

}

bool DataTimes::operator!(){

if(dd<28) return true; else

{

if((mm==1)||(mm==3)||(mm==5)||(mm==7)||(mm==8)||(mm==10)||(mm==12)) {

if(dd==31) return false; else return true;

} else

{

if((mm==4)||(mm==6)||(mm==9)||(mm==11))

{

if(dd==30) return false; else return true;

} else

{

if(((yyyy%4==0) && (yyyy%100!=0)&& (mm==2))||((yyyy%4==0) && (yyyy%100!=0) && (yyyy%400==0) && (mm==2)))

{

if(dd==29) return false; else return true;

} else if(dd==28) return false; else return true;

}

}

}

}

void DataTimes::plusDay(int count) {

dd=dd+count;

while(!this->DataValid())

{

if((mm==1)||(mm==3)||(mm==5)||(mm==7)||(mm==8)||(mm==10)||(mm==12))

{

if(dd>31) {dd-=31;mm++;}

} else

{

if((mm==4)||(mm==6)||(mm==9)||(mm==11))

{

if(dd>30) {dd-=30;mm++;}

} else

{

if(((yyyy%4==0) && (yyyy%100!=0))||((yyyy%4==0) && (yyyy%100!=0) && (yyyy%400==0)))

{

if(dd>29)

{

dd-=29;mm++;

}

} else

{

if(dd>28)

{

dd-=28;mm++;

}

}

}

}

if(mm>12) {

mm=1;

yyyy++;

}

}

}

bool DataTimes::DataValid(){

if((mm==1)||(mm==3)||(mm==5)||(mm==7)||(mm==8)||(mm==10)||(mm==12))

{

if(dd>31) return false;

} else

{

if((mm==4)||(mm==6)||(mm==9)||(mm==11))

{

if(dd>30) return false;

} else

{

if(((yyyy%4==0) && (yyyy%100!=0))||((yyyy%4==0) && (yyyy%100!=0) && (yyyy%400==0)))

{

if(dd>29)

{

return false;

}

} else

{

if(dd>28)

{

return false;

} else return true;

}

}

}

}

**DevStd.h**

#pragma once

#include <iostream>

#include <string>

using namespace std;

class DevStd

{

public:

string name,strana;

DevStd(void);

void input();

void show();

~DevStd(void);

};

**DevStd.cpp**

#include "StdAfx.h"

#include "DevStd.h"

DevStd::DevStd(void)

{

}

void DevStd::input() {

setlocale(LC\_ALL,"rus");

cout << "Введите название студии:";

cin >> name;

cout << "Введите страну студии:";

cin >> strana;

}

void DevStd::show() {

setlocale(LC\_ALL,"rus");

cout << "название студии:"<<name<<endl;

cout << "странa студии:"<<strana<<endl;

}

DevStd::~DevStd(void)

{

}

**Project\_1.cpp**

// Project\_1.cpp: определяет точку входа для консольного приложения.

//

#include "stdafx.h"

#include "PZ.h"

#include "Free.h"

#include "Shareware.h"

#include "DataTimes.h"

#include "Busines.h"

#include <iterator>

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

{

/\*Free ob1;

ob1.Input();

ob1.show();

ob1.use();

Shareware ob2;

ob2.Input();

ob2.show();

ob2.use();

system("pause");

Busines ob3;

ob3.Input();

ob3.show();

ob3.use(); \*/

PZ \*\*pp;

pp = new PZ\*[5];

pp[0]= new Free();

pp[1]= new Shareware();

pp[2]= new Busines();

for(int i=0;i<3;i++) pp[i]->Input();

for(int i=0;i<3;i++) pp[i]->show();

for(int i=0;i<3;i++) pp[i]->use();

system("pause");

return 0;

}

## Лабораторна робота №6. Делегати.

**Мета:** Одержати практичні навички створення та роботи з делегатами.

**Основний зміст роботи.**

Створити декілька делегатів і використати їх для даних різних методів.

**Короткі теоретичні відомості.**

**Делегат –** особливий вид класу,яки зберігає посилання на методы.

**Делегат** – клас (контрольний тип), який інкапсулює (містить в собі) метод з вказаною сигнатурою і повертаємим типом.

Всі делегати є об'єктами типу System.Delegate або System.MulticastDelegate.

Загальний синтаксис оголошення делегата:

[Cпеціфікатори] delegate тип значення методу ім'я делегата ([параметри]), де

delegate - ключове слово.

тип значення - це той тип, який буде повертати функція, на яку посилається делегат.

параметри - це необов'язкові параметри, які будуть присутні у функції, на яку посилається делегат.

**Приклади оголошення делегата:**

delegate void MyDelegate (string s); // делегат, який може працювати для всіх методів, які повертають тип void і мають один строковий параметр.

public delegate int mydel ();

delegate double MyD (double x);

// "Майже універсальний делегат"

delegate void My (object o)

**Алгоритм роботи з делегатом:**

1. Оголошення делегата;
2. Визначення методу, який “підходить” для роботи з делегатом
3. В основній програмі створення екземпляру делегата і зв’зок його з визначеною функцією;
4. Виклик методу через делегат.

Делегат може бути використаним:

* Для звичайної функції без класу;
* Для методу всередині класу.

**Приклад делегата для нормальної функції**

class Program

    {

        // оголошення делегата

        delegate void MyDelegate (string s);

        static void Myfunc (string s)

        {

            Console.WriteLine (s);

        }

        static void Main (string [] args)

        {

            // в конструкторі передається посилання на функцію, яка делегується (связиватся з делегатом)

            MyDelegate del = new MyDelegate (Myfunc); // створення екземпляра делегата

            del ( "Hello World"); // виклик функції через делегат

            Console.ReadKey ();

        }

    }

**Приклад делегата для методу класу**

namespace ConsoleApplication1

{

class my

{

    public string mystring; // поле класу

    public void Myfunc (string s)

  {

    // Висновок значення поля mystring, а також аргументу, переданого методу.

    Console.WriteLine (mystring + s);

  }

}

    class Program

    {

        // оголошення делегата

        delegate void MyDelegate (string s);

        static void Main (string [] args)

        {

            // Створимо екземпляр тестового класу.

        my A = new my ();

        // Створимо екземпляр делегата, що містить, крім посилання на сам метод, також посилання

        // на об'єкт, для якого буде викликаний метод.

        MyDelegate del = new MyDelegate (A.Myfunc);

        // Змінюємо значення поля тестового об'єкта.

        A.mystring = "Hello, World!";

        // Викличемо метод через делегат.

        del ( "- from Delegate");

        // Еквівалентний наступного виклику

        // A.Myfunc ( "- from Instance Delegate");

        // Знову змінимо значення поля тестового об'єкта.

        A.mystring = "Good bye, World!";

        // Знову звернемося до методу.

        del ( "- from Delegate");

            Console.ReadKey (); }}}

**Операції з делегатами: =,!=, +=,-=,= =.**

Делегати можна *порівнювати на рівність и нерівність.* Два делегати рівні, якщо вони обидва не містять посилань на методи, або якщо вони містять посилання на одні й ті самі ж методи в одному й тому самому порядку. Порівнювати можна навіть делегати різноманітних типів при умові, що вони мають один і той самий тип повертаємого значення і однакові списки параметрів.

З делегатами одного типу можна виконати операциї *простого и складного присвоєння:* =, +=,-=.

**Використання делегатів**

Існує 4 види використання делегатів:

**1. Делегат для вибору методів делегування** (програміст не знає, який метод йому знадобиться і задає або визначає його під час виконання програми);

**2. Делегат - оповіщувач спостерігача** (патерн "спостерігач");

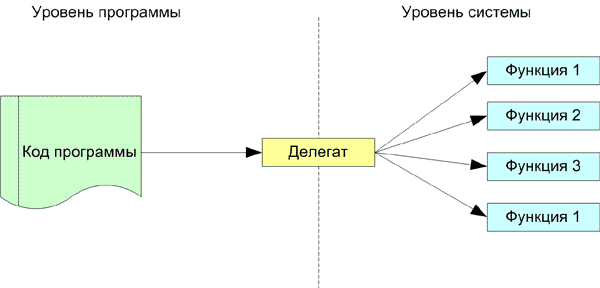
**3. Делегат, як зворотний виклик методу**;

**4. Делегат, як обробник подій**. Зазвичай події обробляються непростими методами, а делегатами. Подія генерує ідею "відбулося щось важливе", і програма повинна на нього відреагувати. Події і делегати є тісно пов'язаними поняттями, оскільки обробка подій вимагає точного вибору обробника. Обробник події реалізується на мові С #, як правило, у вигляді делегата.

**1.Делегат для вибору методів на етапі виконання**

Делегати використовуються для отримання можливості визначати викликається метод не при компіляції, а під час виконання програми (динамічно). Тобто делегат може зв'язуватися ні з одним методом, а з декількома (екземпляри делегата можуть містити кілька посилань на методи).

Завдяки цьому, можна приєднувати до одного делегату кілька методів, кожен з яких при єдиному зверненні до делегату буде викликатися по ланцюжку. Таким чином, з програми буде видно лише один делегат, за яким ховається кілька методів (рисунок).



## 

Delegate.Invoke або що там всередині? (Для просунутих)

Метод Invoke документально. Він не є членом класів Delegate і MulticastDelegate. Відповідно, можна припустити, що це спеціальний метод, що генерується компілятором. Його вивчення показує, що метод позбавлений будь-якого коду. Він просто порожній і нічого не може робити. Так може здатися, якщо не брати до уваги специфікатор runtime, який використовується при визначенні методу. Даний специфікатор вказує на те, що метод буде реалізований по ходу виконання програми самим середовищем виконання. Це означає, що код, що обробляє виклик методу, розташовується в надрах самої середовища виконання. А вона прекрасно обізнана про внутрішній устрій делегата і впорається з викликом всіх його методів без додаткової допомоги з боку програміста.

Код, який реалізує делегат, також знаходиться всередині середовища виконання. Виявляється, метод Invoke відповідає не тільки за виклик делегата, але також за зберігання інформації про прототип методів, на які може посилатися делегат. Прототип закодований в самому методі Invoke, тобто його прототип повністю збігається з прототипом делегата.

#### *Delegate.Invoke або що там всередині? (Для просунутих)*

*Метод Invoke недокументован. Він не є членом класів Delegate і MulticastDelegate. Відповідно, можна припустити, що це спеціальний метод, що генерується компілятором. Його вивчення показує, що метод позбавлений будь-якого коду. Він просто порожній і нічого не може робити. Так може здатися, якщо не брати до уваги специфікатор runtime, який використовується при визначенні методу. Даний специфікатор вказує на те, що метод буде реалізований по ходу виконання програми самим середовищем виконання. Це означає, що код, що обробляє виклик методу, розташовується в надрах самої середовища виконання. А вона прекрасно обізнана про внутрішній устрій делегата і впорається з викликом всіх його методів без додаткової допомоги з боку програміста.*

*Код, який реалізує делегат, також знаходиться всередині середовища виконання. Виявляється, метод Invoke відповідає не тільки за виклик делегата, але також за зберігання інформації про прототип методів, на які може посилатися делегат. Прототип закодований в самому методі Invoke, тобто його прототип повністю збігається з прототипом делегата.*

#### *MulticastDelegate.GetInvocationList*

*Повертає список делегатів, які перебувають у списку дзвінків делегата.*

*public sealed override Delegate [] GetInvocationList ();*

*Існує два недоліки механізму виклику функцій, пов'язаних з делегатом. І обидва вони пов'язані з випадком, коли в списку виклику делегата присутні більше однієї функції.*

* *Функції, на які посилаються делегати, можуть повертати значення. Але ми їх отримати не зможемо, оскільки штатна функція виклику Invoke повертає значення, яка повернула остання з опитаних функцій.*
* *При зверненні до делегату, який містить в списку виклику кілька функцій, зовсім не гарантується, що всі функції зі списку будуть викликані. Якщо одна з них згенерує виняток, то робота методу Invoke буде перервана і інші функції викликані не будуть.*

Приклад делегата для роботи з декількома методами

namespace ConsoleApplication1

{

    class Program

    {

        // оголошення делегата

        delegate void MyDelegate (string s);

        // Перший метод, на який ми будемо посилатися за допомогою

        // делегата. Саме він буде викликати виключення,

        // що не дозволяє звернутися до другого методу.

 static void f1 (string s)

  {

    // Виведемо значення переданого параметра,

    // а також повідомимо користувача про те, що даний

    // метод був викликаний.

    Console.WriteLine ( "Функція 1 викликана з параметром = {0}", s);

    // Навмисно викинемо виняток.

    throw new Exception ();

  }

 // Другий метод, на який ми будемо посилатися з делегата.

  static void f2 (string s)

  {

    // Повідомимо користувачеві про те, що метод

    // був викликаний, а також виведемо значення переданого параметра.

    Console.WriteLine ( "Функція 2 викликає з параметром = {0}", s);

  }

        static void Main (string [] args)

        {

                 MyDelegate del = new MyDelegate (f1);

                // Приєднаємо до нього ще одну функцію.

                del + = new MyDelegate (f2);

                // Послідовно пройдемо по кожному делегату, що входить

                // в список виклику раніше створеного делегата.

                foreach (MyDelegate d in del.GetInvocationList ())

                {

                  // виклик функції - в захищений блок!

                   try

                  {

                    d ( "Hello");

                  }

                  // Це блок обробки виключень, що відбулися

                  // в захищеному блоці.

                  catch (Exception ex)

                  {

                    // Повідомимо користувачеві про те, що при спробі

                    // виклику однієї з функцій відбулося виключення.

                    Console.WriteLine ( "Oh mama, була виявлена ​​виняткова ситуація!");

                  }

                }

            Console.ReadKey (); }}}

**2. Делегат як оповіщувач спостерігача. Патерн «спостерігач»**

**Патерн** – шаблон проектування (одна і та пані схема організації і взаємодії об'єктів в різних ситуаціях). Існує кілька патернів програмування, одним з яких є «патерн спостерігач».

Об'єкт, званий джерелом при зміні свого стану, може становити інтерес для інших об'єктів, завдяки чому він повинен посилати їм повідомлення про те, що у нього, щось сталося. Такі об'єкти, отримавши повідомлення повинні оновити свій стан, що б синхронізувати свої статки з об'єктом джерелом. Такі об'єкти називаються спостерігачами.

Спостерігачі синхронізували свій стан з джерелом

Спостерігач1 …

*Джерело*

Спостерігач n

Змінив стан – відіслав всім спостерігачам звістку.

Приклад делегата для оповіщення спостерігачів

namespace ConsoleApplication1

{

    delegate void MyDelegate (object o);

    class Джерело

    {

// Делегат-поле класу

        MyDelegate del;

        public void Reg (MyDelegate d)

// додавання делегатів

        {Del + = d; }

// подія, що виникає всередині джерела - посилка спостерігачам сповіщення

        public void Event ()

        {Console.WriteLine ( "Подію створено!");

        if (del! = null) // якщо в делегата є функція

del (this); // ВИКОНАТИ метод, "прив'язаний" до делегату

        }

// видалення делегатів зі списку зареєстрованих

  public void UnReg (MyDelegate d)

        {Del - = d; }

    }

    class Спостерігач1

    {

// "відповідний" делегату метод

        public void Do\_Event\_1 (object o)

        {Console.WriteLine ( "БАЧУ, що відбулася подія !!!"); }

    }

    class Спостерігач2

    {

        public static void Do\_Event\_2 (object o)

        {Console.WriteLine ( "Я теж БАЧУ, що відбулася подія !!!"); }

    }

    class Program

    {Static void Main (string [] args)

        {Джерело A = new Джерело ();

            Спостерігач1 B = new Спостерігач1 ();

            Спостерігач1 C = new Спостерігач1 ();

// зареєстрували делегат (створили його екземпляр), "прив'язали" до нього метод

            A.Reg (new MyDelegate (B.Do\_Event\_1));

            A.Reg (new MyDelegate (С.Do\_Event\_1));

// те ж саме, тільки для СТАТИЧНОЇ методу

            A.Reg (new MyDelegate (Спостерігач2.Do\_Event\_2));

            A.Event ();

// прибрали одного спостерігача для джерела

A.UnReg (new MyDelegate (B.Do\_Event\_1));

            A.Event ();

            Console.ReadKey (); }}}

**3. Делегат, як зворотний виклик методу**;

Так як делегат є класом, його можна передавати в методи в якості параметра. Це називається **функціональної параметризацією.** Функціональна параметризация застосовується для створення універсальних методів (наприклад, метод виведення таблиці значень функції) і можливості забезпечення зворотного виклику.

**Зворотній виклик (callback)** являє собою виклик функції, переданої в іншу функцію як параметр.

Як приклад "універсального" методу і демонстрації зворотного виклику функції розглянемо передачу делегата в якості параметра методу, який виводить на екран таблицю значень функції. Як параметри виступають діапазон значень аргументу, крок його вимірів і вид обчислюється функції (ДЕЛЕГАТ).

Приклад передачі делегата в якості параметра функції

namespace ConsoleApplication1

{

    delegate double MyDelegate (double x);

    class Program

    {

// в якості параметрів - делегат (вид функції), діапазон значень)

        public static void Table (MyDelegate del, double x, double b)

        {Console.WriteLine ( "========= X ================ Y ================= = ");

            while (x <= b)

            {

// виклик функції через делегат

                Console.WriteLine ( "| {0,8: 0.000} | {1,8: 0.000} |", x, del (x));

                x + = 1;

            }

            Console.WriteLine ( "--------------------------------------------- ");

        }

        static void Main (string [] args)

        {Console.WriteLine ( "Таблиця функції синуса:");

// викликається фунция sin (x)

// аналогічно написати - new MyDelegate (Math.Sin)

            Table (Math.Sin, -2, 2);

            Console.WriteLine ( "Таблиця будь-якої функції, що приймає будь-яке double і повертає завжди 1 !!!:");

// АНОНІМНИЙ МЕТОД !!!

            Table (delegate (double x) {return 1;}, 0, 3);

            Console.ReadKey ();

        }

    }

}

**4. Делегат, як обробник подій**.

**Подія** - це елемент класу, що дозволяє класу посилати іншим об'єктам повідомлення про зміну свого стану. При цьому для цих об'єктів (спостерігачів) повинні оброблятися ці події. Обробка подій повинна бути зареєстрована в об'єкті - джерелі.

Таким чином, механізм подій реалізує патерн "спостерігач".

Такий патерн можна ще назвати моделлю "**публікація-підписка**" - один клас (джерело, відправник) повідомлення публікує події, які він може створювати, а інші класи (одержувачі, спостерігачі) підписуються на отримання цих подій.

Події тісно пов'язані з делегатами - за допомогою делегатів викликаються обробники подій.

**Загальний синтаксис оголошення події:**

**[Cпеціфікатори] event тип делегата ім'я події, де**

**специфікатор** - new, public, virtual, static і ін, причому подія може бути статичним (для класу) і звичайним (для екземпляра класу)

**event**- ключове слово.

**тип делегата** - це делегат, методи якого будуть обробляти цю подію

Приклад оголошення події:

public delegate void My (object o); // оголошення делегата

class A

{Public event My My\_Event // оголошення події My\_Event

}

**Алгоритм роботи з подіями:**

1. Оголошення делегата, що задає сигнатуру методів - обробників подій

2. Оголошення події всередині класу (джерела)

3. Оголошення і реалізація методу, створюваного подія

4. Оголошення і реалізація методів всередині класів (спостерігачів) - обробників події (прив'язаних до делегату). Їх сигнатура повинна збігатися з делегатом.

5. Кожен екземпляр класу, який бажає отримувати повідомлення про подію повинен зареєструвати в екземплярі джерелі метод обробки події.

Приклад делегата для оповіщення спостерігачів за допомогою подій

namespace ConsoleApplication1

{Public delegate void MyDelegate ();

    class Джерело

    {// подія - поле класу

        public event MyDelegate My\_event;

        public void Створити\_подію ()

        {Console.WriteLine ( "Подію створено !!!");

// значення події за замовчуванням null

            if (My\_event! = null) My\_event ();

        }

     }

    class Спостерігач1:

    {Public void Do\_Event\_1 ()

        {Console.WriteLine ( "БАЧУ, що відбулася подія !!!"); }

    }

    class Спостерігач2:

    {Public static void Do\_Event\_2 ()

        {Console.WriteLine ( "Я теж БАЧУ, що відбулася подія !!!"); }

    }

    class Program

    {Static void Main (string [] args)

        {Джерело A = new Джерело ();

            Спостерігач1 B = new Спостерігач1 ();

            Спостерігач1 С = new Спостерігачй ();

// реєстрація обробників подій

            A.My\_event + = new MyDelegate (B.Do\_Event\_1);

            A.My\_event + = new MyDelegate (С.Do\_Event\_1);

            A.My\_event + = new MyDelegate (Спостерігач2.Do\_Event\_2);

            A.Створити\_подію ();

            A.My\_event - = new MyDelegate (B.Do\_Event\_1);

            A.Створити\_подію (); Console.ReadKey ();

        }

    }

}

**Стандартні делегати**

У бібліотеці .NET є велика кількість стандартних делегатів, призначених для реалізації обробки подій. Більшість з таких класів оформлено так:

• Ім'я делегата закінчується EventHandler

• Делегат отримує 2 параметра - джерело події (тип object), аргументи події (тип EventArgs)

Якщо обробникам події потрібна специфічна інформація про подію, то для цього створюють клас, похідний від стандартного класу EventArgs, і додають власний код. Якщо ж програміст не буде використовувати в делегата власний код, то власний клас не створюється, а використовується стандартний клас System.EventHandler.

Приклад стандартного делегата EventHandler для оповіщення спостерігачів за допомогою подій

namespace ConsoleApplication1

{Class Джерело

    {Public event EventHandler My\_event;

        public void Створити\_подію ()

        {Console.WriteLine ( "Подію створено !!!");

            if (My\_event! = null) My\_event (this, null);

        }

     }

    class Спостерігач1

    {Public void OnDo\_Event (object sender, EventArgs e)

        {Console.WriteLine ( "БАЧУ, що відбулася подія !!!"); }

    }

    class Спостерігач2

    {Public static void OnDo\_Event (object sender, EventArgs e)

        {Console.WriteLine ( "Я теж БАЧУ, що відбулася подія !!!"); }

    }

    class Program

    {Static void Main (string [] args)

        {Джерело A = new Джерело ();

            Спостерігач1 B = new Спостерігач1 ();

            Спостерігач1 C = new Спостерігач1 ();

            A.My\_event + = new EventHandler (B.OnDo\_Event);

            A.My\_event + = new EventHandler (C.OnDo\_Event);

            A.My\_event + = new EventHandler (Спостерігач2.OnDo\_Event);

// анонімний обробник !!! Створення нового методу для обробки події !!!

           A.My\_event + = delegate (object sender, EventArgs e)

{Console.WriteLine ( "Я теж з вами обробляю цю подію"); };

A.Створити\_подію (); Console.ReadKey (); }

    }

}

**Завдання до ЛР6.**

**В проєкті до Задачі з ЛР5 (с#) програмно реалізувати декілька делегатів(2-4), застосовуючи різні способи їх використання. Прокоментувати алгоритм роботи з делегатами в проєкті.**

**На оцінку «задовільно» можна до Задачі 1 з ЛР1 (с#) програмно реалізувати декілька делегатів(2).**

## Лабораторна робота №7. Шаблони функцій і класів.Контейнери. STL. Колекції в С#. Серіалізація і Десеріалізація

**Мета.** Одержати практичні навички створення шаблонів і використання їх у програмах С++, роботи з серіалізацією та десеріалізацією.Робота з колекціями в С#.

**Основний зміст роботи.**

Створити шаблон заданого класу і використовувати його для даних різних типів.

Виконати серіалізацію і десеріалізацію.

**Короткі теоретичні відомості.**

**Шаблон функції.**

Шаблон функції (інакше параметризована функція) визначає загальний набір операцій (алгоритм), що будуть застосовуватися до даних різних типів. При цьому тип даних, над якими функція повинна виконувати операції, передається їй у виді параметра на стадії компіляції.

У С++ параметризована функція створюється за допомогою ключового слова **template**. Формат шаблона функції:

template <class тип\_даних> тип\_поверн\_значення ім'я\_функції(список\_параметрів){тіло\_функції}

**Основні властивості параметрів шаблона функції.**

1. Імена параметрів шаблона повинні бути унікальними у всім визначенні шаблона.
2. Список параметрів шаблона не може бути порожнім.
3. У списку параметрів шаблона може бути кілька параметрів і кожному з них повинне передувати ключове слово class.
4. Ім'я параметра шаблона має усі права імені типу у визначеної шаблоном функції.
5. Визначена за допомогою шаблона функція може мати будь-яку кількість непараметризованих формальних параметрів. Може бути непараметризоване і значення, що повертається функцією.
6. У списку параметрів прототипу шаблона імена параметрів не зобов'язані збігатися з іменами тих же параметрів у визначенні шаблона.
7. При конкретизації параметризованої функції необхідно, щоб при виклику функції типи фактичних параметрів, що відповідають однаково параметризованим формальним параметрам, були однакові.

**Шаблон класу.**

Шаблон класу(інакше параметризований клас) використовується для побудови родового класу. Створюючи родовий клас, ви створюєте ціле сімейство родинних класів, які можна застосовувати до будь-якого типу даних. Таким чином, тип даних, яким оперує клас, указується як параметр при створенні об'єкта, що належить до цього класу. Подібно тому, як клас визначає правила побудови і формат окремих об'єктів, шаблон класу визначає спосіб побудови окремих класів. У визначенні класу, що входить у шаблон, ім'я класу є не ім'ям окремого класу, а параметризованим ім'ям сімейства класів.

Загальна форма оголошення параметризованого класу:

*template <class тип\_даних> class ім'я\_класу { . . . };*

**Основні властивості шаблонів класів.**

1. Компонентні функції параметризованого класу автоматично є параметризованими. Їх не обов'язково повідомляти як параметризовані за допомогою *template.*
2. Дружні функції, що описуються в параметризованому класі, не є автоматично параметризованими функціями, тобто за замовчуванням такі функції є дружніми для всіх класів, що організуються по даному шаблоні.
3. Якщо *friend-функція* містить у своєму описі параметр типу параметризованого класу, то для кожного створеного по даному шаблоні класу є власна *friend-функція*.
4. У рамках параметризованого класу не можна визначити *friend-шаблони* (дружні параметризовані класи).
5. З однієї сторони шаблони можуть бути похідними (успадковуватися) як від шаблонів, так і від звичайних класів, з іншого боку, вони можуть використовуватися в якості базових для інших шаблонів чи класів.
6. Шаблони функцій, які є членами класів, не можна описувати як *virtual.*
7. Локальні класи не можуть містити шаблони як свої елементи.

**Компонентні функції параметризованих класів**

Реалізація компонентної функції шаблона класу, що знаходиться поза визначенням шаблона класу, повинна включати додатково наступні два елементи:

1. Визначення повинне починатися з ключового слова *template*, за яким слідує такий же *список\_параметрів\_типів* у кутових дужках, який зазначений у визначенні шаблона класу.
2. За *ім'ям\_класу, що* передує операції області видимості (**::**), повинний слідувати *список\_імен\_параметрів* шаблона.

*template<список\_типів>тип\_поверн\_значення ім'я\_класу<список\_імен\_ параметрів> : : ім'я\_функції(список\_параметрів){ . . . }*

## Стандартна бібліотека шаблонів

**Мета.** Освоїти технологію узагальненого програмування з використанням бібліотеки стандартних шаблонів (STL) мови C++.

**Основний зміст роботи.**

Написати чотири програми з використанням STL. Перша і друга програми повинні демонструвати роботу з контейнерами STL, третя і четверта — використання алгоритмів STL.

**Основні теоретичні відомості.**

**Стандартна бібліотека шаблонів (STL).**

STL забезпечує загально цільові, стандартні класи і функції, що реалізують найбільш популярні і широко використовувані алгоритми і структури даних.

STL будується на основі шаблонів класів і тому вхідні в неї алгоритми і структури застосовуються майже до всіх типів даних.

**Склад STL.**

Ядро бібліотеки утворять три елементи: **контейнери**, **алгоритми** і **ітератори**.

**Контейнери** (containers) – це об'єкти, призначені для збереження інших елементів. Наприклад: вектор, лінійний список, множина.

**Асоціативні контейнери** (associative containers) дозволяють за допомогою ключів одержати швидкий доступ до значень, що зберігаються в них.

У кожному класі-контейнері визначений набір функцій для роботи з ними. Наприклад: список містить функції для вставки, видалення і злиття елементів.

**Алгоритми** (algorithms) виконують операції над змістом контейнера. Існують алгоритми для ініціалізації, сортування, пошуку, заміни вмісту контейнерів. Багато алгоритмів призначені для роботи з послідовністю (sequence), що являє собою лінійний список елементів всередині контейнера.

**Ітератори** (iterators) – це об'єкти, що стосовно контейнера відіграють роль покажчиків. Вони дозволяють одержати доступ до вмісту контейнера приблизно так, як покажчики використовуються для доступу до елементів масиву.

З ітераторами можна працювати так само, як з покажчиками. До них можна застосувати операції \*, інкремента, декремента. Типом ітератор являється тип iterator, що визначений у різних контейнерах.

Існує п'ять типів ітераторів:

**1.Ітератори вводу** (input\_iterator) підтримують операції рівності, розіменування та інкремента.

==, !=, \*i, ++i, i++, \*i++

Спеціальним випадком ітератора вводу є istream\_iterator.

**2.Ітератори виводу** (output\_iterator) підтримують операції розіменування, припустиме тільки з лівої сторони присвоювання та інкремента.

++i, i++, \*i=t, \*i++=t

Спеціальним випадком ітератора виводу є ostream\_iterator.

**3.Односпрямовані літератори** (forward\_iterator) підтримують всі операції ітераторів вводу/виводу і, крім того, дозволяють без обмеження застосовувати присвоювання:

==, !=, =, \*i, ++i, i++, \*i++

**4.Двохнаправлені літератори** (biderectional\_iterator) мають усі властивості forward-ітераторів, а також мають додаткову операцію декремента (--i, i--, \*i--), що дозволяє їм проходити контейнер в обох напрямках.

**5.Ітератори довільного доступу** (random\_access\_iterator) мають усі властивості biderectional-ітераторів, а також підтримують операції порівняння та адресної арифметики, а саме безпосередній доступ за індексом.

i+=n, i+n, i-=n, i-n, i1-i2, i[n], i1<i2, i1<=i2, i1>i2, i1>=i2

У STL також підтримуються **зворотні ітератори** (reverse iterators). Зворотніми ітераторами можуть бути або двонаправлені ітератори, або ітератори довільного доступу, але минаючі послідовність у зворотному напрямку.

Надодачу до контейнерів, алгоритмів і ітераторів у STL підтримується ще кілька стандартних компонентів. Головними серед них є **розподільники пам'яті**, **предикати** і **функції** **порівняння**.

У кожного контейнера є визначений для нього розподільник пам'яті (**allocator**), що керує процесом виділення пам'яті для контейнера.

За замовчуванням розподільником пам'яті є об'єкт класу **allocator**. Можна визначити власний розподільник.

У деяких алгоритмах і контейнерах використовується функція особливого типу, що називається **предикатом**. Предикат може бути унарним і бінарним. Значення, що повертається: істина або неправда. Точні умови одержання того чи іншого значення визначаються програмістом. Тип унарних предикатів **UnPred**, бінарних – **BinPred**. Тип аргументів відповідає типу об'єктів, що зберігаються в контейнері.

Визначено спеціальний тип бінарного предиката для порівняння двох елементів. Він називається **функцією порівняння** (comparison function). Функція повертає істину, якщо перший елемент менше другого. Типом функції є тип **Comp**.

Особливу роль у STL грають об'єкти-функції.

Об'єкти-функції - це екземпляри класу, у якому визначена операція “круглі дужки” (). У ряді випадків зручно замінити функцію на об'єкт - функцію. Коли об'єкт – функція використовується як функцію, то для її виклику використовується operatot().

Приклад 1.

*class less{*

*public:*

*bool operator()(int x,int y)*

*{return x<y;}*

*};*

**3.Класи-контейнери.**

У STL визначені два типи контейнерів – послідовності й асоціативні контейнери.

Ключова ідеядля стандартних контейнерів полягає в тім, що коли це представляється розумним, вони повинні бути логічно взаємозамінними. Користувач може вибирати між ними, ґрунтуючись на розуміннях ефективності і потреби в спеціалізованих операціях. Наприклад, якщо часто потрібен пошук по ключі. можна скористатися **map** (асоціативним масивом). З іншого боку, якщо переважають операції, характерні для списків, можна скористатися контейнером **list**. Якщо додавання і видалення елементів часто виконується в кінці контейнера, варто подумати про використання черги **queue**, черги з двома кінцями **deque**, стека **stack**. За замовчуванням користувач повинний використовувати **vector**; він реалізований, щоб добре працювати для самого широкого діапазону задач.

Ідея звертання з різними видами контейнерів – і, у загальному випадку, із усіма видами джерел інформації – уніфікованим способом, веде до поняття **узагальненого програмування**. Для підтримки цієї ідеї STL містить множину узагальнених алгоритмів. Такі алгоритми рятують програміста від необхідності знати подробиці окремих контейнерів.

У STL визначені наступні класи-контейнери(у кутових дужках зазначені заголовні файли, де визначені ці класи):

**bitset** множину бітів <bitset.h>

**vector** динамічний масив <vector.h>

**list** лінійний список <list.h>

**deque** двостороння черга <deque.h>

**stack** стек <stack.h>

**queue** черга <queue.h>

**priority\_queue** черга з пріоритетом <queue.h>

**map** асоціативний список для збереження пар ключ/значення, де з кожним ключем зв'язане одне значення <map.h>

**multimap** з кожним ключем зв'язано два чи більше значення <map.h>

**set**  множина <set.h>

**multiset** множина, у якій кожен елемент не обов'язково унікальний <set.h>

**Огляд операцій**

Типи

**value\_type** тип елемента

**allocator\_type** тип розподільника пам'яті

**size\_type** тип індексів, лічильника елементів і т.д.

**iterator** поводиться як value\_type\*

**reverse\_iterator** переглядає контейнер у зворотному порядку

**reference** поводиться як value\_type&

**key\_type** тип ключа (тільки для асоціативних контейнерів)

**key\_compare** тип критерію порівняння (тільки для асоціативних контейнерів)

**mapped\_type** тип відображеного значення

Ітератори

**begin()** вказує на перший елемент

**end()** вказує на елемент, що випливає за останнім

**rbegin()** вказує на перший елемент у зворотній послідовності

**rend()** вказує на елемент, що випливає за останнім у зворотній послідовності

Доступ до елементів

**front()** посилання на перший елемент

**back()** посилання на останній елемент

**operator[]**(i) доступ за індексом без перевірки

**at**(i) доступ за індексом з перевіркою

Включення елементів

**insert(**p,x**)** додавання х перед елементом, на який вказує р

**insert(**p,n,x**)** додавання n копій х перед р

**insert(**p,first,last**)** додавання елементів з [first:last] перед р

**push\_back(**x**)** додавання х в кінець

**push\_front(**x**)** додавання нового першого елемента (тільки для списків і черг із двома кінцями)

Видалення елементів

**pop\_back()** видалення останнього елемента

**pop\_front()** видалення першого елемента (тільки для списків і черг із двома кінцями)

**erase(**p**)** видалення елемента в позиції р

**erase(**first,last**)** видалення елементів з [first:last]

**clear()** видалення всіх елементів

Інші операції

**size()** число елементів

**empty()** контейнер порожній?

**capacity()** пам'ять, виділена під вектор (тільки для векторів)

**reserve(**n**)**  виділяє пам'ять для контейнера під n елементів

**resize(**n**)** змінює розмір контейнера (тільки для векторів, списків і черг із двома кінцями)

**swap(**x**)**  обмін місцями двох контейнерів

**==**, **!=**, **<**  операції порівняння

Операції присвоювання

**operator=(**x**)** контейнеру привласнюються елементи контейнера х

**assign(**n,x**)** присвоювання контейнеру n копій елементів х (не для асоціативних контейнерів)

**assign(**first,last**)** присвоювання елементів з діапазону [first:last]

Асоціативні операції

**operator[](**k**)** доступ до елемента з ключем k

**find(**k**)** знаходить елемент із ключем k

**lower\_bound(**k**)** знаходить перший елемент із ключем k

**upper\_bound(**k**)** знаходить перший елемент із ключем великим k **equal\_range(**k**)** знаходить lower\_bound (нижню границю) і upper\_bound (верхню границю) елементів із ключем k

**Контейнери vector - вектор.**

Вектор vector у STL визначений як динамічний масив з доступом до його елементів по індексі.

*template<class T,class Allocator=allocator<T>>class std::vector{…};*

де *T* – тип призначених для збереження даних.

*Allocator* задає розподільник пам'яті, що за замовчуванням є стандартним.

У класі vector визначені наступні конструктори.

explicit **vector**(const Allocator& a=Allocator());

explicit **vector**(size\_type число, const T&значення= T(), const Allocator&a= Allocator());

**vector**(const vector<T,Allocator>&об'єкт);

template<class InIter>**vector**(InIter початок, InIter кінець, const Allocator&a= Allocator());

Перша форма являє собою конструктор порожнього вектора.

В другій формі конструктора вектора число елементів – це число, а кожен елемент дорівнює значенню значення. Параметр значення може бути значенням за замовчуванням.

Третя форма конструктора вектор – це конструктор копіювання.

Четверта форма – це конструктор вектора, що містить діапазон елементів, заданий ітераторами початок і кінець.

Приклад 1.

*vector<int> a;*

*vector<double> x(5);*

*vector<char> c(5,’\*’);*

*vector<int> b(a); //b=a*

Для будь-якого об'єкта, що буде зберігатися у векторі, повинний бути визначений конструктор за замовчуванням. Крім того, для об'єкта повинні бути визначені оператори < і ==.

Для класу вектор визначені наступні оператори порівняння

==, <, <=, !=, >, >=.

Крім цього, для класу vector визначається оператор індексу [].

1. Нові елементи можуть включатися за допомогою функцій

*insert(), push\_back(), resize(), assign().*

1. Існуючі елементи можуть віддалятися за допомогою функцій

*erase(), pop\_back(), resize(), clear().*

1. Доступ до окремих елементів здійснюється за допомогою ітераторів

*begin(), end(), rbegin(), rend(),*

1. Маніпулювання контейнером, сортування, пошук у ньому, тощо, можливо за допомогою глобальних функцій файлу-заголовка <algorithm.h>.

Приклад 2.

*#include<iostream.h>*

*#include<vector.h>*

*using namespace std;*

*void main()*

*{vector<int> v;*

*int i;*

*for(i=0;i<10;i++)v.push\_back(i);*

*cout<<“size=”<<v.size()<<“\n”;*

*for(i=0;i<10;i++)cout<<v[i]<<“ ”;*

*cout<<endl;*

*for(i=0;i<10;i++)v[i]=v[i]+v[i];*

*for(i=0;i<v.size();i++)cout<<v[i]<<“ ”;*

*cout<<endl;*

*}*

Приклад 3. Доступ до вектора через ітератор.

*#include<iostream.h>*

*#include<vector.h>*

*using namespace std;*

*void main()*

*{vector<int> v;*

*int i;*

*for(i=0;i<10;i++)v.push\_back(i);*

*cout<<“size=”<<v.size()<<“\n”;*

*vector<int>::iterator p=v.begin();*

*while(p!=v.end())*

*{cout<<\*p<<” “;p++;}*

*}*

Приклад 4. Вставка і видалення елементів

*#include<iostream.h>*

*#include<vector.h>*

*using namespace std;*

*void main()*

*{vector<int> v(5,1);*

*int i;*

*//вивід*

*for(i=0;i<5;i++)cout<<v[i]<<“ ”;*

*cout<<endl;*

*vector<int>::iterator p=v.begin();*

*p+=2;*

*//вставити 10 елементів зі значенням 9*

*v.insert(p,10,9);*

*//вивід*

*p=v.begin();*

*while(p!=v.end())*

*{cout<<\*p<<” “;p++;}*

*//видалити вставлені елементи*

*p=v.begin();*

*p+=2;*

*v.erase(p,p+10);*

*//вивід*

*p=v.begin();*

*while(p!=v.end())*

*{cout<<\*p<<” “;p++;}*

*}*

Приклад 5. Вектор містить об'єкти класу користувача.

*#include<iostream.h>*

*#include<vector.h>*

*#include”student.h”*

*using namespace std;*

*void main()*

*{vector<STUDENT> v(3);*

*int i;*

*v[0]=STUDENT(“Іванов”,45.9);*

*v[1]=STUDENT(“Петров”,30.4);*

*v[2]=STUDENT(“Сідоров”,55.6);*

*//вивід*

*for(i=0;i<3;i++)cout<<v[i]<<“ ”;*

*cout<<endl;*

*}*

**Асоціативні контейнери (масиви)**

Асоціативний масив містить пари значень. Знаючи одне значення, що називається **ключем** (key), ми можемо одержати доступ до іншого, oj yfpbdf’nmcz **відображеним значенням** (mapped value).

Асоціативний масив можна представити як масив, для якого індекс не обов'язково повинний мати цілочисельний тип:

V& operator[](const K&) повертає посилання на V, що відповідає K.

Асоціативні контейнери – це узагальнення поняття асоціативного масиву.

Асоціативний контейнер **map** - це послідовність пар (ключ, значення), що забезпечує швидке одержання значення за ключем. Контейнер map надає двонаправлені ітератори.

Асоціативний контейнер map вимагає, щоб для типів ключа існувала операція “<”. Він зберігає свої елементи відсортованими за ключем так, що перебір відбувається один по одному.

Специфікація шаблона для класу map:

template<class Key,class,class Comp=less<Key>,class Allocator=allocator<pair> >

class std::map

У класі map визначені наступні конструктори:

explicit **map**(const Comp& c=Comp(),const Allocator& a=Allocator());

**map**(const map<Key,T,Comp,Allocator>& ob);

template<class InIter> **map**(InIter first,InIter last,const Comp& c=Comp(),const Allocator& a=Allocator());

Перша форма являє собою конструктор порожнього асоціативного контейнера. Друга – конструктор копії. Третя – конструктор асоціативного контейнера, що містить діапазон елементів.

Визначено операцію присвоювання:

map& **operator=**(const map&);

Визначено наступні операції: ==, <, <=, !=, >, >=.

У map зберігаються пари ключ/значення у виді об'єктів типу **pair**.

Створювати пари ключ/значення можна не тільки за допомогою конструкторів класу pair, але і за допомогою функції **make\_pair**, що створює об'єкти типу pair, використовуючи типи даних як параметри.

Типова операція для асоціативного контейнера – це асоціативний пошук за допомогою операції індексації([]).

mapped\_type& **operator[]**(const key\_type& K);

Множини **set** можна розглядати як асоціативні масиви, у яких значення не грають ролі, так що ми відслідковуємо тільки ключі.

template<class,class Cmp=less<T>,class Allocator=allocator<T>>class std::set{...};

Множину, як і асоціативний масив, вимагає, щоб для типу T існувала операція “менше” (<). Воно зберігає свої елементи відсортованими, так що перебір відбувається один по одному.

**Алгоритми**

Кожен алгоритм виражається шаблоном функції чи набором шаблонів функцій. Таким чином, алгоритм може працювати з дуже різними контейнерами, що містять значення різноманітних типів. Алгоритми, що повертають ітератор, як правило, для повідомлення про невдачу використовують кінець вхідної послідовності. Алгоритми не виконують перевірки діапазону на їх вході і виході. Коли алгоритм повертає ітератор, це буде ітератор того ж типу, що і був на вході. Алгоритми в STL реалізують більшість розповсюджених універсальних операцій з контейнерами, такі як перегляд, сортування, пошук, вставку і видалення елементів.

Алгоритми визначені в заголовному файлі <algorithm.h>.

Нижче приведені імена деяких найбільше часто користуємих функцій-алгоритмів STL.

1. **Операції, що не модифікують**

**for\_earch()**  Виконує операції для кожного елемента послідовності

**find()** Знаходить перше входження значення в послідовність

**find\_if()** Знаходить першу відповідність предикату в послідовності

**count()** Підраховує кількість входжень значення в послідовність

**count\_if()**  Підраховує кількість виконань предиката в послідовності

**search()**  Знаходить перше входження послідовності як підпослідовності

**search\_n()**  Знаходить n-і входження значення в послідовність

1. **Операції, що модифікують**

**copy()** Копіює послідовність, починаючи з першого елемента

**swap()** Міняє місцями два елементи

**replace()** Замінює елементи з зазначеним значенням

**replace\_if()** Замінює елементи при виконанні предиката

**replace\_copy()** Копіює послідовність, замінючи елементи з зазначеним значенням

**replace\_copy\_if()** Копіює послідовність, замінюючи елементи при виконанні предиката

**fill()** Замінює всі елементи даним значенням

**remove()** Видаляє елементи з даним значенням

**remove\_if()** Видаляє елементи при виконанні предиката

**remove\_copy()** Копіює послідовність, видаляючи елементи з зазначеним значенням

**remove\_copy\_if()** Копіює послідовність, видаляючи елементи при виконанні предиката

**reverse()** Змінює порядок проходження елементів на зворотний

**random\_shuffle()** Переміщає елементи відповідно до випадкового рівномірного розподілу (“тасує” послідовність).

**transform()** Виконує задану операцію над кожним елементом послідовності

**unique()** Видаляє рівні сусідні елементи

**unique\_copy()** Копіює послідовність, видаляючи рівні сусідні елементи

1. **Сортування**

**sort()** Сортує послідовність з гарною середньою ефективністю

**partial\_sort()** Сортує частину послідовності

**stable\_sort()** Сортує послідовність, зберігаючи порядок проходження рівних елементів

**lower\_bound()** Знаходить перше входження значення у відсортованій послідовності

**upper\_bound()** Знаходить перший елемент, більший чим задане значення

**binary\_search()** Визначає, чи є даний елемент у відсортованій послідовності

**merge()** Зливає дві відсортовані послідовності

1. **Робота з множинами**

**includes()**  Перевірка на входження

**set\_union()** Об'єднання множин

**set\_intersection()** Перетин множин

**set\_difference()** Різниця множин

1. **Мінімуми і максимуми**

**min()** менше з двох

**max()** більше з двох

**min\_element()** найменше значення в послідовності

**max\_element()** найбільше значення в послідовності

1. **Перестановки**

**next\_permutation()** Наступна перестановка в лексикографічному порядку

**pred\_permutation()** Попередня перестановка в лексикографічному порядку

**Порядок виконання роботи.**

Написати і налагодити дві програми. Перша програма демонструє використання контейнерних класів для збереження вбудованих типів даних.

Друга програма демонструє використання контейнерних класів для збереження типів даних користувача – обєктів класів**( Задача з ЛР5 , ЛР3 або Задача 1.ЛР1** ).

Також програма демонструє використання алгоритмів STL (4-6 викликів різних методів.)

**У програмі №1** виконати наступне:

1.Створити об'єкт-контейнер відповідно до варіанта завдання і заповнити його даними, тип яких визначається варіантом завдання.

2.Переглянути контейнер.

3.Змінити контейнер, видаливши з нього одні елементи і замінивши інші.

4.Переглянути контейнер, використовуючи для доступу до його елементів ітератори.

**У програмі №2** виконати наступне:

1.Створити контейнер, що містить об'єкти типу користувача. Тип 1 контейнера вибирається відповідно до варіанта завдання.

2.Відсортувати його по спаданню елементів.

3.Переглянути контейнер.

4.Використовуючи придатний алгоритм, знайти в контейнері елемент, що задовольняє заданій умові.

5.Перемістити елементи, що задовольняють заданій умові в іншій (попередньо порожній) контейнер. Тип другого контейнера визначається варіантом завдання.

6.Переглянути другий контейнер.

7.Відсортувати перший і другий контейнери по зростанню елементів.

8.Переглянути їх.

**9.ЗАПИСАТИ ОБЄКТИ У ФАЙЛ ТА ПРОЧИТАТИ З НЬОГО.**

**Методичні вказівки.**

1. Проект повинний містити 2 цільових вузли (по числу програм).

2.Як тип даних користувача використовувати клас користувача лабораторної роботи №1. або ЛР5.(С++) ЛР6(С#)

3.При створенні контейнерів у програмі №2 об'єкти завантажувати з потоку, для чого використовувати програми запису і читання потоку з лабораторної роботи №1.

3.Для вставки і видалення елементів контейнера в програмі №2 використовувати відповідні операції, визначені в класі контейнера.

4.Для створення другого контейнера в програмі №2 можна використовувати або алгоритм **remove\_copy\_if**, або визначити свій алгоритм **copy\_if**, якого немає в STL.

5.Для пошуку елемента в колекції можна використовувати алгоритм **find\_if**, або **for\_each**, або **binary\_search**, якщо контейнер відсортований.

6. Для порівняння елементів при сортуванні по зростанню використовується операція <, що повинна бути перевантажена в класі користувача. Для сортування по спаданню варто написати функцію **comp** і використовувати другу версію алгоритму **sort.**

7.Умови пошуку і заміни елементів вибираються самостійно і для них пишеться функція-предикат.

8.Для вводу-виводу об'єктів класу користувача варто перевантажити операції “>>” і “<<”.

9.Деякі алгоритми можуть не підтримувати використовувані у вашій програмі контейнери. Наприклад, алгоритм **sort** не підтримує контейнери, що не мають ітераторів довільного доступу. У цьому випадку варто написати свій алгоритм. Наприклад, для стека алгоритм сортування може виконуватися в такий спосіб: переписати стек у вектор, відсортувати вектор, переписати вектор у стек.

10.При переміщенні елементів асоціативного контейнера в неасоціативний, переміщаються тільки дані (ключі не переміщаються). І навпаки, при переміщенні елементів неасоціативного контейнера в асоціативний повинний бути сформований ключ.

**Зміст звіту.**

1.Назва, мета роботи

2.Постановка задач.

3.Визначення класу користувача.

4.Визначення використовуваних у програмах компонентних функцій для роботи з контейнером, включаючи конструктори.

5.Пояснення цих функцій.

6.Пояснення використовуваних у програмах алгоритмів STL.

7.Визначення і пояснення, використовуваних предикатів і функцій порівняння

**Приклади виконаних завдань**

**Варіант 11**

**Завдання 1**

**Project\_1.cpp**

#include "stdafx.h"

#include <vector>

#include <map>

#include <iostream>

#include <list>

#include "Person.h"

#include <algorithm>

using namespace std;

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

{

vector<double> v;

vector<double>::iterator myIt;

int n;

cin>>n;

for(int i=0;i<n;i++) v.push\_back(rand()%10);

for(int i=0;i<v.size();i++) {std::cout<<v[i]<<" ";};

cout<<endl;

myIt=v.begin();

double total=0;

while(myIt!=v.end()) {

total+=\*myIt++;

};

cout<<total<<endl;

sort(v.begin(),v.end());

for(int i=0;!v.empty();i++) {std::cout<<v.back()<<" "; v.pop\_back();};

system("pause");

return 0;

}

**Завдання 2**

**Project\_2.cpp**

#include "stdafx.h"

#include <iostream>;

#include <map>;

#include <algorithm>

#include "Person.h"

using namespace std;

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

map<int,Person> myMap;

int n;

Person a;

cin>>n;

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

a.setPerson();

myMap.insert(pair<int,Person>(i,a));

}

for (std::map<int, Person>::iterator it = myMap.begin(); it != myMap.end(); ++it)

{

it->second.printPerson();

}

system("pause");

return 0;

}

**Person.h**

#pragma once

class Person

{

public:

char name[20];

int age;

char pol[20];

public:

Person();

Person(int age);

Person(const Person&);

~Person(void);

char\* getName();

int getAge();

void setPerson();

void setAge(int old);

void printPerson();

};

**Person.cpp**

#include "StdAfx.h"

#include "Person.h"

#include <iostream>

using namespace std;

Person::Person() :age(0)

{

}

Person::Person(int old) : age(old)

{

}

Person::Person(const Person&)

{

};

Person::~Person(void)

{

cout << "Vuklukano destruktor " << this << endl;

};

char\* Person::getName()

{

return name;

};

int Person::getAge()

{

return age;

};

void Person::setAge(int old)

{

age=old;

};

void Person::printPerson()

{

cout << "Name:"<< name << "\t age:" << age << "\t pol:" << pol << endl;

}

void Person::setPerson()

{

cout << "Name:";

cin>> name;

cout << "age:";

cin >> age;

cout << " pol:";

cin >>pol;

}

**Додаток. Варіанти завдань.**

|  |  |  |  |
| --- | --- | --- | --- |
| № | ПЕРШИЙ  КОНТЕЙНЕР | ДРУГИЙ  КОНТЕЙНЕР | ВБУДОВАНИЙ ТИП ДАНИХ |
| 1 | vector | list | int |
| 2 | list | deque | long |
| 3 | deque | stack | float |
| 4 | stack | queue | double |
| 5 | queue | vector | char |
| 6 | vector | stack | string |
| 7 | map | list | long |
| 8 | multimap | deque | float |
| 9 | set | stack | int |
| 10 | multiset | queue | char |
| 11 | vector | map | double |
| 12 | list | set | int |
| 13 | deque | multiset | long |
| 14 | stack | vector | float |
| 15 | queue | map | int |
| 16 | priority\_queue | stack | char |
| 17 | map | queue | char |
| 18 | multimap | list | int |
| 19 | set | map | char |
| 20 | multiset | vector | int |
| 21 | deque | map | int |

Постановка задачі для С

**С#-Колекції. Серіалізація.Десеріалізація.**

**Неузагальнені колекції**

Неузагальнені чи прості колекції визначені у просторі імен System.Collections. Їх особливість полягає в тому, що їхня функціональність, функціональні можливості описуються в інтерфейсах, які також знаходяться в цьому просторі імен.

Розглянемо основні інтерфейси:

• **IEnumerable**: визначає метод GetEnumerator. Цей метод повертає перечислювач - тобто деякий об'єкт, що реалізує інтерфейс IEumerator.

• **IEumerator:** реалізація цього інтерфейсу дозволяє перебирати елементи колекції за допомогою циклу foreach

• **ICollection:** є основою для всіх неузагальнених колекцій, що визначає основні методи та властивості для всіх неузагальнених колекцій (наприклад, метод CopyTo та властивість Count). Цей інтерфейс успадкований від інтерфейсу IEnumerable, завдяки чому базовий інтерфейс також реалізується всіма класами неузагальнених колекцій

• **IList:** дозволяє отримувати елементи колекції по порядку. Також визначає ряд методів для маніпуляції елементами: Add (додавання елементів), Remove/RemoveAt (видалення елемента) та інших.

• **IComparer:** визначає метод int Compare(object x, object y) порівняння двох об'єктів

• **IDictionary:** визначає поведінку колекції, при якій вона повинна зберігати об'єкти у вигляді пар ключ-значення: для кожного об'єкта визначається унікальний ключ і цьому ключу відповідає певне значення

• **IDictionaryEnumerator:** визначає методи та властивості для перечислювача словника

• **IEqualityComparer:** визначає два методи Equals та GetHashCode, за допомогою яких два об'єкти порівнюються на предмет рівності

• **IStructuralComparer:** визначає метод Compare для структурного порівняння двох об'єктів: за такого порівняння порівнюються не посилання об'єкти, а безпосередній вміст об'єктів

**• IStructuralEquatable**: дозволяє провести структурну рівність двох об'єктів. Як і у випадку з інтерфейсом IStructuralComparer порівнюється вміст двох об'єктів

Ці інтерфейси реалізуються такими класами колекцій у просторі імен System.Collections:

1. **ArrayList:** клас простої колекції об'єктів. Реалізує інтерфейси IList, ICollection, IEnumerable

2. **BitArray**: клас колекції, що містить масив бітових значень. Реалізує інтерфейси ICollection, IEnumerable

3. **Hashtable:** клас колекції, що представляє хеш-таблицю та зберігає набір пар "ключ-значення"

4. **Queue:** клас черги об'єктів, що працює за алгоритмом FIFO ("перший увійшов-перший вийшов"). Реалізує інтерфейси ICollection, IEnumerable

5. **SortedList:** клас колекції, що зберігає набори пар "ключ-значення", відсортованих за ключом. Реалізує інтерфейси ICollection, IDictionary, IEnumerable

**6. Stack: клас** стека

**Узагальнені колекції**

У темі Узагальнені типи ми вже говорили про узагальнення: як створювати та використовувати узагальнені класи, роль універсального параметра T. Узагальнені колекції – це ті ж узагальнені класи. І знову ж таки їх використання перед неузагальненими колекціями має ті ж переваги: підвищення продуктивності (не треба витрачати час на упаковку та розпакування об'єкта) та підвищена типобезпека.

**Класи узагальнених колекцій знаходяться у просторі імен System.Collections.Generic.** Функціонал колекцій також здебільшого описується в узагальнених інтерфейсах.

Тільки інтерфейси узагальнених колекцій відрізняються від неузагальнених двійників як наявністю універсального параметра T, а й самої функціональністю. Розглянемо основні інтерфейси узагальнених колекцій:

• IEnumerable<T>: визначає метод GetEnumerator, за допомогою якого можна отримувати елементи будь-якої колекції

. Реалізація даного інтерфейсу дозволяє перебирати елементи колекції за допомогою циклу foreach

• IEumerator<T>: визначає методи, за допомогою яких потім можна отримати вміст колекції по черзі

• ICollection<T>: представляє ряд загальних властивостей та методів для всіх неузагальнених колекцій (наприклад, методи CopyTo, Add, Remove, Contains, властивість Count)

• IList<T>: надає функціонал для створення послідовних списків

• IComparer<T>: визначає метод Compare для порівняння двох однотипних об'єктів

• IDictionary<TKey, TValue>: визначає поведінку колекції, при якій вона повинна зберігати об'єкти у вигляді пар ключ-значення: для кожного об'єкта визначається унікальний ключ типу, вказаного в параметрі TKey, і цьому ключу відповідає певне значення, що має тип, вказаний у параметрі TValue

• IEqualityComparer<T>: визначає методи, за допомогою яких два однотипні об'єкти порівнюються на предмет рівності

Ці інтерфейси реалізуються такими класами колекцій у просторі імен System.Collections.Generic:

1. **List<T>:** клас, який представляє послідовний список. Реалізує інтерфейси IList<T>, ICollection<T>, IEnumerable<T>

2. **Dictionary<TKey, TValue>:** клас колекції, що зберігає набори пар "ключ-значення". Реалізує інтерфейси ICollection<T>, IEnumerable<T>, IDictionary<TKey, TValue>

3. **LinkedList<T>:** клас двозв'язаного списку. Реалізує інтерфейси ICollection<T> та IEnumerable<T>

4. **Queue<T>:** клас черги об'єктів, що працює за алгоритмом LIFO("перший увійшов-перший вийшов"). Реалізує інтерфейси ICollection, IEnumerable<T>

5. **SortedSet<T>:** клас відсортованої колекції однотипних об'єктів. Реалізує інтерфейси ICollection<T>, ISet<T>, IEnumerable<T>

6. **SortedList<TKey**, TValue>: клас колекції, що зберігає набори пар "ключ-значення", відсортованих за ключом. Реалізує інтерфейси ICollection<T>, IEnumerable<T>, IDictionary<TKey, TValue>

7. **SortedDictionary<TKey, TValue>:** клас колекції, що зберігає набори пар "ключ-значення", відсортованих за ключом. Загалом схожий на клас SortedList<TKey, TValue>, основні відмінності полягають лише у використанні пам'яті та швидкості вставки та видалення

8. **Stack<T>:** клас стека однотипних об'єктів. Реалізує інтерфейси ICollection<T> та IEnumerable<T>

Більшість узагальнених класів колекцій дублюють неузагальнені класи колекцій. Але якщо вам не потрібно зберігати об'єкти різних типів, краще використовувати узагальнені колекції.

Раніше ми зберігали інформацію в текстові файли, а також торкнулися збереження нескладних структур в бінарні дані. Але нерідко подібних механізмів виявляється недостатньо особливо для збереження складних об'єктів. З цією проблемою покликаний впоратися механізм серіалізації. **Серіалізація** представляє процес перетворення будь-якого об'єкта в потік байтів. Після перетворення ми можемо цей потік байтів або записати на диск або зберегти його тимчасово в пам'яті. А при необхідності можна виконати зворотній процес - **десеріалізацію**, тобто отримати з потоку байтів раніше збережений об'єкт.

**Атрибут Serializable**

Щоб об'єкт певного класу можна було серіалізувати, треба цей клас помітити атрибутом **Serializable**:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | [Serializable]  class Person  {      public string Name { get; set; }      public int Year { get; set; }        public Person(string name, int year)      {          Name = name;          Year = year;      }  } |

При відсутності даного атрибута об'єкт Person не зможе бути серіалізовано, і при спробі серіализації буде викинуто виключення SerializationException.

Якщо ми не хочемо, щоб якийсь член класу серіалізовані, то ми його помічаємо атрибутом **NonSerialized**:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | [Serializable]  class Person  {      public string Name { get; set; }      public int Year { get; set; }        [NonSerialized]      public string AccNumber { get; set; }        public Person(string name, int year)      {          Name = name;          Year = year;      }  } |

При спадкуванні подібного класу, слід враховувати, що атрибут Serializable автоматично не успадковується. І якщо ми хочемо, щоб похідний клас також міг би бути серіалізовані, то знову ж таки ми застосовуємо до нього атрибут:

|  |  |
| --- | --- |
| 1  2 | [Serializable]  class Worker : Person |

**Формат серіалізації**

Хоча серіалізація є перетворенням об'єкта в певний набір байтів, але в дійсності тільки бінарним форматом вона не обмежується. Отже, в .NET можна використовувати такі формати:

* бінарний
* SOAP
* xml
* JSON

Для кожного формату передбачений свій клас: для серіалізації в бінарний формат - клас **BinaryFormatter**, для формата SOAP - клас **SoapFormatter**, для xml - **XmlSerializer**, для json - **DataContractJsonSerializer**.

Для класів BinaryFormatter и SoapFormatter сам функціонал серіализації визначений в інтерфейсі **IFormatter**:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | public interface IFormatter  {      SerializationBinder Binder { get; set;}      StreamingContext Context { get; set;}      ISurrogateSelector SurrogateSelector  { get; set;}      object Deserialize (Stream serializationStream);      void Serialize (Stream serializationStream, object graph);  } |

Хоча класи BinaryFormatter і SoapFormatter по-різному реалізують даний інтерфейс, але загальний функціонал буде той же: для серіалізації буде використовуватися метод Serialize, який в якості параметрів приймає потік, куди поміщає серіалізовані дані (наприклад, бінарний файл), і об'єкт, який треба серіалізувати. А для десеріалізації буде застосовуватися метод Deserialize, який в якості параметра приймає потік з серіалізованими даними.

Клас XmlSerializer не реалізує інтерфейс IFormatter і по функціональності в цілому дещо відрізняється від BinaryFormatter і SoapFormatter, але і він також надає для серіалізації метод Serialize, а для десеріалізації Deserialize. І в цьому плані дуже легко при необхідності перейти від одного способу сериализации до іншого.

**Бінарна серіалізація. BinaryFormatter**

Для бінарної серіалізації приміняється клас BinaryFormatter:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50 | using System;  using System.IO;  using System.Runtime.Serialization.Formatters.Binary;    namespace Serialization  {      [Serializable]      class Person      {          public string Name { get; set; }          public int Age { get; set; }            public Person(string name, int age)          {              Name = name;              Age = age;          }      }        class Program      {          static void Main(string[] args)          {              // об'єкт для серіалізації              Person person = new Person("Tom", 29);              Console.WriteLine("Об'єкт створено");                // створюємо объект BinaryFormatter              BinaryFormatter formatter = new BinaryFormatter();              // отримуємо потік, куди будемо записувати серіалізований об'єкт              using (FileStream fs = new FileStream("people.dat", FileMode.OpenOrCreate))              {                  formatter.Serialize(fs, person);                    Console.WriteLine("Об’єкт серіалізовано");              }                // десеріалізація із файлу people.dat              using (FileStream fs = new FileStream("people.dat", FileMode.OpenOrCreate))              {                  Person newPerson = (Person)formatter.Deserialize(fs);                    Console.WriteLine("Об’єкт десеріалізовано");                  Console.WriteLine("Ім’я: {0} --- Вік: {1}", newPerson.Name, newPerson.Age);              }                Console.ReadLine();          }      }  } |

Так як клас BinaryFormatter визначено в просторі імен System.Runtime.Serialization.Formatters.Binary, то на самому початку підключаємо його.

У нас є простенький клас Person, який оголошений з атрибутом Serilizable. Завдяки цьому його об'єкти будуть доступні для серіалізації.

Далі створюємо об'єкт BinaryFormatter: BinaryFormatter formatter = new BinaryFormatter ();

Потім послідовно виконуємо серіалізацію і десеріалізацію. Для обох операцій нам потрібен потік, в який або зберігати, або з якого зчитувати дані. Даний потік являє об'єкт FileStream, який записує потрібний нам об'єкт Person в файл people.dat.

Серіалізація одним методом formatter.Serialize (fs, person) додає всі дані про об'єкт Person в файл people.dat.

При десеріалізації нам потрібно ще перетворити об'єкт, що повертається функцією Deserialize, до типу Person: (Person) formatter.Deserialize (fs). Як ви бачите, серіалізація значно спрощує процес збереження об'єктів в бінарну форму в порівнянні, наприклад, з використанням зв'язки класів BinaryWriter / BinaryReader.

Хоча ми взяли лише один об'єкт Person, але так само ми можемо використовувати і масив подібних об'єктів, список або іншу колекцію, до якої застосовується атрибут Serializable. Подивимося на прикладі масиву:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | Person person1 = new Person("Tom", 29);  Person person2 = new Person("Bill", 25);  // масив для серіалізації  Person[] people = new Person[] { person1, person2 };    BinaryFormatter formatter = new BinaryFormatter();    using (FileStream fs = new FileStream("people.dat", FileMode.OpenOrCreate))  {      // серіалізуємо весь масив people      formatter.Serialize(fs, people);        Console.WriteLine("Об’єкт серіалізовано");  }    // десеріалізація  using (FileStream fs = new FileStream("people.dat", FileMode.OpenOrCreate))  {      Person[] deserilizePeople = (Person[])formatter.Deserialize(fs);        foreach (Person p in deserilizePeople)      {          Console.WriteLine("Ім’я: {0} --- Вік: {1}", p.Name, p.Age);      }  } |

**Серіалізація в формат SOAP. SoapFormatter**

Протокол SOAP (Simple Object Access Protocol) являє простий протокол для обміну даними між різними платформами. При такій серіалізації дані упакуються в конверт SOAP, дані в якому мають вигляд xml-подібного документа. Подивимося на прикладі.

Перш ніж використовувати клас SoapFormatter, нам треба додати в проект збірку System.Runtime.Serialization.Formatters.Soap.dll. Після цього нам стане доступним функціональність SoapFormatter:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51 | using System;  using System.IO;  using System.Runtime.Serialization.Formatters.Soap;    namespace Serialization  {      [Serializable]      class Person      {          public string Name { get; set; }          public int Age { get; set; }            public Person(string name, int age)          {              Name = name;              Age = age;          }      }      class Program      {          static void Main(string[] args)          {              Person person = new Person("Tom", 29);              Person person2 = new Person("Bill", 25);              Person[] people = new Person[] { person, person2 };                // створюємо об’єкт SoapFormatter              SoapFormatter formatter = new SoapFormatter();              // отримуємо потік, куди будео записувати серіалізований об’єкт              using (FileStream fs = new FileStream("people.soap", FileMode.OpenOrCreate))              {                  formatter.Serialize(fs, people);                    Console.WriteLine("Об’єкт серіалізовано");              }                // десеріалізація              using (FileStream fs = new FileStream("people.soap", FileMode.OpenOrCreate))              {                  Person[] newPeople = (Person[])formatter.Deserialize(fs);                    Console.WriteLine("Об’єкт десеріалізовано");                  foreach (Person p in newPeople)                  {                      Console.WriteLine("Ім’я: {0} --- Вік: {1}", p.Name, p.Age);                  }              }              Console.ReadLine();          }      }  } |

Принцип використання SoapFormatter схожий на розглянуту в минулій темі бінарну серіалізацію. Тут також створюється потік, який записує дані в файл people.soap. Для серіалізації використовується метод formatter.Serialize (fs, people), який використовує потік і об'єкт для серіалізації.

При десеріалізації зчитуємо раніше збережені об'єкти і перетворимо їх до потрібного нам об'єкту в методі Deserialize: Person [] newPeople = (Person []) formatter.Deserialize (fs)

Після серіалізації всі дані будуть збережені в файл people.soap, який буде мати такий зміст:

<SOAP-ENV:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:clr="http://schemas.microsoft.com/soap/encoding/clr/1.0" SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">

<SOAP-ENV:Body>

<SOAP-ENC:Array SOAP-ENC:arrayType="a1:Person[2]" xmlns:a1="http://schemas.microsoft.com/clr/nsassem/Serialization/Serialization%2C%20Version%3D1.0.0.0%2C%20Culture%3Dneutral%2C%20PublicKeyToken%3Dnull">

<item href="#ref-3"/>

<item href="#ref-4"/>

</SOAP-ENC:Array>

<a1:Person id="ref-3" xmlns:a1="http://schemas.microsoft.com/clr/nsassem/Serialization/Serialization%2C%20Version%3D1.0.0.0%2C%20Culture%3Dneutral%2C%20PublicKeyToken%3Dnull">

<\_x003C\_Name\_x003E\_k\_\_BackingField id="ref-5">Tom</\_x003C\_Name\_x003E\_k\_\_BackingField>

<\_x003C\_Age\_x003E\_k\_\_BackingField>29</\_x003C\_Age\_x003E\_k\_\_BackingField>

</a1:Person>

<a1:Person id="ref-4" xmlns:a1="http://schemas.microsoft.com/clr/nsassem/Serialization/Serialization%2C%20Version%3D1.0.0.0%2C%20Culture%3Dneutral%2C%20PublicKeyToken%3Dnull">

<\_x003C\_Name\_x003E\_k\_\_BackingField id="ref-6">Bill</\_x003C\_Name\_x003E\_k\_\_BackingField>

<\_x003C\_Age\_x003E\_k\_\_BackingField>25</\_x003C\_Age\_x003E\_k\_\_BackingField>

</a1:Person>

</SOAP-ENV:Body>

</SOAP-ENV:Envelope>

**Серіалізація в XML. XmlSerializer**

Для серіалізації об'єктів в файли xml використовується клас XmlSerializer. Він стоїть дещо окремо від інших раніше розглянутих класів серіалізацій, тому робота з ним буде трохи відрізнятися.

По-перше, XmlSerializer передбачає деякі обмеження. Наприклад, клас, який підлягає серіалізації, повинен мати стандартний конструктор без параметрів. Також серіалізації підлягають тільки відкриті члени. Якщо в класі є поля або властивості з модифікатором private, то при серіалізації вони будуть ігноруватися.

По-друге, XmlSerializer вимагає вказівки типу:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55 | using System;  using System.IO;  using System.Xml.Serialization;    namespace Serialization  {      // клас і його члени об'явлені як public      [Serializable]      public class Person      {          public string Name { get; set; }          public int Age { get; set; }            // стандартний конструктор без параметрів          public Person()          { }            public Person(string name, int age)          {              Name = name;              Age = age;          }      }      class Program      {          static void Main(string[] args)          {              // об’єкт для серіалізації              Person person = new Person("Tom", 29);              Console.WriteLine("Об’єкт створено");                // передаємо в конструктор тип класу              XmlSerializer formatter = new XmlSerializer(typeof(Person));                // отримуємо потік, куди будемо записувати серіалізованьй об'єкт              using (FileStream fs = new FileStream("persons.xml", FileMode.OpenOrCreate))              {                 formatter.Serialize(fs, person);                    Console.WriteLine("Об'єкт серіалізовано");              }                // десеріалізація              using (FileStream fs = new FileStream("persons.xml", FileMode.OpenOrCreate))              {                  Person newPerson = (Person)formatter.Deserialize(fs);                    Console.WriteLine("Об'єкт десеріалізовано");                  Console.WriteLine("Ім’я: {0} --- Вік: {1}", newPerson.Name, newPerson.Age);              }                Console.ReadLine();          }      }  } |

Отже, клас Person загальнодоступний і має загальнодоступні властивості, тому він може серіалізуватісь. При створенні об'єкта XmlSerializer передаємо в конструктор тип класу.

І, як і з іншими класами-серіалзіаторами, метод Serialize додає дані в файл persons.xml. А метод Deserialize витягує їх звідти.

Якщо ми відкриємо файл persons.xml, то побачимо зміст нашого об'єкта:

|  |  |
| --- | --- |
| 1  2  3  4  5 | <?xml version="1.0"?>  <Person xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>" xmlns:xsd="<http://www.w3.org/2001/XMLSchema>">    <Name>Tom</Name>    <Age>29</Age>  </Person> |

Так само ми можемо серіалізувати масив або колекцію об'єктів, але головна вимога полягає в тому, щоб в них був визначений стандартний конструктор:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | Person person1 = new Person("Tom", 29);  Person person2 = new Person("Bill", 25);  Person[] people = new Person[] { person1, person2 };    XmlSerializer formatter = new XmlSerializer(typeof(Person[]));    using (FileStream fs = new FileStream("people.xml", FileMode.OpenOrCreate))  {      formatter.Serialize(fs, people);  }    using (FileStream fs = new FileStream("people.xml", FileMode.OpenOrCreate))  {      Person[] newpeople = (Person[])formatter.Deserialize(fs);        foreach (Person p in newpeople)      {          Console.WriteLine("Имя: {0} --- Возраст: {1}", p.Name, p.Age);      }  } |

Але це був простий об'єкт. Однак з більш складними за складом об'єктами працювати так само просто. Наприклад:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65 | using System;  using System.IO;  using System.Xml.Serialization;    namespace Serialization  {      [Serializable]      public class Person      {          public string Name { get; set; }          public int Age { get; set; }          public Company Company { get; set; }            public Person()          { }            public Person(string name, int age, Company comp)          {              Name = name;              Age = age;              Company = comp;          }      }        [Serializable]      public class Company      {          public string Name { get; set; }            // стандартний конструктор без параметрів          public Company() { }            public Company(string name)          {              Name = name;          }      }      class Program      {          static void Main(string[] args)          {              Person person1 = new Person("Tom", 29, new Company("Microsoft"));              Person person2 = new Person("Bill", 25, new Company("Apple"));              Person[] people = new Person[] { person1, person2 };                XmlSerializer formatter = new XmlSerializer(typeof(Person[]));                using (FileStream fs = new FileStream("people.xml", FileMode.OpenOrCreate))              {                 formatter.Serialize(fs, people);              }                using (FileStream fs = new FileStream("people.xml", FileMode.OpenOrCreate))              {                  Person[] newpeople = (Person[])formatter.Deserialize(fs);                    foreach (Person p in newpeople)                  {                      Console.WriteLine("Ім’я: {0} --- Вік: {1} --- Компанія: {2}", p.Name, p.Age, p.Company.Name);                  }              }              Console.ReadLine();          }      }  } |

Клас Person містить властивість Company, яке буде зберігати об'єкт класу Company. Члени класу Company оголошуються з модифікатором public, крім того також присутній стандартний конструктор без параметрів. У підсумку після серіалізації ми отримаємо наступний xml-документ:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | <?xml version="1.0"?>  <ArrayOfPerson xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>" xmlns:xsd="<http://www.w3.org/2001/XMLSchema>">    <Person>      <Name>Tom</Name>      <Age>29</Age>      <Company>        <Name>Microsoft</Name>      </Company>    </Person>    <Person>      <Name>Bill</Name>      <Age>25</Age>      <Company>        <Name>Apple</Name>      </Company>    </Person>  </ArrayOfPerson> |

**Серіалізація в JSON. DataContractJsonSerializer**

Для серіалізації об'єктів в формат JSON в просторі System.Runtime.Serialization.Json визначено клас DataContractJsonSerializer. Щоб задіяти цей клас, в проект треба додати збірку System.Runtime.Serialization.dll. Для запису об'єктів в json-файл в цьому класі є метод WriteObject (), а для читання раніше серіалізованих об'єктів - метод ReadObject (). Розглянемо їх застосування.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51 | using System;  using System.IO;  using System.Runtime.Serialization.Json;  using System.Runtime.Serialization;    namespace Serialization  {      [DataContract]      public class Person      {          [DataMember]          public string Name { get; set; }          [DataMember]          public int Age { get; set; }            public Person(string name, int year)          {              Name = name;              Age = year;          }      }      class Program      {          static void Main(string[] args)          {              // об'єкт для серіалізації              Person person1 = new Person("Tom", 29);              Person person2 = new Person("Bill", 25);              Person[] people = new Person[] { person1, person2 };                DataContractJsonSerializer jsonFormatter = new DataContractJsonSerializer(typeof(Person[]));                using (FileStream fs = new FileStream("people.json", FileMode.OpenOrCreate))              {                  jsonFormatter.WriteObject(fs, people);              }                using (FileStream fs = new FileStream("people.json", FileMode.OpenOrCreate))              {                  Person[] newpeople = (Person[])jsonFormatter.ReadObject(fs);                    foreach (Person p in newpeople)                  {                      Console.WriteLine("Ім’я: {0} --- Вік: {1}", p.Name, p.Age);                  }              }                Console.ReadLine();          }      }  } |

Щоб позначити клас як серіалізуємий, до нього застосовується атрибут DataContract, а до всіх його серіалізуємих властивостей - атрибут DataMember.

Метод WriteObject () приймає два параметри: файловий потік FileStream і об'єкт, який треба серіалізувати - в даному випадку масив об'єктів Person. А метод ReadObject () приймає як параметр файловий потік, який представляє файл у форматі json.

Якщо ми відкриємо файл people.json, то побачимо зміст нашого об'єкта:

|  |  |
| --- | --- |
| 1  2  3  4  5 | [{      "Age":29,"Name":"Tom"  },{      "Age":25,"Name":"Bill"  }] |

Подібним чином можна серіалізувати / десеріалізувати більш складні об'єкти:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67 | using System;  using System.IO;  using System.Runtime.Serialization.Json;  using System.Runtime.Serialization;    namespace Serialization  {      [DataContract]      public class Person      {          [DataMember]          public string Name { get; set; }          [DataMember]          public int Age { get; set; }          [DataMember]          public Company Company { get; set; }            public Person()          { }            public Person(string name, int age, Company comp)          {              Name = name;              Age = age;              Company = comp;          }      }        public class Company      {          public string Name { get; set; }            public Company() { }            public Company(string name)          {              Name = name;          }      }      class Program      {          static void Main(string[] args)          {              Person person1 = new Person("Tom", 29, new Company("Microsoft"));              Person person2 = new Person("Bill", 25, new Company("Apple"));              Person[] people = new Person[] { person1, person2 };                DataContractJsonSerializer jsonFormatter = new DataContractJsonSerializer(typeof(Person[]));                using (FileStream fs = new FileStream("people.json", FileMode.OpenOrCreate))              {                  jsonFormatter.WriteObject(fs, people);              }                using (FileStream fs = new FileStream("people.json", FileMode.OpenOrCreate))              {                  Person[] newpeople = (Person[])jsonFormatter.ReadObject(fs);                    foreach (Person p in newpeople)                  {                      Console.WriteLine("Ім’я: {0} --- Вік: {1} --- Компанія: {2}", p.Name, p.Age, p.Company.Name);                  }              }              Console.ReadLine();          }      }  } |

В результаті програма створить наступний json-файл:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | [{      "Age":29,      "Company":{"Name":"Microsoft"},      "Name":"Tom"  },{      "Age":25,      "Company":{"Name":"Apple"},      "Name":"Bill"  }] |

**Порядок виконання роботи.**

Нижче наведений приклад використання параметризованої функції swap(T& x, T& y), що здійснює обмін значеннями між двома її параметрами:

*Приклад 1.*

#include "stdio.h"  
template <class T> void swap(T& x, T& y);  
int main()  
{  
   int a=5,b=10;  
   float c=7,d=14;  
   char ch='a',chh='A';  
   swap(a,b); // виклик для цілих аргументів  
   printf("a=%d, b=%d\n",a,b);  
   swap <float>(c,d); // виклик для float аргументів  
   printf("c=%f, d=%f\n",c,d);  
   swap <char>(ch,chh); // виклик для char аргументів  
   printf("ch=%c, chh=%c\n",ch,chh);  
   return 0;  
}  
  
// визначення параметризованої функції   
template <class T> void swap(T& x, T& y)  
{  
    T temp=x;  
    x=y;  
    y=temp;  
}

Програма відкривається прототипом шаблонної функції. Аналізуючи приклад, зауважимо, що для інстанціювання шаблону застосовуються дві форми.

Конкретний тип для цього визначається компілятором автоматично, виходячи з типів параметрів у місці виклику функції (у наведеному вище прикладі це тип *int*), або задається явним способом (у прикладі це підстановка типів *float*та *char*).

Якби *swap()* була звичайною функцією, то потрібною була б її реалізація. Оскільки це шаблонна функція, компілятор сам реалізуватиме код такої функції, замінивши у даному випадку тип *Swap* на *int*, *float* та *char*. Таким чином можна зменшити розмір та складність програми, запропонувавши компілятору реалізацію узагальнених функцій.

*Приклад 2*. Функція приймає масив невизначеного типу та два цілих,  обмінюючи зміст елементів масиву з індексами x та y

template <class T> void change (T t[ ], int x, int y)  
{  
   T tmp=t[x];  
   t[x]=t[y];  
   t[y]=tmp;  
}

Коректними будуть такі виклики функцій за цим шаблоном:

double ar[7]={2,3,4,5,6,7,8};  
change(ar,2,3); // або change <double>(ar,2,3);  
char car[5]={'a','b','c','d','e'};  
change(car,1,4); // або change <char>(car,1,4);

Таким чином, є можливість легко створювати нові функції, підставляючи в існуючі шаблони конкретні аргументи.

#### 1.1.2 Параметризовані класи.

Визначаючи параметризований клас, ми створюємо його каркас (шаблон), що описує усі алгоритми, які використовуються класом. Фактичний тип даних, над яким проводитимуться маніпуляції, буде вказаний в якості параметру при конкретизації об’єктів цього класу. Компілятор автоматично згенерує відповідний об’єкт на основі вказаного типу. Загальна форма декларування параметризованого класу буде такою:

template <class Type> class class\_name  
{  
    // протокольна частина класу   
}

Визначення тіла класу аналогічно звичайному визначенню плюс використанню списку аргументів шаблону. Тип*Type* являє собою ім’я типу шаблону, яке в кожному випадку реалізації буде замінюватися конкретним типом даних. Типи можуть бути як стандартні, так і визначені користувачем, для їх опису завжди використовується ключове слово *class*. При необхідності можливо визначити більше одного параметризованого типу даних, використовуючи їх список через кому. У межах визначення класу шаблонне ім’я можна використовувати у будь-якому місці. Для створення конкретної реалізації використовується наступна форма:

class\_name <type> ob;

У цьому випадку *type* представляє собою ім’я конкретного типу даних, над якими фактично оперуватиме клас, та замінює змінну *Type*. До речі, елементи-функції, над якими оперуватиме клас, автоматично стають параметризованими, тобто їх необов’язково декларувати за допомогою ключового слова *template*(див. розділ 2.9.1. "Параметризовані функції"). Якщо метод описується за межами шаблону, його заголовок повинен мати наступні елементи:

template <список\_аргументів\_шаблону> тип\_функц

ім’я\_класу <аргументи\_шаблону>:: імя\_функц (список\_пар\_функц) {…}

При більш широкому розгляді шаблон класу - узагальнене визначення сімейства класів, яке може використовувати не лише довільні типи, а й константи. Синтаксис його узагальненого опису наступний:

template <список\_аргументів\_шаблону> class ім’я\_класу   
{  
   // визначення класу  
}

Аргумент (список\_аргументів\_шаблону) може складатися :

з ключового слова *class* , за яким слідує ідентифікатор, який визначає параметризований тип (*типовий параметр*);

з конкретного імені типу, за яким слідує ідентифікатор (*нетиповий параметр*, константа);

Відповідно до кожного з типів параметрів існує два [правила](https://learn.ztu.edu.ua/mod/page/view.php?id=1548) використання шаблонів класу. Для створення представника шаблонного класу потрібно вказати ім’я шаблону зі списком аргументів, що заключений у кутові дужки в якості специфікатору типу. Список аргументів модифікується так:

при використанні аргументів виду нетиповий параметр, тобто «*ім’я типу ідентифікатор*» заміна відбувається константним виразом.

при використанні аргументів виду “типовий параметр”, тобто “*class ідентифікатор*”, список аргументів модифікується з іменем типу;

Створивши представника шаблонного класу, надалі можливо працювати з ним так само, як з представником звичайного класу.

*Приклад 1*. Створимо параметризовану чергу із застосуванням в якості аргументу шаблону типового параметру. Головна функція демонструє використання цілих черг та черг з плаваючою комою на основі створеного шаблону класу.

#include "iostream.h"  
#include "stdlib.h"  
template <class TypeQ> class queue   
{  
   TypeQ \*q;  
   int sloc,rloc;  
   int length;  
  public:  
   queue(int size);   
   ~queue()  
   {   
     delete [] q;  
   }  
   void qstore(TypeQ i); // розміщення елемента в кінець черги  
   TypeQ qretrieve(); // вилучення першого елемента з черги  
};  
template <class TypeQ> queue<TypeQ>::queue(int size)   
{  
   size++;  
   q=new TypeQ[size];  
   if (!q)   
   {  
      cout<<"Неможливо створити чергу!\n";  
      exit(1);   
   }  
   length=size;  
   sloc=rloc=0;   
}  
  
template <class TypeQ> void queue<TypeQ>::qstore(TypeQ i)   
{  
   if (sloc+1==length)  
   {  
      cout<<"Черга переповнена!\n";  
      return;   
   }  
   sloc++;  
   q[sloc]=i;   
}  
template <class TypeQ> TypeQ queue<TypeQ>::qretrieve()   
{  
   if (rloc==sloc)  
   {  
      cout<<"Черга порожня!\n";  
      return 0;  
   }  
   rloc++;  
   return q[rloc]; }  
   int main(){  
   queue <int> a(5), b(5); //створення двох черг типу int  
   a.qstore(100);  
   b.qstore(200);  
   a.qstore(300);  
   b.qstore(400);  
   cout<< a.qretrieve()<<" ";   
   cout<< a.qretrieve()<<" ";   
   cout<< b.qretrieve()<<" ";   
   cout<< b.qretrieve()<<endl;   
   queue <float> f(5), e(5); //створення двох черг типу float  
   f.qstore(2.12);  
   e.qstore(2.99);  
   f.qstore(-30.00);  
   e.qstore(1.986);  
   cout<< f.qretrieve()<<" ";   
   cout<< f.qretrieve()<<" ";   
   cout<< e.qretrieve()<<" ";   
   cout<< e.qretrieve()<<endl;   
   return 0;  
}

Кожна з черг міститься у динамічному масиві, що адресується покажчиком *q.* Розмір черги передається як параметр конструктору класу *queue,*що зберігається у члені клас*у length.*. Відповідно змінні *rloc* та *slo*c використовуються для індексації черги; перша вказує індекс елементу, що буде вилучений, друга містить адресу, за якою буде збережено наступний елемент. Тип *TypeQ* можна розглядати як деякий формальний параметр черги, на місце якого при компіляції буде підставлено конкретний тип даних.

*Приклад 2*. Створимо визначення класу із застосуванням в якості аргументу шаблону типового та нетипового параметру. У якості прикладу розглянемо клас, що містить блок пам’яті визначеного типу та довжини.

// параметр шаблону - тип та константа   
template <class Type, int size>  class memo {  
  private:  
    Type \*p;   
  public:  
    memo()    {   
       p=new Type[size];   
    }  
   ~memo()    {  
       delete [ ] p;   
   }  
   operator Type\*();   
};   
template <class Type, int size>  
 memo <Type, size> :: operator Type \*() {   
   return p;   
}

Створення представника такого класу можливо таким способом:

memo<float, 50> floatblock;

Константа тут виступає нетиповим параметром. Взагалі у такій якості можуть передаватися змінні цілого та типу перелічення, а також покажчики або посилання на об'єкт або функцію.

Найпоширеніше застосування шаблони класів знаходять при створенні контейнерних класів. Перевага тут у тому, що як тільки логіка, необхідна для підтримки контейнеру, визначена, він може бути застосований до будь-яких типів даних без необхідності будь-якої перебудови. Завдяки цьому одного разу написаний та відлагоджений контейнерний клас можна використовувати повторно. Добре відома бібліотека шаблонів класів фірми *Borland*, що включає контейнерні класи, які можуть включатися до програм для керування колекціями даних різних типів.

Декілька слів щодо перевантаження шаблонів класу та функцій. В той час, як є можливим перевантаження імен шаблонів функцій, неможливим є цей процес для імен шаблонів класів.

Наприклад, неможливо визначити одночасно *Tarray<class T> та Tarray<class T, int size)*, в той час як без проблем перевантажуються шаблони функцій. Адже ніщо не завадить описати декілька шаблонів функцій з одним і тим же ім’ям, якщо лише вони мають відмінне число або різний тип параметрів.

## Лабораторна робота №8. Робота з формами. Обробка подій.

**Мета.** Одержати практичні навички розробки об’єктно-орієнтованої програми, з графічним інтерфейсом що керується подіями.

**Короткі теоретичні відомості.**

* Об’єктно-орієнтована програма як програма, що керується подіями.

При використанні ООП всі об'єкти є в деякому змісті відособленими один від одного і виникають визначені труднощі в передачі інформації від об'єкта до об'єкта. В ООП для передачі інформації між об'єктами використовується механізм обробки подій.

Події найкраще уявити собі як пакети інформації, якими обмінюються об'єкти і які створюються об’єктно-орієнтованим середовищем у відповідь на ті чи інші дії користувача. Натискання на клавішу чи маніпуляція мишею породжує подію, що передається по ланцюжку об'єктів, поки не знайдеться об'єкт, що знає як обробляти цю подію. Для того, що б подія могла передаватися від об'єкта до об'єкта, всі об'єкти програми повинні бути об'єднані в групу. Звідси випливає, що прикладна програма повинна бути об'єктом-групою, у яку повинні бути включені всі об'єкти, які використовуються в програмі.

Таким чином об’єктно-орієнтована програма – це програма, що керується подіями. Події самі по собі не роблять ніяких дій у програмі, але у відповідь на подію можуть створюватися нові об'єкти, модифікуватися чи знищуватися існуючі, що і приводить до зміни стану програми. Іншими словами всі дії по обробці даних реалізуються об'єктами, а події лише керують їхньою роботою.

Принцип незалежності обробки від процесу створення об'єктів приводить до появи двох паралельних процесів у рамках однієї програми: процесу створення об'єктів і процесу обробки даних.

Це означає, що дії по створенню, наприклад, інтерактивних елементів програми (вікон, меню й ін.) можна здійснювати, не піклуючись про дії користувача, що будуть зв'язані з ними.

І, навпаки, ми можемо розробляти частини програми, які відповідають за обробку дій користувача, не зв'язуючи ці частини зі створенням потрібних інтерактивних елементів.

* **Подія.**

Подія з погляду мови С++ – це об'єкт, окремі поля якого характеризують ті чи інші властивості переданої інформації, наприклад:

*struct TEvent*

*{int what*

*union{*

*MouseEventType mouse;*

*KeyDownEvent keyDown;*

*MessageEvent message;*

*};*

Об'єкт *TEvent* складається з двох частин. Перша (*what*) задає тип події, що визначає джерело даної події. Друга задає інформацію, передану з подією. Для різних типів подій зміст інформації різний . Поле *what* може приймати наступні значення:

***evNothing***– це порожня подія, яка означає, що нічого робити не треба. Полю *what* привласнюється значення *evNothing*, коли подія оброблена яким -небудь об'єктом.

***evMouse*** – подія від “миші”.

Подія від “миші” може мати, наприклад, таку структуру:

*struct MouseEventType*

*{int buttons;*

*int doubleClick;*

*TPoint where;*

*};*

де *buttons* вказує натиснуту клавішу;

*doubleClick* вказує чи був подвійний щиглик;

*where* вказує координати “миші”.

***evKeyDown*** -подія від клавіатури.

Подія від клавіатури може мати, наприклад, таку структуру:

*struct KeyDownEvent*

*{union{int keyCode;*

*union{char charCode;*

*char scanCode;*

*};*

*};*

*};*

***evMessage***- подія-повідомлення від об'єкта.

Для події від об'єкта (*evMessage*) задаються два параметри :

*command* – код команди, яку необхідно виконати з появою даної події;

*infoPtr* – передається з подією (повідомлення) інформація.

*struct MessageEvent*

*{int command;*

*void infoPtr;*

*};*

* **Методи обробки подій.**

Наступні методи необхідні для організації обробки подій(назви довільні).

**GetEvent** – формування події;

**Execute** – реалізує головний цикл обробки подій. Він постійно одержує подію шляхом виклику GetEvent і обробляє їх за допомогою HandleEvent. Цей цикл завершується, коли надійде подія “кінець”.

**HandleEvent** – оброблювач подій. Обробляє кожну подію потрібним для нього чином. Якщо об'єкт повинен обробляти визначену подію (повідомлення), то його метод HandleEvent повинен розпізнавати цю подію і реагувати на неї належним чином. Подія може розпізнаватися, наприклад, по коду команди (поле command).

**ClearEvent** – очищає подію, коли вона оброблена, щоб вона не оброблялася далі.

* **Оброблювач подій (метод HandleEvent)**

Одержавши подію (структуру типу TEvent) оброблювач подій для класу TDerivedClass обробляє його за наступною схемою:

void TDerivedClass::HandleEvent(TEvent& event)

{ //Виклик оброблювача подій базового класу

TBaseClass::handleEvent( event );

if( event.what == evCommand ) // Якщо оброблювач подій базового класу

// подію не обробив

{

switch( event.message.command )

{

case cmCommand1:

// Обробка команди cmCommand1

// Очищення події

СlearEvent( event );

break;

case cmCommand2:

// Обробка команди cmCommand2

СlearEvent( event );

break;

…

case cmCommandN:

// Обробка команди cmCommand

СlearEvent( event );

break;

default: // подія не оброблена

break;

}

};

}

Оброблювач подій групи спочатку обробляє команди групи, а потім, якщо подія не оброблена, передає його своїм елементам, викликаючи їх оброблювачі подій.

void TGroup::HandleEvent(TEvent& event)

{ if( event.what == evCommand )

{switch( event.message.command )

// обробка подій об'єкта-групи

default: // подія не оброблена групою

//одержати доступ до першого елемента групи

while((event.what != evNothing)!!( /\* переглянуті не всі елементи \*/)

{

//викликати HandleEvent поточного елемента

//перейти до наступного елементу групи

}

break;

}

}

* **Метод ClearEvent-очищення події.**

ClearEvent очищає подію, привласнюючи полю event.What значення evNothing.

* **Головний цикл обробки подій (метод Execute)**

Головний цикл обробки подій реалізується в методі Execute головної групи – об'єкта “прикладна програма” за наступною схемою:

int TMyApp::Execute()

{do{endState=0;

GetEvent(event); //одержати подію

HandleEvent(event); //обробити подію

if(event.what!=evNothing) //подія залишилася не обробленою

EventError(event);

}

while(!Valid());

return endState;

}

Метод HandleEvent програми обробляє подію “кінець роботи”, викликаючи метод EndExec. EndExec змінює значення private – змінної EndState. Значення цієї змінної перевіряє метод – функція Valid, що повертає значення true, якщо “кінець роботи”. Такий трохи складний спосіб завершення роботи програми зв'язаний з тим, що в активному стані можуть знаходитися кілька елементів групи. Тоді метод Valid групи, викликаючи методи Valid своїх піделементів, поверне true, якщо усі вони повернуть true. Це гарантує, що програма завершить свою роботу, коли завершать роботу всі її елементи.

Якщо подія залишилася не обробленим, то викликається метод EventError, що у найпростішому випадку може просто видати повідомлення.

* **Приклад обробки подій**.

Розглянемо найпростіший калькулятор, що сприймає команди в командному рядку. Тут приводиться спрощений варіант. Варіант, за схемою якого варто виконати лабораторну роботу, приведений у **Додатку.**

Формат команди:

знак параметр

Знаки +, –, \*, /, =, ?, q

Параметр – ціле число

Константи-команди:

сonst int evNothing = 0;

сonst int evMessage = 100;

сonst int cmSet = 1; //занести число

сonst int cmGet = 2; //подивитися значення

сonst int cmAdd = 3; //додати

і т.д.

сonst int cmQuit = 101; //вихід

Клас-подія

*struct TEvent*

*{int what*

*union{*

*int evNothing;*

*union{int command;*

*int a;}*

*}*

*}*

**Варіанти завдань до Лр 8 ООП.**

*З 2023року згідно завдань до Лр1-ЛР7 по С# в проєкт ЛР7 додаємо події, делегати через події, власні простори імен, псевдоніми просторів, класів, функції розширення закритих класів, часткові класи, часткові методи.*

Завдання при дистанційній формі навчання на 1 пару.

**Розробити проект «Конвектор величин» за технологією візуального та об’єктно – орієнтованого програмування у Visual Studio C#. Передбачити запис у файл даних та результатів та читання з файлу з обробкою виключних випадків з використанням обробників подій.**

**Задача. Таблиця мір**

Побудувати таблицю відповідностей між мірами. Початкове значення міри, крок зміни цього значення та кількість рядків у таблиці (10-15) задати самостійно у режимі діалогу. Оформити таблицю якнайкраще, використовуючи формати виведення.

1. 1 унція = 28.353495 г = 142 карати;
2. 1 драхм = 1.77185 г = 0.06249 унцій;
3. 1 карат = 0.2 г = 2.9412 гран;
4. 1 гран = 0.068 г = 0.038378 драхм;
5. 1 пайп = 54.18 пек = 477.33 л;
6. 1 галон (брит.) = 1.2 галон (США) = 4.546л;
7. 1 галон (США) = 0.0347 сак = 3.785 л;
8. 1 чарка = 0.0568 л = 0.00012 пайпа;
9. 1 квартет = 291 л = 5123.24 чарок;
10. 1 страйк = 72.73 л =1280.46 чарок;
11. 1 челдрон = 1.309 л = 0.149 пека;
12. 1 сак = 109 л =1.499 страйка;
13. 1 пек = 8.81 л = 0.07929 сака;
14. 1 корд малий = 3.624 куб. м = 128 куб. футів;
15. 1 стандарт = 4.672 куб. м = 0.165 рода;
16. 1 род = 28.3 куб. м = 1000 куб. футів;
17. 1 чейн будівельний = 30.48 м = 100 футів; І
18. 1 фінгер =11.4 см = 4.5 дюймів;
19. 1 нейл = 5.7 см = 2.25 дюймів;
20. 1 фут = 0.3048 м = 12 дюймів;
21. 1 ярд = 0.9144 м =3 фути;
22. 1 кабельт Брит. = 0.183 км = 680 футів;
23. 1 кабельт США = 219.5 м = 720 футів;
24. 1 дюйм= 2.54 см = 12 ліній;
25. 1 Морська миля =1.852 км =6076 футів.

Оформити звіт про виконану роботу.

## Лабораторна робота №9. Проектування класів n та компонент.

**Мета:** Набути навиків створення нащадків класів. Засвоєння принципу ( наслідування( спадковості) в ООП(C#).

**Постановка задачі:** Розробити програму, після запуску якої на формі з’являється анімоване зображення многокутника: вершини рухаються під кутом в різних напрямках і відбиваються від меж форми.

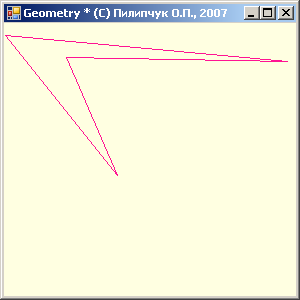
## Проект «Каток» (katok.exe)

### Базовий матеріал

1. Створення нового компоненту UserControl.
2. Генерування випадкових чисел.
3. Використання таймера.
4. Метод **DrawPolygon**.

### Детальний опис функціонування проекту

1. Зовнішній вигляд вікна програми під час виконання показаний на малюнку:

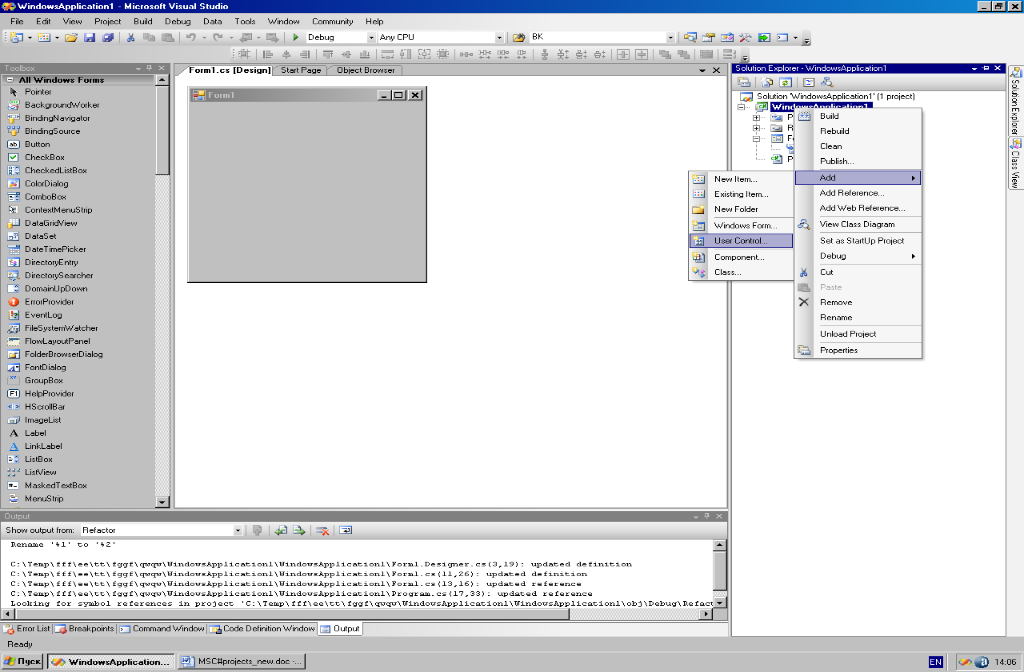


1. Кількість вершин обирається випадково при запуску програми в межах від 3 до 7.
2. При зміні розмірів вікна програми положення вершин многокутника повинне коректно змінюватись.

### План розробки

1. Створити новий проект. File/New/Project

2. В розділі Solution Explorer виділити об’єкт WindowsApplication1і викликати контекстне меню. В ньому вибрати Add/User Control та натисніть кнопку Add.



3, На вкладці Class view перейдіть WindowsApplication/WindowsApplication1.

Перейменуйте об’єкти Form1 і UserControl на MainForm та Katok відповідно.

1. В розділі Solution Explorer клацніть двічі на UserControl1.cs. Додаємо об’єкт Timer на вкладку UserControl1.cs [design]
2. На вкладці Class view клацніть двічі по Katok та перепишіть код

**using** System;

**using** System.ComponentModel;

**using** System.Drawing;

**using** System.Windows.Forms;

**namespace** Geometry

{

**public** partial class Katok : UserControl

{

**public** Point[] P;

**int**[] dx;

**int**[] dy;

Random R = **new Random**();

**int** n;

**public Katok**(**int** N)

{

**InitializeComponent**();

n=N;

P = **new** Point[N];

dx = **new int**[N];

dy = **new int**[N];

// Генеруємо ламану

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

{

P[i].X=R.**Next**(Width);

P[i].Y=R.**Next**(Height);

**do** {dx[i]=R.**Next**(6)-3;} **while** (dx[i]==0);

**do** {dy[i]=R.**Next**(6)-3;} **while** (dy[i]==0);

}

timer1.**Start**();

}

1. Двічі клацніть по компоненту **Timer** розташовному на вкладці UserControl1.cs [design] та перепишіть код

void **Timer1Tick**(object sender, EventArgs e)

{

//Обчислюємо нові координати і, за потреби,

//змінюємо напрями

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

{

P[i].X+=dx[i];

P[i].Y+=dy[i];

**if** ((P[i].X<3)||(P[i].X>Width-3))

dx[i]=-dx[i];

**if** ((P[i].Y<3)||(P[i].Y>Height-3))

dy[i]=-dy[i];

**if** (P[i].X>Width-3) P[i].X=Width-3;

**if** (P[i].Y>Height-3) P[i].Y=Height-3;

}

**Invalidate**();

}

1. Перейдіть на вкладку UserControl1.cs [design] та обираємо кнопку Events. Створюємо обробник подій Paint та перепишіть код

void **KatokPaint**(object sender, PaintEventArgs e)

{

e.Graphics.**DrawPolygon**(Pens.DeepPink, P);

}

1. На вкладці Class view клацніть двічі по MainForm та перепишіть код

**using** System.Drawing;

**using** System.Windows.Forms;

**namespace** Geometry

{

**public** partial class MainForm : Form

{

Katok K;

**public MainForm**()

{

**InitializeComponent**();

Random R=**new Random**();

K=**new Katok**(R.**Next**(3,8));

K.Parent=**this**;

K.Size=ClientSize;

K.Anchor = (AnchorStyles.Bottom |  
 AnchorStyles.Left |

AnchorStyles.Right |  
 AnchorStyles.Top);

}

}

}

### Завдання для вдосконалення проекту

1. Доповнити компонент Katok встановленням випадкових кольорів фону та ліній, причому кольори мають відрізнятися так, щоб лінії було добре видно.
2. Забезпечити відображення одночасно п’ятьох послідовних фаз руху. При цьому малюється 1-ше, 2-е, 3-є, 4-е, 5-те положення многокутника. Потім стирається 1-е і домальовується 6-те, стирається 2-е і домальовується 7-е і т.п.

### Завдання для самостійної розробки

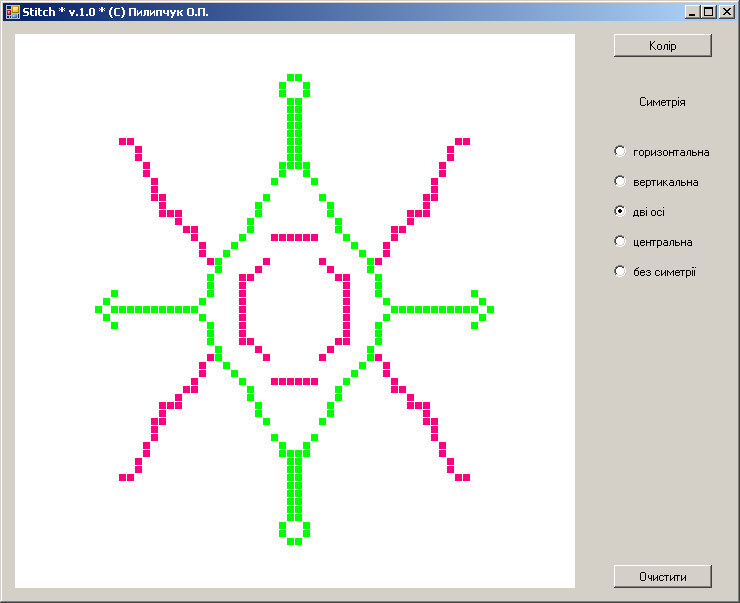
1. Створити проект, у якому було б одночасно декілька рухомих многокутників:
   1. в різних частинах екрану (декілька компонентів Katok);
   2. в одному прямокутнику (змінений компонент Katok).

## Проект «Редактор візерунків» (stitch.exe)

**Постановка задачі:** Розробити програму - редактор візерунків для вишивки хрестиком.

### Детальний опис функціонування проекту

1. Зовнішній вигляд вікна програми під час роботи показаний на малюнку:



1. Кнопкою «Колір» відкривається стандартне діалогове вікно для вибору кольору.
2. При клацанні в межах білого прямокутника на ньому з’являються квадратики, з яких складається візерунок.
3. Залежно від обраного виду симетрії (група радіокнопок), з’являються також квадратики, розміщені симетрично відносно певних осей (вертикальної, горизонтальної, обох) або центра малюнка.
4. Кнопкою «Очистити» візерунок стирається, а програма залишається готовою для подальшої роботи.

### План розробки

1. Розробити компонент Polotno – спадкоємець UserControl з такими властивостями:
   1. при клацанні його поверхні зберігається інформація про координати майбутнього квадратика і він з’являється на екрані. Також, за потреби, з’являються симетричні квадратики;
   2. метод Clear призводить до знищення інформації про наявний візерунок і очищення малюнка;
2. Встановити на форму компоненти у відповідності до малюнка.
3. Налаштувати дію радіокнопок.

### Приклад програмного коду

1. Основна програма.

/\*

\*

\*/

**using** System;

**using** System.Collections.Generic;

**using** System.Drawing;

**using** System.Windows.Forms;

**namespace** Stitch

{

**public** partial class MainForm : Form

{

**public MainForm**()

{

**InitializeComponent**();

}

void **Button1Click**(object sender, EventArgs e)

{

colorDialog1.**ShowDialog**();

polotno1.ActiveColor=colorDialog1.Color;

}

void **RBGorizCheckedChanged**(object sender, EventArgs e)

{

polotno1.symm=2;

}

void **RBWertCheckedChanged**(object sender, EventArgs e)

{

polotno1.symm=1;

}

void **RBDwiCheckedChanged**(object sender, EventArgs e)

{

polotno1.symm=3;

}

void **RBCentrCheckedChanged**(object sender, EventArgs e)

{

polotno1.symm=4;

}

void **RBNoSymCheckedChanged**(object sender, EventArgs e)

{

polotno1.symm=0;

}

void **Button2Click**(object sender, EventArgs e)

{

polotno1.**Clear**();

}

}

}

1. Модуль компоненту Polotno.

**using** System;

**using** System.ComponentModel;

**using** System.Drawing;

**using** System.Windows.Forms;

**namespace** Stitch

{

**public** partial class Polotno : UserControl

{

**public Polotno**()

{

**InitializeComponent**();

}

**public** Color ActiveColor=Color.Red;

**public** string protokol="";

**public int** symm=0; //Вид симетрії: //0 - без симетрії; 1-вертикальна вісь;

//2 - горизонтальна вісь; 3 - дві осі;

//4 - центральна

**public int** CWid,CHig; //Ширина і висота в

//квадратиках

**int** CrossWidth=8;

void **AddCross** (MouseEventArgs e, **int** x, **int** y)

{

//додаємо "клаць" до протоколу

//вилучивши, за потреби, вже наявного хрестика

//в цій точці

**int** X=x/CrossWidth\*CrossWidth;

**int** Y=y/CrossWidth\*CrossWidth;

**for** (**int** p=0; p<protokol.Length; p+=26)

**if** ((X==**int**.**Parse**(protokol.**Substring**(p,5))) && Y==**int**.**Parse**(protokol.**Substring**(p+5,5))))

{ protokol=protokol.**Remove**(p,26); break;}

protokol+=string.**Format**("{0,5}{1,5}{2,4}{3,4}{4,4}{5,4}",X,Y,ActiveColor.A.**ToString**(),ActiveColor.R.**ToString**(),ActiveColor.G.**ToString**(),ActiveColor.B.**ToString**());

}

void **PolotnoMouseClick**(object sender, MouseEventArgs e)

{

//ставимо хрестик в точці клацання

**AddCross**(e,e.X,e.Y);

**if** (symm>=3) //центральна

**AddCross**(e,CWid\*(CrossWidth)-e.X-1,   
 CHig\*(CrossWidth)-e.Y-1);

**if** ((symm==1)||(symm==3)) // вертикальна вісь

**AddCross**(e,CWid\*(CrossWidth)-e.X-1,e.Y);

**if** ((symm==2)||(symm==3)) // горизонтальна вісь

**AddCross**(e,e.X,CHig\*(CrossWidth)-e.Y-1);

**Invalidate**();

}

void **PolotnoPaint**(object sender, PaintEventArgs e)

{

SolidBrush B=**new SolidBrush**(ActiveColor);

**for** (**int** x=0; x<protokol.Length;x+=26)

{// малюємо прямокутнички за даними протоколу

B.Color=Color.**FromArgb**(**int**.**Parse**(protokol.**Substring**(x+10,4)),**int**.**Parse**(protokol.**Substring**(x+14,4)),**int**.**Parse**(protokol.**Substring**(x+18,4)),**int**.**Parse**(protokol.**Substring**(x+22,4)));

e.Graphics.**FillRectangle**(B,**int**.**Parse**(protokol.**Substring**(x,5)),**int**.**Parse**(protokol.**Substring**(x+5,5)),CrossWidth-1,CrossWidth-1);

}

}

**public** void **Clear**()

{

protokol="";

**Invalidate**();

}

void **PolotnoSizeChanged**(object sender, EventArgs e)

{

CWid=Width/CrossWidth;

CHig=Height/CrossWidth;

}

}

}

### Завдання для вдосконалення проекту

1. Забезпечити збереження візерунка на диск та завантаження його з диска для продовження роботи.
2. Забезпечити збереження візерунка на диск у графічному форматі.
3. Забезпечити друкування візерунка на папері.
4. Забезпечити побудову суцільних ліній перетягуванням мишею.
5. Забезпечити «заливання» суцільної області обраним кольором.

### Завдання для самостійної розробки

1. Розробити програму – графічний редактор (на зразок Paint) з можливістю побудови симетричних фігур.

## Проект «Хрестики-нулики» (xo.exe)

**Постановка задачі:** Розробити програму для гри з комп’ютером у хрестики-нулики.

### Детальний опис функціонування проекту

1. Зовнішній вигляд вікна програми показаний на малюнках:



На початку гри



У процесі гри

1. Після запуску людина розпочинає гру клацанням однієї з ігрових кнопок.
2. Людина грає хрестиками, а комп’ютер – нуликами.
3. Комп’ютер коректно відповідає на ходи людини.
4. На клацання кнопки з позначкою комп’ютер не реагує.
5. У випадку виграшу одного з партнерів:
   * невикористані кнопки блокуються, тобто перестають реагувати на клацання;
   * у відповідному написі відображається поновлене значення лічильника перемог для гравця;
   * після клацання кнопки «Почати» ігрове поле очищається і програма стає готова для нової гри.

### План розробки

1. Розробити клас GameButton – нащадок класу Button.
2. Розробити клас Pole – компонент користувача, який, зокрема, містить подію «Хтось виграв».
3. На формі розмістити компонент Pole та інші компоненти і забезпечити їх взаємодію.

### Приклад програмного коду необхідних класів.

1. Клас GameButton – ігрова кнопка.

Розпочнемо з розробки компоненту GameButton (ігрова кнопка), 9 екземплярів якого складатимуть поле для гри. Клас GameButton буде спадкоємцем класу Button (кнопка), тому що саме компонент Button має потрібний для нас базовий набір властивостей, методів та подій:

class GameButton : Button

{

Поле Text буде використовуватись для виведення на кнопку позначки «Х» або «0». Додамо три поля логічного типу, значення яких описуватимуть стан кнопки: IsClear, IsCross, IsZero. Їх призначення зрозуміле з таблиці:

|  |  |  |  |
| --- | --- | --- | --- |
| Поле | Про що сигналізує? | | Примітка |
|  | True | False |  |
| IsClear | Кнопка «чиста» | На кнопці є позначка | Потрібне для прийняття рішення, чи має кнопка реагувати на клацання |
| IsCross | На кнопці є хрестик | На кнопці не хрестик | Потрібне для аналізу стану ігрового поля після ходу гравця та для вибору найкращого ходу комп’ютером |
| IsZero | На кнопці є нулик | На кнопці не нулик | «« |

**public bool** IsClear, //прапорець «Кнопка чиста»

IsCross, //прапорець «Кнопка з хрестиком»

IsZero; //прапорець «Кнопка з нуликом»

Новостворена кнопка не повинна містити позначки (Text="";). Також при створенні кнопки треба налаштувати деякі її властивості (розмір шрифту, значення згаданих логічних полів). Тому створимо для неї конструктор такого вигляду:

**public GameButton**() //конструктор

{

Text="";

Font = **new Font**(Font.Name, 30, Font.Style);

IsClear=**true**;

IsCross=IsZero=**false**;

}

Метод SetCross компоненту GameButton призначений для того, щоб у процесі гри поставити на кнопці позначкук-хрестик. При цьому відповідно мають змінитися поля IsClear, IsCross, IsZero:

**public** void **SetCross**()//Метод «Поставити хрестик»

{

Text="X";

IsCross=**true**; IsClear=IsZero=**false**;

}

Аналогічно напишемо й метод SetNull для встановлення нулика:

**public** void **SetNull**()//Метод «Поставити нулик»

{

Text="O";

IsClear=IsCross=**false**; IsZero=**true**;

}

Для початку нової гри потрібно буде стирати позначки з кнопок, тому додамо метод SetClear:

**public** void **SetClear**()//Метод «Зробити чистою»

{

Text="";

IsClear=**true**; IsCross=IsZero=**false**;

}

}

1. Клас Pole – ігрове поле з 9 кнопок.

public partial class Pole : UserControl

{

GameButton[,] P = new GameButton [3,3]; //Ігрове поле з кнопок

public event FinishEventHandler OnFinish; //Подія "Хтось виграв"

bool active=true; //Прапорець "Чи реагувати на кнопки?"

int Moves=0; //Лічильник ходів на випадок "нічиєї"

public Pole()

{

InitializeComponent();

this.ClientSizeChanged +=new EventHandler(PoleClientSizeChanged);

//Створення кнопок

for (int r=0; r<3; r++)

for (int c=0; c<3; c++)

{

P[r,c]=new GameButton ();

P[r,c].Parent = this;

P[r,c].Click += new EventHandler(GameButtonClick);

}

}

void Control() //перевірка, чи раптом хтось не виграв. Якщо виграв

//- генерується подія OnFinish і передається символ-переможець

{

if ( P[0,0].IsCross && P[0,1].IsCross && P[0,2].IsCross || P[1,0].IsCross && P[1,1].IsCross && P[1,2].IsCross || P[2,0].IsCross && P[2,1].IsCross && P[2,2].IsCross || P[0,0].IsCross && P[1,0].IsCross && P[2,0].IsCross || P[0,1].IsCross && P[1,1].IsCross && P[2,1].IsCross || P[0,2].IsCross && P[1,2].IsCross && P[2,2].IsCross || P[0,0].IsCross && P[1,1].IsCross && P[2,2].IsCross || P[0,2].IsCross && P[1,1].IsCross && P[2,0].IsCross)   
{active=false;   
 if (OnFinish != null) OnFinish ('X');

}

if ( P[0,0].IsZero && P[0,1].IsZero && P[0,2].IsZero || P[1,0].IsZero && P[1,1].IsZero && P[1,2].IsZero || P[2,0].IsZero && P[2,1].IsZero && P[2,2].IsZero || P[0,0].IsZero && P[1,0].IsZero && P[2,0].IsZero || P[0,1].IsZero && P[1,1].IsZero && P[2,1].IsZero || P[0,2].IsZero && P[1,2].IsZero && P[2,2].IsZero || P[0,0].IsZero && P[1,1].IsZero && P[2,2].IsZero || P[0,2].IsZero && P[1,1].IsZero && P[2,0].IsZero)  
{active=false;   
 if (OnFinish != null) OnFinish ('O');}

}

void PoleClientSizeChanged(object sender, EventArgs e)

{

//Підгонка розмірів та розміщення кнопок на полі

for (int r=0; r<3; r++)

for (int c=0; c<3; c++)

{

P[r,c].Width = ClientRectangle.Width/3;

P[r,c].Height = ClientRectangle.Height/3;

P[r,c].Left = c\*P[r,c].Width;

P[r,c].Top = r\*P[r,c].Height;

}

}

public void Clear() //Підготовка до нової партії

{

for (int r=0; r<3; r++)

for (int c=0; c<3; c++)

P[r,c].SetClear();

active=true; Moves=0;

}

void GameButtonClick(object sender, EventArgs e)

{

if ((sender as GameButton).IsClear)

if (active) //Якщо партія не закінчена

{

(sender as GameButton).SetCross(); //ставимо на   
 //кнопці хрестик

Control(); //Перевірити, чи не переміг   
 //той, хто клацав

if (++Moves>8) active=false;//Якщо останній   
 //хід - блокуємо поле

if (active) //Якщо потрібна відповідь   
 //комп'ютера

{ int kh=0, nr=0, ns=0;

//Пошук виграшного ходу для нуликів

for (int r=0; r<3; r++)

for (int s=0; s<3; s++)

if (P[r,s].IsClear)

{ //перевірка у стовпці

if (kh!=2)

{ kh=0; for (int rr=0; rr<3; rr++)

if (P[rr,s].IsZero)

{ kh++;

if (kh==2) { ns=s; nr=r; }

}

}

//перевірка у рядку

if (kh!=2)

{ kh=0;

for (int ss=0; ss<3; ss++)

if (P[r,ss].IsZero)

{ kh++;

if (kh==2) { ns=s; nr=r; }

}

}

//перевірка у діагоналі 1

if ((kh!=2) && (r==s))

{ kh=0;

for (int ss=0; ss<3; ss++)

if (P[ss,ss].IsZero)

{ kh++;

if (kh==2) { ns=s; nr=r; }

}

}

//перевірка у діагоналі 2

if ((kh!=2) && (r==2-s))

{ kh=0;

for (int ss=0; ss<3; ss++)

if (P[ss,2-ss].IsZero)

{ kh++;

if (kh==2) { ns=s; nr=r; }

}

}

}

if (kh!=2) //Якщо нулики не виграють цим ходом, то

//пошук виграшного ходу хрестиків, щоб не програти

for (int r=0; r<3; r++)

for (int s=0; s<3; s++)

if (P[r,s].IsClear)

{ //перевірка у стовпці

if (kh!=2)

{ kh=0;

for (int rr=0; rr<3; rr++)

if (P[rr,s].IsCross)

{ kh++;

if (kh==2) { ns=s; nr=r; }

}

}

//перевірка у рядку

if (kh!=2)

{ kh=0;

for (int ss=0; ss<3; ss++)

if (P[r,ss].IsCross)

{ kh++;

if (kh==2) { ns=s; nr=r; }

}

}

//перевірка у діагоналі 1

if ((kh!=2) && (r==s))

{ kh=0;

for (int ss=0; ss<3; ss++)

if (P[ss,ss].IsCross)

{ kh++;

if (kh==2) { ns=s; nr=r;}

}

}

//перевірка у діагоналі 2

if ((kh!=2) && (r==2-s))

{ kh=0;

for (int ss=0; ss<3; ss++)

if (P[ss,2-ss].IsCross)

{ kh++;

if (kh==2) { ns=s; nr=r; }

}

}

}

if (kh!=2) //якщо немає виграшного ходу і

//загрози програшу – випадковий хід

{

Random RN = new Random();

do {nr = RN.Next(3); ns = RN.Next(3);}

while (!P[nr,ns].IsClear);

}

P[nr,ns].SetNull(); Moves++;

Control(); return;

}

}

}//Кінець GameButtonClick

}//Кінець Pole

1. Програмний код головної форми програми (на доповнення до згенерованого автоматично)

int CrossCount=0, ZeroCount=0; //Лічильники перемог

public MainForm()

{

InitializeComponent();

//додаємо обробник для події OnFinish

pole1.OnFinish += new FinishEventHandler (Finish);

}

void Finish (char Winner)

{

switch (Winner)

{case 'X': CrossCount++; labelMan.Text = Convert.ToString (CrossCount); break;

case 'O': ZeroCount++; labelComp.Text = Convert.ToString (ZeroCount); break;

}

}

void Button1Click(object sender, EventArgs e)

{

pole1.Clear();

}

}

### Завдання для вдосконалення проекту

1. Ім'я та результат того, хто має більше перемог, виділяти червоним жирним.
2. Зробити хрестики зеленими, а нулики червоними.
3. Після початку партії кнопка "Почати" стає недоступною до кінця партії.
4. Коли перемагає людина, напис "Почати" на кнопці змінюється на "Вітаю!" (червоним кольором), а після клацання - знову стає "Почати" (чорним кольором).
5. Підраховувати і виводити кількість нічиїх.
6. При клацанні кнопки, що вже позначена генерувати якийсь звук.
7. Забезпечити введення імені людини-гравця та відображення його замість слова «ЛЮДИНА».
8. Забезпечити три рівні "майстерності" гри комп'ютера:
   * початківець (ходить випадково),
   * кандидат (не пропускає своїх виграшних ходів)
   * експерт (не пропускає своїх виграшних ходів і, по можливості, заважає виграти людині).
9. Підсвічувати лінію знаків, яка призвела до перемоги.
10. Замість символів «Х» та «О» виводити на кнопці графічні зображення хрестика та нулика.
11. Забезпечити людині можливість вибору, чим грати (хрестиками чи нуликами).
12. Забезпечити можливість вибору, хто ходитиме першим (у всіх партіях): комп’ютер чи людина.
13. Забезпечити можливість вибору, хто ходитиме першим: у першій партії і при нічиїй – комп’ютер чи людина, а в наступних – переможець попередньої партії, чи переможений.

### Завдання для самостійної розробки

Розробити програму, що імітує головоломку «15».

|  |  |
| --- | --- |
| Шпаргалка по **С та С++** | |
| // *ля-ля-ля* | Коментар. Дія коментарю – від // та до кінця цієї одної стрічки |
| /\* *ку-ку-ку*  *гав-мяв*  \*/ | Коментар. Позначає певну часту коду як коментар. /\* позначає початок, \*/ кінець блоку. |
| **int** | Цілий тип даних. Має 32 розряди (від -2 147 483 648 до 2 147 483 647).  **int** k;  k=3; |
| **char** | Символьний тип. Для представлення символів та чисел. Розмірність 8 біт.  -128 <= X <= 127  **char** ch;  ch=’M’; |
| **bool** | Логічний тип. Приймає лише 2 значення: **true** (вірно) та **false** (невірно)  **bool** begin\_bool;  begin\_bool=**true**; |
| **float** | Тип даних з плаваючою точкою. Розмірність 32 розряди. 7 значущих цифр.  1.18e-38 < X < 3.40e+38  **float** s, d, k;  s=4.345; d=3.45e8; k=6.5e-4; |
| **double** | Тип даних з плаваючою точкою. Розмірність 64 розряди. 15 значущих цифр.  2.23e-308 < X < 1.79e+308 |
| **long double** | Тип даних з плаваючою точкою. Розмірність 80 розрядів. 18 значущих цифр.  3.37e-4932 < X < 1.18e+4932 |
| **int** number[25]; | Масив даних типу **int**, що складається з 25 значень. Нумерація починається з нуля.  **int** sum, number[25];  number[0]=23; number[5]=12; number[15]=9; //*присвоєння значень*  sum=number[0]+ number[5]+ number[15];  **int** k[4]={3,4,1,7}; //*ініціалізація масиву, лише при його визначенні* |
| **double** D[5][5]; | Двохвимірний масив. Аналогічно представляються масиви й вищої розмірності.  **double** r, K[3][3][3];  K[2][2][2]=3.45; K[1][1][1]=0.34; K[0][0][0]=1.2e-2; //*присвоєння значень*  r = K[2][2][2] – K[1][1][1] – K[0][0][0];  **double** D[3][3]={{1.2, 5.45, 2.4},{0.1, 2.1, 3.4},{5.7, 6.5, 7.3}}; //*ініціалізація* |
| **char** S[5]; | Масив символів. Стрічкою він стає при наявності нульового символу \0 , що служить для позначення кінця стрічки  **char** dog[5] = {‘s’,’f’’,’d’,’r’,’q’}; //*простий масив символів*  **char** pig[5] = {‘r’,’t’,’z’,’k’,’\0’}; //*стрічка*  **char** fish[20] = “My name is Andrew”; //*ініціалізація стрічки з \0 в кінці*  **char** car[] = “Who is this?”; //*автоматичне визначення розміру масиву* |
| **struct** | Структура – визначений користувачем тип даних. Опис структури здійснюється як правило на початку лістингу, хоча може описуватися і всередині функції, але тоді вона буде недоступною для інших функцій. В С++ зручніше використовувати класи. Доступ до елементів структури здійснюється через крапку (.).  **struct** book //*шаблон структури*  {  **char** name[20];  **double** price;  };  void main()  {  book Kipling={“The jungle book”, 25.5}; //*ініціалізація структури*  Kipling.price=21.25; //*присвоєння елементу іншого значення*  } |
| **union** | Об’єднання можуть містити один елемент різних типів. Використовується для економії пам’яті. Розмір виділяється для найбільш ємнісного типу.  **union** one  {  **int** k;  **double** dd;  };  ………………………  one men;  men.k=15; //*містить значення типу* ***int***  men.dd=2.34; //*містить значення типу* ***double****, значення* ***int*** *втрачено* |
| **enum** | Перелік створює символічні константи для цілих чисел, починаючи з 0 (по замовчуванню). Єдино допустимими значеннями можуть бути лише значення (назви) зі списку. Часто має застосування для оператора **switch**.  **enum** color {red, orange, yellow};  ……………………..  color a, b;  a=red; b=yellow;  a=2; //*невірно!!!!!* |
| &number | Оператор & визначає адрес змінної. |
| **int**  \* | Оголошення покажчика на значення типу **int**.  **int** M, number=5; //*оголошення та ініціалізація змінних*  **int** \*p\_number; //*оголошення покажчика*  p\_number = &number; //*присвоєння адресу змінної покажчику*  M = \*p\_number; //*отримання значення змінної через покажчик*  \*p\_number = \*p\_number + 2; //*додавання до змінної числа 2* |
| **new** | Оператор виділення пам’яті. Виділяє пам'ять під змінні будь-яких типів, в тому числі структури, класи і т.п. Це є необхідним особливо при створенні динамічних масивів, розмір яких ми не знаємо наперед.  **double** \*dob; //*оголошення покажчика*  dob = **new** **double**; //*виділення області пам’яті для величини типу double*  \*dop = 34.237; //*занесення значення в область пам’яті* |
| **delete** | Оператор очищення виділеної пам’яті.  **int** \*ps = **new** **int**; //*виділення пам’яті*  **delete** ps; //*очищення пам’яті* |
| **new** **int** [10] | Виділення пам’яті для масиву з 10 значень. Доступ до елементів масиву, виділеного динамічно, здійснюється так само, як і до звичайного.  **int** \*k = **new** **int** [10]; //*виділення блоку пам’яті для масиву з 10 елементів*  k[0]=2; k[4]=4;  **delete** [ ] k; //*очищення пам’яті, відведеної для динамічного масиву* |
| **double** \*\* | Покажчик на двовимірний масив. Спершу виділяється пам'ять для масиву покажчиків, а потім для елементів цього масиву покажчиків покроково виділяються блоки пам’яті.  **double** \*\*dd; //*оголошення покажчика на покажчики типу* ***double***  dd = **new** **double** \*[3]; //*виділення пам’яті для масиву покажчиків*  **for** (**int** j=0; j<3; j++)  dd[j] = **new** **double** [3]; //*поелементне виділення пам’яті одинарних масивів*  dd[1][2] = 3.45; dd[0][2] = 0.54; //*присвоєння значень елементам масиву*  Очищення пам’яті здійснюється у зворотному порядку  **for** (**int** i=0; i<3; i++)  **delete** [ ] dd[i]; //*очищення пам’яті для елементів масиву покажчиків*  **delete** [ ] dd; //*очищення пам’яті з під масиву покажчиків* |
| -> | Оператор приналежності. Використовується для доступу до елементів структур та класів, які оголошені через покажчик та є виділена для них пам’ять.  **struct** book { **сhar** name[20]; **double** price; }; //шаблон структури  void main()  {  book \*Kipling; //*оголошення покажчика на структуру*  Kipling = **new** book; //*виділення пам’яті*  Kipling->price = 31.25; //*присвоєння елементу значення*  } |
| ++  – – | Оператори інкремента та декремента. ++ збільшує значення на 1, -- зменшує. Розрізняють префіксну та суфіксну форму: ++x та x++. При ++x зміна значення здійснюється до обробки операнда, при x++ після обробки.  **int** x = 5, z = 5, y;  y = ++x; //*змінити значення x, а потім присвоїти його y. (x = 6, y = 6)*  y = z++; //*z присвоїти y, а потім змінити z (z=6, y=5)* |
| += –=  \*= /= | Комбіновані оператори присвоєння. Суть полягає в накопиченні значення для змінної.  i = i+ b; k = k \* c;  i += b; k \*= c; //*результат аналогічний як і у рядку вище* |
| <, <=, ==, >, >=, != | Оператори порівняння. Використовують для перевірки умов. Кожен вираз зводиться до значення **true** або **false**, в залежності від результату порівняння.  < менше, <= менше або рівно, == рівно, > більше, >= більше або рівно, != не рівно |
| ||  &&  ! | Логічні операції. Використовують для об’єднання та зміни існуючих операцій порівняння. || – або, && – і, ! – не  if (x<-3 && x>3) //*результат* ***true*** *якщо виконуються обидні умови*  if (x<-3 || x>3) //*результат* ***true*** *якщо виконується одна з умов*  if ( !(x>3) ) //*міняє результат логіки на протилежний,* ***false*** *= !* ***true*** |
| **for** (i=0; i<m; i++) | Цикл. Здійснює кількість повторів операції, допоки умова має значення **true**.  **for** (ініціалізація змінної; умова продовження циклу; обновлення змінної циклу)  **int** i, k;  **for** (i=0; i<5; i++)  k=k+i; //*в циклі повторюється одна операція*,  **for** (**int** j=50; j >10; j=j-5) //*або декілька, які об’єднані у блок за допомогою { }*  {  k=k\*j;  k=k+j;  }  **int** h, g;  **for** (h = 1, g = 25; g<5; h++, g--) // *h = 1, g = 25 – один вираз розділений комою*  k=k\*g+h; |
| **while** () | Простий цикл. Циклить допоки умова має значення **true**  **int** k=1;  **while** (k<5) //*коли k стане рівним 5, цикл завершиться*  k++; |
| **do while** () | Цикл, який виконується хоча б один раз. Циклить допоки умова має значення **true**.  **int** z, k=1;  **do**  {  k++;  z=z\*k;  }  **while** (k<5) //*коли k стане рівним 5, цикл завершиться* |
| **if** () **else** | Логічний оператор. Якщо умова виконується, то тоді виконується певна дія, в противному випадку виконується інша дія. Переважно використовують у спрощеній формі **if** (), не зазначаючи протилежної дії.  **int** z, x, k=1;  **if** (k==1) z=k;  k=2;  **if** (k==1) z=k;  **else** x=k;  При вкадених операторах **if** **if else** рекомендується використовувати { }, щоб уникнути помилки належності останньої операції. |
| ? : | Умовний оператор. Альтернатива **if else**  (умова) ? (вираз1) : (вираз2)  c = a>b ? a : b; //якщо a>b , тоді c=a, в протилежному випадку c=b |
| **switch** () | Оператор переключення.  **int** choice, k;  …………………….  **switch** (choice)  {  **case** 1 : k=1; **break**;  **case** 2 : k=5; **break**;  **case** 3 : k=9; **break**;  **default** : k=1; **break**;  }  В дужках () має бути значення цілочислового виразу. Якщо choice рівне 2, то програма переходить до стрічки із міткою **case** 2:, якщо значення не відповідає ні одній мітці, тоді виконується стрічка **default:**  Найчастіше мітками є просто константи типу **int**, **char** або елементи з переліку **enum**.  Варто зазначити, що даний оператор **switch** не здійснює єдиний вибір між наявними варіантами операцій, а лише здійснює перехід до позначеної міткою стрічки. Тобто, якщо значення дорівнює 2, то він переходить на другу стрічку та послідовно потім виконує усі наступні. Тому тут використовують оператор **break** для пропуску решта стрічок та виходу із оператора **switch.** |
| **break** | Оператор **break** використовується в операторі **switch** та любих циклах. Він завершує їхню дію та переходить до наступного оператора, що стоїть за ними. |
| **continue** | Оператор **continue** використовується в циклах. Він пропускає решту тіла цикла та починає нову ітерацію циклу.  **for** (**int** i=0; i<24; i++)  {  **if** (i==15) **continue**; //пропускає решта циклу цієї ітерації  **if** (i==20) **break**; //вихід з циклу  k=k+23;  } |
| **return** | Оператор завершення тіла функції та передачі назовні певного значення. |
| Функція () | Функція складається з переліку вхідних змінних різних типів та однієї змінної, що повертається туди, де був здійснений виклик даної функції  **int** get (**int** a, **int** b)  {  **int** c;  c=a+b\*3;  **return** c; //*повернення назовні значення*  }  **void** main (**void**)  {  int k, f=5, d=2;  k = get(f, d); //*виклик функції та присвоєння змінній k результату*  } |
| Функції та масиви | Одномірні масиви. У функції необхідно передавати покажчик на масив та кількість елементів масиву.  **int** sum (**int** arr[], **int** n)  {  **int** S=0;  **for** (**int** i=0 ; i<n ; i++)  S+=arr[i];  **return** S;  }  **void** main (**void**)  {  **int** d, f[5]= {2, 4, 6, 5, 3};  d=sum(f, 5);  }  У заголовку функції sum() замість **int** arr[] можна використовувати запис **int** \*arr. По змісту вони ідентичні. При передачі покажчика на масив у фунцію, ми одразу працюємо з оригінальним масивом, а не з копією.  **int** sum (**int** arr[], **int** n)  {  **for** (**int** i=0 ; i<n ; i++)  arr[i] = i\*2; //*заповнення масиву значеннями*  **return** 1;  }  Двовимірні масиви.  **int** data [3][4] = {{1,2,3,4,}, {2,3,4,5}, {3,4,5,6}};  s = sum(data, 3);  ……………………………………….  **int** sum (**int** ar2[ ] [4], **int** n)  {  **return** ar2[2][3];  } |
| <<  >> | Організація потокового вводу-виводу у файлах.  #include<fstream.h>  …………………………………………………..  **int** x1,x2,y1,y2;  fstream file; //*оголошення файлового потоку*  file.open("kaka.txt",ios\_base::out | ios\_base::trunc); //*відкриття файлу для запису*  file <<x1<<" "<<x2<<endl; //*вивід у файл значень x1,x2. endl – кін.стрічки*  file.close(); //*закриття файлу*  file.open("kaka.txt",ios\_base:: in); //*відкриття файлу для читання*  while(!file.eof()) //*перевірка на кінець файлу*  file>>y1>>y2; //*вивід значень у змінні y1 та y2*  file.close(); //*закриття файлу* |

Список рекомендованої літератури

**а) Основна**

1. Страуструп Б. Язык программирования С++.Третье издание, М.:Бином,1999.
2. Подбельский В.В. Язык С++.- М.: Финансы и статистика,1996.
3. Скляров В.А. Язык С++ и ООП.- Минск: Вышэйшая школа,1997.
4. Фейсон Т. Объектно - ориентированное программирование на С++ 4.5.-Киев:Диалектика,1996.
5. Шилдт Г. Теория и практика С++.-СПб.:BHV,1996.

**а) Додаткова**

1. Буч Г. Объектно - ориентированный анализ и проектирование с примерами приложений на С++. Второе издание. - М.:Бином,1998.
2. Бадд Т. Объектно - ориентированное программирование в действии. - СПб:Питер,1997.
3. Луис Д. С и С++.Справочник.- М: Бином,1997.
4. Киммел П. Borland C++5. -СПб.: BHV,1997.
5. Крейг Арнуш. BorlandC++:Освой самостоятельно-М.:Бином, 1997.
6. Пол Айра. Объектно - ориентированное программирование на С++ .Второе издание-М.: Бином,1999.
7. Шилдт Г. Самоучитель С++. Второе издание.- СПб.:BHV,1998.
8. Элджер Дж. С++: библиотека программиста - СПб: Питер,1999.

Електронні ресурси:

1. [www.intuit.ru](http://www.intuit.ru) ( курси по С++ та ООП)

2. [www.Cppstudio.com](http://www.Cppstudio.com)