Міністерство освіти і науки України

Національний університет „Львівська політехніка”

Кафедра “Електронних обчислювальних машин”



**Звіт з лабораторної роботи №7**

**Програмування, частина 2**

**(Об’єктно - орієнтоване програмування)**

на тему:

“Множинне спадкування. Поліморфізм.”

Варіант - 25

**Виконала:**

ст. гр. КІ-15

Романішина А. Ю.

**Перевірив:**

асист. каф. ЕОМ

Козак Н. Б.

Львів - 2020

**Мета:** познайомитися із множинним спадкуванням класів та поліморфізмів.

**Теоретичні відомості:**

*Множинне спадкуування :*

Якщо спадкування здійснюється від декількох батьківських класів одночасно, тоді воно називається множинним спадкуванням. Визначальним для похідного класу породженого множинним спадкуванням є те, що він явно чи неявно повинен успадковувати характеристики декількох базових класів.

Основні принципи одинарного спадкування, зокрема спадкування членів, модифікаторів доступу до членів базових класів, розширення та обмеження характеристик, без жодних доповнень можуть бути перенесені на множинне спадкування.

Неявним множинним спадкуванням можуть бути випадки змішаного спадкування. Результатом цих спадкуваннь 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 спадкування.

*Конструктори при множинному спадкуванні :*

Розглянемо алгоритм роботи конструкторів при множинному спадкуванні. При створенні об'єкта класу, який множинно породжений, 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 гілці дерева від найпершого класу, розпочне виконуватись гілка від другого батьківського класу і т.д. Після того, як відпрацюють конструктори усіх батьківських класів, виконається тіло конструктора похідного класу.

*Порядок передавання аргументів конструкторам :*

в оголошенні конструктора похідного класу може бути довільним, оскільки виклики і виконання конструкторів визначаються порядком спадкування в оголошенні класу.

Як і у випадку одинарного спадкування при визначенні конструктора можна передавати аргументи лише конструкторам виключно базових класів. Конструкторам класів, які не є безпосередньо в ієрархії, аргументи передаються лише через визначення конструкторів проміжних класів. Наприклад, наступне визначення конструктора є неправильним, якщо O() є батьківським класом тільки класу А або B.

*Деструктори при множинному спадкуванні :*

Порядок виклику деструкторів є таким як у конструкторів, а виконання - зворотнім. Найпершим почне виконуватись деструктор похідного класу, а далі - деструктори гілки породженої останньою в оголошенні батьківським класом. У порядку зворотному до декларації батьківських класів відпрацюють деструктори класів усіх гілок від них породжених. Лише по закінченню роботи і закритті батьківських деструкторів закриється деструктор похідного класу.

**Індивідуальне завдання:**

Спроектувати і реалізувати ієрархію класів, що описують предметну область згідно варіанту, яка реалізується класом 1. Клас 1 в свою чергу утворюється шляхом множинного спадкування класів 2 і 3 кожен з яких в свою чергу успадковує клас 4.

*Додаткові вимоги:*

1.Базовий клас містить мінімум один віртуальний метод, один невіртуальний метод і одну динамічно створювану властивість.  
2. Забезпечити механізми коректної роботи конструкторів і деструкторів.   
3. Перевантажити оператор присвоєня з метою його коректної роботи.  
4. Кожен з класів має містити мінімум одину властивість і 4 методи.   
5.Написати main() функцію де створити об‘єкт класу 1 і продемонструвати поліморфізмом.



**Код програми:**

#include <iostream>

#include <iomanip>

#include <Windows.h>

#define w 10

**using** **namespace** std;

**class** CCloth

{

**protected**:

**int** length, size;

**char**\* brand;

**public**:

CCloth(**const** **char**\* brand, **int** length, **int** size)

{

setInfo(brand, length, size);

}

~CCloth()

{

**delete**[] brand;

}

**void** setInfo(**const** **char**\* brand, **int** length, **int** size)

{

**this**->brand = **new** **char**[strlen(brand) + 1];

strcpy\_s(**this**->brand, strlen(brand) + 1, brand);

**this**->length = length;

**this**->size = size;

}

**virtual** **void** PrintInfo() = 0;

};

**class** CPants : **public** CCloth

{

**protected**:

**char**\* season;

**public**:

CPants(**const** **char**\* brand, **int** length, **int** size, **const** **char**\* season) :

CCloth(brand, length, size)

{

setInfo(season);

}

**void** setInfo(**const** **char**\* season)

{

**this**->season = **new** **char**[strlen(season) + 1];

strcpy\_s(**this**->season, strlen(season) + 1, season);

}

**virtual** **void** PrintInfo()

{

cout << "Pants info:" << endl;

cout << setw(w) << left << "Бренд: " << CCloth::brand << endl;

cout << setw(w) << left << "Довжина: " << CCloth::length << " см" << endl;

cout << setw(w) << left << "Розмір: " << CCloth::size << endl;

cout << setw(w) << left << "Сезон: " << season << endl;

}

};

**class** CShorts : **public** CCloth

{

**protected**:

**char**\* style;

**public**:

CShorts(**const** **char**\* brand, **int** length, **int** size, **const** **char**\* style) : CCloth(brand, length, size)

{

setInfo(style);

}

**void** setInfo(**const** **char**\* style)

{

**this**->style = **new** **char**[strlen(style) + 1];

strcpy\_s(**this**->style, strlen(style) + 1, style);

}

**virtual** **void** PrintInfo()

{

cout << "Shorts info:" << endl;

cout << setw(w) << left << "Бренд: " << CCloth::brand << endl;

cout << setw(w) << left << "Довжина: " << CCloth::length << " см" << endl;

cout << setw(w) << left << "Розмір: " << CCloth::size << endl;

cout << setw(w) << left << "Стиль: " << style << endl;

}

};

**class** CPantsShorts : **public** CPants, **public** CShorts

{

**public**:

CPantsShorts(**const** **char**\* brand, **int** length, **int** size, **const** **char**\* season, **const** **char**\* style) : CPants(brand, length, size, season), CShorts(brand, length, size, style)

{

}

**virtual** **void** PrintInfo()

{

cout << "Pants-Shorts info:" << endl;

cout << setw(w) << left << "Бренд: " << CCloth::brand << endl;

cout << setw(w) << left << "Довжина: " << CCloth::length << " см" << endl;

cout << setw(w) << left << "Розмір: " << CCloth::size << endl;

cout << setw(w) << left << "Сезон: " << CPants::season << endl;

cout << setw(w) << left << "Стиль: " << CShorts::style << endl;

}

};

**int** main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

CPants Pans1("Jeans", 70, 32, "Spring");

Pans1.PrintInfo();

cout << endl << endl;

CShorts Shorts1("Shorts", 40, 35, "Wide");

Shorts1.PrintInfo();

cout << endl << endl;

CPantsShorts PantsShorts1("PantsShorts", 50, 31, "Summer", "Capri");

PantsShorts1.PrintInfo();

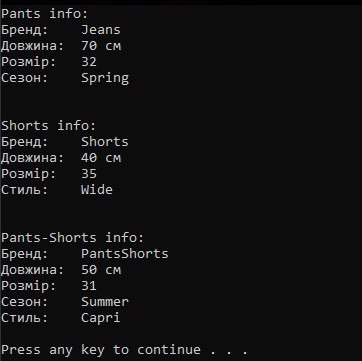
cout << endl;

system("pause");

**return** 0;

}

**Результат виконання функції:**

****

**Висновок**: на даній лабораторній роботі я познайомилась з множинним спадкуванням класів та поліморфізмів.