

Міністерство освіти і науки України  
Національний університет «Львівська політехніка»  
Кафедра ЕОМ



**Звіт**  
**до лабораторної роботи № 7**  
**з дисципліни: «Програмування, частина 2 (ООП)»**  
**на тему: «МНОЖИННЕ СПАДКУВАННЯ. ПОЛІМОРФІЗМ»**  
**Варіант № 14**

Підготував: Мишак М.А  
студент групи КІ-103  
Перевірив:  
Ст. викладач  
Каф. ЕОМ  
Гузинець Н.В

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

**Завдання:** Спроекувати 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 присвоєння з метою його коректної роботи.
4. Кожен з класів має містити мінімум одну властивість і 4 методи.
5. Написати main() функцію де створити об'єкт класу 1 і продемонструвати різницю між статичним і динамічним поліморфізмом.

14	Розкладний стіл	CFoldingTable	CCommode	CTable	CFurniture
----	-----------------	---------------	----------	--------	------------

### Теоретичний матеріал

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.

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.

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.

Поліморфізм – це здатність коду при постійному інтерфейсі змінювати свою поведінку в залежності від ситуації, яка виникає на момент виконання. Іншими словами один і той же метод може бути визначений для об'єктів різних класів,

що є між собою в ієрархії спадкування, при цьому метод якого класу викликати вирішується під час виконання програми.

Статичний поліморфізм реалізовується за допомогою так званого раннього зв'язування через механізм перевантаження функцій, методів та операторів і віртуальні класи. Він притаманний класичним структурним мовам програмування. При використанні статичного поліморфізму вибір функції чи методу, що буде викликатися здійснюється компілятором при компіляції програми (раннє зв'язування). Вибір функції чи методу в даному випадку залежить від типу вказівника чи посилання і не залежить від типу реального об'єкту на який вказує вказівник чи посилання. Тобто при звертанні до методу об'єкту похідного класу, використовуючи вказівник чи посилання на базовий клас, викликається буде метод базового класу.

Динамічний поліморфізм реалізовується за допомогою так званого пізнього зв'язування через механізм віртуальних функцій. Він притаманний об'єктно-орієнтованим мовам програмування і може бути застосований лише до методів класів. При використанні динамічного поліморфізму вибір методу, що буде викликатися здійснюється в процесі виконання програми (пізнє зв'язування). Вибір методу в даному випадку залежить від типу реального об'єкту на який вказує вказівник чи посилання. Тобто при звертанні до методу об'єкту похідного класу, використовуючи вказівник чи посилання на базовий клас, викликається буде метод похідного класу.

### **Лістинги (тексти) програм:**

#### **// main.cpp**

```
#include <iostream>
#include <windows.h>
#include "CFoldingTable.h"
using namespace std;

int main() {
    SetConsoleOutputCP(1251);
    SetConsoleCP(1251);

    CFoldingTable table("Дуб", 200, 100, 75, 3, true);

    table.display_info();

    table.increase_size(20, 10, 5);
    table.decrease_size(10, 5, 2);

    CFurniture* pFurniture = &table;
    pFurniture->display_info();

    return 0;
}
```

#### **// CFurniture.h**

```
#pragma once
#include <iostream>
#include <string>
using namespace std;

class CFurniture {
protected:
    string* material;
```

```

public:
    CFurniture(const string& material);
    CFurniture(const CFurniture& other);
    CFurniture& operator=(const CFurniture& other);
    virtual ~CFurniture();

    void set_material(const string& material);
    string get_material() const;

    virtual void display_info() const;
};

```

## // CFurniture.cpp

```
#include "CFurniture.h"
```

```

CFurniture::CFurniture(const string& mat) {
    material = new string(mat);
}

```

```

CFurniture::CFurniture(const CFurniture& other) {
    material = new string(*other.material);
}

```

```

CFurniture& CFurniture::operator=(const CFurniture& other) {
    if (this != &other) {
        delete material;
        material = new string(*other.material);
    }
    return *this;
}

```

```

CFurniture::~CFurniture() {
    delete material;
}

```

```

void CFurniture::set_material(const string& mat) {
    *material = mat;
}

```

```

string CFurniture::get_material() const {
    return *material;
}

```

```

void CFurniture::display_info() const {
    cout << "Matepian: " << *material << endl;
}

```

## // CTable.h

```
#pragma once
```

```
#include "CFurniture.h"
```

```

class CTable : public virtual CFurniture {
private:
    int length;
    int width;
    int height;

```

```

public:
    CTable(const string& material, int length, int width, int height);
    void set_dimensions(int length, int width, int height);
    void get_dimensions(int& length, int& width, int& height) const;

```

```

    void increase_size(int length, int width, int height);
    void display_info() const override;
};

```

## // CTable.cpp

```
#include "CTable.h"
```

```

CTable::CTable(const string& material, int length, int width, int height)
    : CFurniture(material), length(length), width(width), height(height) {}

void CTable::set_dimensions(int length, int width, int height) {
    this->length = length;
    this->width = width;
    this->height = height;
}

void CTable::get_dimensions(int& length, int& width, int& height) const {
    length = this->length;
    width = this->width;
    height = this->height;
}

void CTable::increase_size(int length, int width, int height) {
    this->length += length;
    this->width += width;
    this->height += height;
    cout << "Розміри столу збільшені до: " << this->length << "x" << this->width << "x"
<< this->height << " см." << endl;
}

void CTable::display_info() const {
    CFurniture::display_info();
    cout << "Розміри столу: " << length << "x" << width << "x" << height << " см." <<
endl;
}

```

**// CCommode .h**

```

#pragma once
#include "CFurniture.h"

class CCommode : public virtual CFurniture {
private:
    int numberOfDrawers;

public:
    CCommode(const string& material, int numberOfDrawers);
    void set_number_of_drawers(int drawers);
    int get_number_of_drawers() const;

    void display_info() const override;
};

```

**//CCommode .cpp**

```

#include "CCommode.h"

CCommode::CCommode(const string& material, int numberOfDrawers)
    : CFurniture(material), numberOfDrawers(numberOfDrawers) {}

void CCommode::set_number_of_drawers(int drawers) {
    numberOfDrawers = drawers;
}

int CCommode::get_number_of_drawers() const {
    return numberOfDrawers;
}

void CCommode::display_info() const {
    CFurniture::display_info();
    cout << "Кількість шухляд: " << numberOfDrawers << endl;
}

```

**// CFoldingTable.h**

```

#pragma once
#include "CTable.h"
#include "CCommode.h"

class CFoldingTable : public CTable, public CCommode {
private:
    bool isFoldable;

public:
    CFoldingTable(const string& material, int length, int width, int height, int
numberOfDrawers, bool foldable);
    void increase_size(int length, int width, int height);
    void decrease_size(int length, int width, int height);
    void display_info() const override;
};

// CFoldingTable.cpp
#include "CFoldingTable.h"

CFoldingTable::CFoldingTable(const string& material, int length, int width, int height,
int numberOfDrawers, bool foldable)
    : CFurniture(material), CTable(material, length, width, height), CCommode(material,
numberOfDrawers), isFoldable(foldable) {}

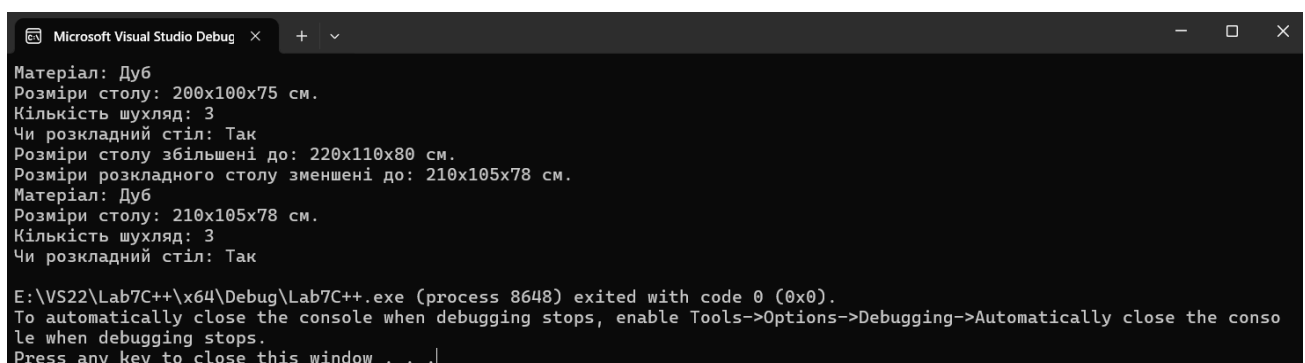
void CFoldingTable::increase_size(int length, int width, int height) {
    CTable::increase_size(length, width, height);
}

void CFoldingTable::decrease_size(int length, int width, int height) {
    int newLength, newWidth, newHeight;
    CTable::get_dimensions(newLength, newWidth, newHeight);
    newLength -= length;
    newWidth -= width;
    newHeight -= height;
    CTable::set_dimensions(newLength, newWidth, newHeight);
    cout << "Розміри розкладного столу зменшені до: " << newLength << "x" << newWidth
<< "x" << newHeight << " см." << endl;
}

void CFoldingTable::display_info() const {
    CTable::display_info();
    cout << "Кількість шухляд: " << CCommode::get_number_of_drawers() << endl;
    cout << "Чи розкладний стіл: " << (isFoldable ? "Так" : "Ні") << endl;
}

```

## Результати виконання програм:



```

Microsoft Visual Studio Debug
Матеріал: Дуб
Розміри столу: 200x100x75 см.
Кількість шухляд: 3
Чи розкладний стіл: Так
Розміри столу збільшені до: 220x110x80 см.
Розміри розкладного столу зменшені до: 210x105x78 см.
Матеріал: Дуб
Розміри столу: 210x105x78 см.
Кількість шухляд: 3
Чи розкладний стіл: Так

E:\VS22\Lab7C++\x64\Debug\Lab7C++.exe (process 8648) exited with code 0 (0x0).
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .

```

**Висновок:** На даній лабораторній роботі я познайомився з множинним наслідуванням у мові програмування C++. На основі цих знань створив програму з 4 класами згідно даного мені завдання.