Міністерство освіти і науки України Національний

технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Інститут прикладного системного аналізу

Кафедра математичних методів системного аналізу

**ЗВІТ**

при виконанні комп’ютерного практику №3

з дисципліни «Об’єктно-орієнтоване програмування»

Виконав:

Студент II курсу

Групи КА-03

Курилюк Р. М.

Варіант №13

Перевірив:

Шолохов О.В

Київ—2021

**Мета роботи:** Навчитись коректно користуватися перевантаженням функцій,

перевантаженням операцій та механізмом віртуальних функцій в С++.

**Основні теоретичні відомості:**

*Перевантаження функцій*. Перевантаження функцій означає, що для передачі повідомлень об’єктам різних класів можна скористатись функціями з однаковими іменами і що кожний об’єкт буде реагувати відповідним чином.

Така техніка використовується для базових операцій С++. Наприклад, існує одна назва і одна позначка для додавання: «+», але його можна використовувати для додавання цілих, дійсних, для інкременту вказівників. Ця ідея поширюється і на функції користувача. Фактично це різні функції і перевантаження імен має значення тільки для зручності запису.

Під час виклику функції f процес пошуку потрібної із множини перевантажених відбувається як пошук найбільшої відповідності типів формальних і фактичних параметрів.

*Перевантаження операторів*. Перевантаження операторів – можливість зміни семантичного навантаження стандартних операторів відповідно потребам нового класу або нових потреб існуючого класу.

В С++ допустимі перевантаження таких операторів:

*+, -, /, \* , %, ^ , &, !, ~, =, <, >, +=, - =, \* =, / =, % =, ^ =, & =, / =, <<, >>, ==, ! =, < =, > =, &&, //, ++, --, -> \*, [], (), new, new[], delete, delete[].*

Не можуть бути перевизначені оператори:

*::, ., .\*, :, sizeof()* та непередбачені в синтаксисі мови.

Назва операторної функції починається з ключового слова *operator*, за яким іде символ оператора, наприклад

*operator \**

Операторна функція оголошується і може бути викликана як звичайна функція. Використання її як оператора – скорочена форма явного виклику. Наприклад в описі класу «комплексне число» оголошено оператор += :

*class Complex*

*{ double re, im;*

*public:*

*complex (double r, double i): re(r), im(i){};*

*complex& operator += (complex&);*

*};*

Тоді операторна функція може бути використана двома способами:

*Complex a(1,2), b(4,-5);*

*a. operator += (b); // явний виклик*

*a+=b; // скорочена форма*

У перевантажених операторах зберігається пріоритет і порядок виконання у виразах, закріплені за даними операторами.

Унарний оператор можна визначити або у вигляді функції-члена без аргументів, або у вигляді функції-не-члена з одним аргументом.

Оператор індексації використовується для класів-контейнерів і надає можливість доступу до елемента колекції, яку містить такий клас, напряму, за індексом. Такий оператор має повертати посилання на тип елемента колекції, якщо його бажано використовувати і зліва і справа від оператора присвоєння.

Бінарний оператор можна визначити або у вигляді функції-члена з одним аргументом, або у вигляді функції-не-члена з двома аргументами.

Для ілюстрації сказаного, можна розглянути наступний приклад. Нехай в деякому класі Х існує ціле поле some. Операцію «+=« можна визначити як суму полів some двох об’єктів Х:

*class X*

*{...friend X& operator += (X&, const X&);...};*

*X& operator += (X&a, const X&b) { a.some+=b.some; return a;}*

*Або з використанням функції-члена:*

*class X{. . . X& operator +=(const X&);. . . };*

*X& X :: operator += (const X&b){ some += b.some; return\* this;}*

Оператор присвоєння відноситься до бінарних операторів і в С++ може бути перевизначений як звичайний бінарний оператор, але тільки як член класу.

*Дружні класи і дружні функції*. Оскільки деякі операторні функції можуть бути зовнішніми, виникає проблема доступу до прихованих частин класу. В цьому випадку використовують так звані дружні функції. Дружні функції повинні бути оголошені в середині опису класу, з яким вони дружні, за допомогою префікса friend. Наприклад:

*class A{int q; . . . friend void func (A, int);. . .};*

*void func (A a1, int i) {a1. q =i;};*

Якщо усі функції-члени одного класу (Х) є дружніми для другого класу (Y), говорять, що клас Х – дружній класу Y. Тоді цей факт оголошують так:

*class Y{. . . friend class X;. . . };*

*Перетворення типів.* У арифметичних виразах можуть бути присутні об’єкти різних типів, тому виникає проблеми перетворення типів.

Для перетворення визначеного раніше типу до типу, який визначає користувач, можна скористатись конструктором з одним параметром (конструктор перетворення). Наприклад:

*complex (double r) {re = r; im = 0;}*

Для перетворення у зворотному порядку, використовують спеціальну нестатичну функцію-член класу operator тип () {…};

Наприклад:

*complex :: operator double () {return (sqrt(re\*re+im\*im));};*

*Перевантаження операторів уведення – виведення*. Оскільки існує можливість створення типів даних користувача, логічно мати можливість форматного уведення і виведення значень такого типу, скориставшись формою запису відповідних стандартних операторів. Це бінарні операції «<<« – «помістити в потік» і «>>« – «взяти із потоку». Вони можуть бути визначені тільки як зовнішні функції.

Операторну функцію «<<« потрібно перевизначити так, щоб вона мала два аргументи: типу ostream & і користувацького типу і повертала ostream & (посилання на потік використовується, щоб не копіювати об’єкт потоку).

Для розглянутого раніше типу complex:

*class complex {. . . friend ostream & operator << (ostream & , complex x). .*

*.};*

*ostream & operator << (ostream & o, complex x)*

*{ return o <<x.re << ’+’ <<’ i’ <<x. im; };*

Аналогічно можна перевизначити оператор уведення з потоку.

У деяких випадках для перевантаження цих операторів зручно користуватись функціями-членами класів ostream та istream такими як put, get, getline, write, read та ignore.

*Пізнє зв’язування*. Якщо у класах, зв’язаних відносинами успадкування, існують функції з однаковими сигнатурами, це означає, що об’єктам цих класів можуть бути передані однакові повідомлення. Може виникнути ситуація, коли на етапі компіляції відомо, яке повідомлення потрібно передати, але тільки під час виконання стане відомо об’єкту якого класу. У цьому випадку використовують механізм пізнього зв’язування.

В С++ пізнє зв’язування реалізоване за допомогою функцій-членів, які називаються віртуальними функціями.

Віртуальну функцію оголошують у базовому класі за допомогою префікса virtual, а потім переозначають у похідних класах.

Віртуальні функції покладаються на додаткову структуру даних, яка підтримує зв’язок між різними версіями функцій. Це таблиця віртуальних функцій (virtual method table-vtbl) – таблиця вказівників на віртуальні функції, яка конструюється для кожного класу окремо. Всі екземпляри класу містять вказівник на цю таблицю.

Такий механізм дозволяє асоціювати і зв’язати повідомлення з методом під час виконання програми автоматично. Програміст тільки визначає дії, які повинен виконати об’єкт, отримавши повідомлення.

Щоб досягти пізнього зв’язування для об’єкта, необхідно скористатись вказівником або посиланням на нього. Для відкритих похідних класів вказівники і посилання на об’єкти цих класів сумісні з вказівниками і посиланнями на об’єкти базового класу. Обрана функція-член залежить від класу, на об’єкт якого вказана, а не від типу вказівника.

Наприклад:

*class Animal{ int age;*

*public: . . . virtual void Ask(); };*

*class Dog: public Animal{ char\* breed;*

*public: . . . void Ask(); };*

*class Cat: public Animal{ char\* color;*

*public: . . . void Ask(); };*

*void Animal:: Ask(){cout<<«I’m animal ! «;};*

*void Dog:: Ask(){cout<<«I’m “<<breed <<’!’;};*

*void Cat:: Ask(){cout<<«I’m “<<color<<« cat! «;};*

*void main()*

*{Animal \*a[3];*

*Animal a; Cat c(“white»); Dog d(“Doberman»);*

*a[0] = &a; a[1] = &c; a[2] = &d;*

*for (int i; i=0; i<3) a[i]->Ask(); };*

Результат:

*I’m animal! I’m white cat! I’m Doberman!*

Якщо в класі присутня хоча б одна віртуальна функція, його деструктор потрібно визначити як віртуальний.

**Завдання**

Відповідно варіанту до тексту другої лабораторної роботи потрібно внести наступні зміни:

* перевантажити указані арифметичні та логічні оператори, операто індексації та оператори форматного уведення-виведення для одного з класів відповідно варіанту;
* перевірку коректності за бажанням можна реалізувати без генерування виключних ситуацій;
* визначити оператор присвоювання для класів, для яких це доцільно;
* функцію, що виводить деяку скорочену інформацію про об’єкт,визначити як віртуальну.

Реалізувати тестовий приклад, у якому передбачити:

* демонстрацію роботи кожного з перевантажених операторів;
* демонстрацію роботи кожної з віртуальних функцій так, щоб був задіяний віртуальний механізм.

*Варіант 13.* Визначити:

* арифметичний оператор «+» для класу Date;
* логічні оператори «<»та «>» для класу «Страва» за вартістю;
* логічні оператори «==»та «!=» для перевірки збігу двох об’єктів класу «Повар»;
* оператор індексації для доступу до інформації про страву – для класу «Замовлення»;
* оператори форматного уведення-виведення – для класів «Страва» та «Повар».

**Діаграма класів**

Рис 1. Діаграма класів

**Лістинг програми**

**Source.cpp**

#include <locale.h>

#include <iostream>

#include <conio.h>

#include "Dish.h"

using namespace std;

int main() {

setlocale(LC\_ALL, "Ukrainian");

cout << "Введiть iм'я i прiзвище кухаря" << endl;

char\* name\_cook = new char;

char\* surname\_cook = new char;

cin >> name\_cook >> surname\_cook;

Cook cook(name\_cook, surname\_cook);

cook.show();

cout << endl << "Введiть iм'я страви, цiну, час приготування, категорiю, iм'я i прiзвище кухаря" << endl;

char\* name\_dish = new char;

int price = 0, time = 0, category = 0;

Category temp\_category;

cin >> name\_dish >> price >> time >> category >> name\_cook >> surname\_cook;

switch (category) {

case 0:

temp\_category = NONE;

break;

case 1:

temp\_category = COLD\_SNACKS;

break;

case 2:

temp\_category = FIRST\_COURSES;

break;

case 3:

temp\_category = SECOND\_COURSES;

break;

case 4:

temp\_category = DESSERTS;

break;

case 5:

temp\_category = DRINKS;

break;

default:

break;

}

Dish dish(name\_dish, price, time, temp\_category, name\_cook, surname\_cook);

dish.show();

\_getch();

return 0;

}

**Category.cpp**

enum Category {

NONE,

COLD\_SNACKS,

FIRST\_COURSES,

SECOND\_COURSES,

DESSERTS,

DRINKS

};

**Cook.h**

#pragma once

class Cook {

private:

char\* name, \* surname;

public:

Cook();

Cook(char\*, char\*);

Cook(Cook&);

~Cook();

char\* get\_name();

char\* get\_surname();

Cook& set\_name(char\*);

Cook& set\_surname(char\*);

void show();

};

**Cook.cpp**

#include <string>

#include <iostream>

#include "Cook.h"

using namespace std;

Cook::Cook()

{

name = new char[7];

strcpy\_s(name, sizeof "noname", "noname");

surname = new char[10];

strcpy\_s(surname, sizeof "nosurname", "nosurname");

cout << "Конструктор Cook за замовчування " << this << endl;

}

Cook::Cook(char\* \_name, char\* \_surname)

{

name = new char[strlen(\_name) + 1];

strcpy\_s(name, strlen(\_name) + 1, \_name);

surname = new char[strlen(\_surname) + 1];

strcpy\_s(surname, strlen(\_surname) + 1, \_surname);

cout << "Конструктор Cook з параметрами " << this << endl;

}

Cook::Cook(Cook& \_cook)

{

name = new char[strlen(\_cook.name) + 1];

strcpy\_s(name, strlen(\_cook.name) + 1, \_cook.name);

surname = new char[strlen(\_cook.surname) + 1];

strcpy\_s(surname, strlen(\_cook.surname) + 1, \_cook.surname);

cout << "Конструктор Cook копіювання " << this << endl;

}

Cook::~Cook()

{

cout << "Деструктор Cook " << this << endl;

}

char\* Cook::get\_name()

{

return name;

}

char\* Cook::get\_surname()

{

return surname;

}

Cook& Cook::set\_name(char\* \_name)

{

name = new char[strlen(\_name) + 1];

strcpy\_s(name, strlen(\_name) + 1, \_name);

return \*this;

}

Cook& Cook::set\_surname(char\* \_surname)

{

surname = new char[strlen(\_surname) + 1];

strcpy\_s(surname, strlen(\_surname) + 1, \_surname);

return \*this;

}

void Cook::show()

{

cout << name << ", " << surname << endl;

}

**Dish.h**

#pragma once

#include "Category.cpp"

#include "Cook.h"

class Dish :

public Cook

{

private:

char\* name\_dish;

int price, time;

Category category;

public:

Dish();

Dish(char\*, int, int, Category&);

Dish(char\*, int, int, Category&, Cook&);

Dish(char\*, int, int, Category&, char\*, char\*);

Dish(Dish&);

~Dish();

char\* get\_name();

int get\_price();

int get\_time();

Category& get\_category();

Dish& set\_name(char\*);

Dish& set\_price(int);

Dish& set\_time(int);

Dish& set\_category(Category&);

void show();

};

**Dish.cpp**

#include <string>

#include <iostream>

#include "Dish.h"

using namespace std;

Dish::Dish()

{

name\_dish = new char[7];

strcpy\_s(name\_dish, sizeof "noname", "noname");

price = 0;

time = 0;

category = NONE;

cout << "Конструктор Dish за замовчування " << this << endl;

}

Dish::Dish(char\* \_name\_dish, int \_price, int \_time, Category& \_category) :

Cook()

{

name\_dish = new char[strlen(\_name\_dish) + 1];

strcpy\_s(name\_dish, strlen(\_name\_dish) + 1, \_name\_dish);

price = \_price;

time = \_time;

category = \_category;

cout << "Конструктор Dish з параметрами " << this << endl;

}

Dish::Dish(char\* \_name\_dish, int \_price, int \_time, Category& \_category, Cook& \_cook) :

Cook(\_cook)

{

name\_dish = new char[strlen(\_name\_dish) + 1];

strcpy\_s(name\_dish, strlen(\_name\_dish) + 1, \_name\_dish);

price = \_price;

time = \_time;

category = \_category;

cout << "Конструктор Dish з параметрами " << this << endl;

}

Dish::Dish(char\* \_name\_dish, int \_price, int \_time, Category& \_category, char\* \_name, char\* \_surname) :

Cook(\_name, \_surname)

{

name\_dish = new char[strlen(\_name\_dish) + 1];

strcpy\_s(name\_dish, strlen(\_name\_dish) + 1, \_name\_dish);

price = \_price;

time = \_time;

category = \_category;

cout << "Конструктор Dish з параметрами " << this << endl;

}

Dish::Dish(Dish& \_dish):

Cook(\_dish.get\_name(), \_dish.get\_surname())

{

name\_dish = new char[strlen(\_dish.name\_dish) + 1];

strcpy\_s(name\_dish, strlen(\_dish.name\_dish) + 1, \_dish.name\_dish);

price = \_dish.price;

time = \_dish.time;

category = \_dish.category;

cout << "Конструктор Dish копіювання " << this << endl;

}

Dish::~Dish()

{

cout << "Деструктор Dish " << this << endl;

}

char\* Dish::get\_name()

{

return name\_dish;

}

int Dish::get\_price()

{

return price;

}

int Dish::get\_time()

{

return time;

}

Category& Dish::get\_category()

{

return category;

}

Dish& Dish::set\_name(char\* \_name\_dish)

{

delete[] name\_dish;

name\_dish = new char[strlen(\_name\_dish) + 1];

strcpy\_s(name\_dish, strlen(\_name\_dish) + 1, \_name\_dish);

return \*this;

}

Dish& Dish::set\_price(int price)

{

price = price;

return \*this;

}

Dish& Dish::set\_time(int time)

{

time = time;

return \*this;

}

Dish& Dish::set\_category(Category& category)

{

category = category;

return \*this;

}

void Dish::show()

{

cout << name\_dish << ", " << price << " грн, " << time << "хв, ";

switch (category)

{

case NONE:

cout << "NONE" << ", ";

break;

case COLD\_SNACKS:

cout << "холоднi закуски" << ", ";

break;

case FIRST\_COURSES:

cout << "перша страва" << ", ";

break;

case SECOND\_COURSES:

cout << "друга страва" << ", ";

break;

case DESSERTS:

cout << "десерт" << ", ";

break;

case DRINKS:

cout << "напiй" << ", ";

break;

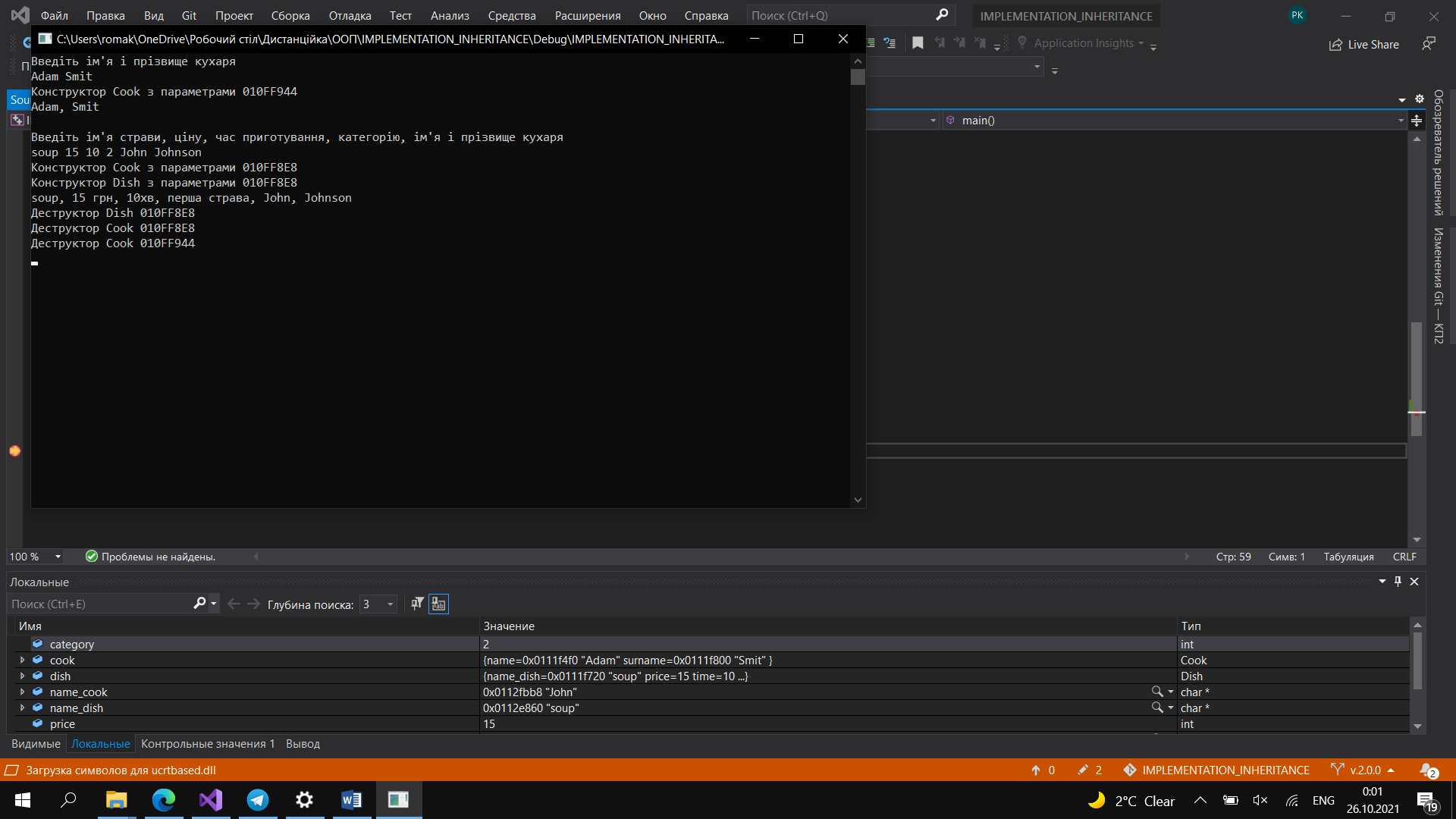
default:

break;

}

Cook::show();

}

**Результати роботи програми**

**Висновок**

Під час виконання комп'ютерного практикуму я навчився описувати ієрархії об’єктів та ієрархії класів мовою С++.

Було перероблено тип “Страва” як похідний від класу “Кухар”. Наданий тестовий приклад відповідно до загальних вимог та з виведенням інформації про об'єкти класу “Кухар” та “Страва”, а також повідомлення про конструювання або деструкцію.