Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»



Звіт

З лабораторної роботи №1

З дисципліни «Об’єктно-орієнтоване програмування »

На тему:

Перевантаження функій

Виконав: Куцевол В.С. KI-109

Прийняв: асис. каф. ЕОМ

Гузинець Н.В

Львів – 2024

**Тема роботи:** Перевантаження функій

**Мета.** : ознайомитися із перевантаженням функцій

**Теоретичні відомості**

У мові С++, на відміну від мови С дозволяється визначати декілька функцій з одним і тим же іменем за умови, що дані функції мають різну сигнатуру (різні типи та кількість аргументів функції). При цьому тип значення що повертається при перевантаженні до уваги не приймається.

При виклику функції з ім‘ям func спершу компілятор намагатиметься знайти функцію, формальні аргументи якої відповідають фактичним аргументам без усяких перетворень типів або з використанням тільки неминучих перетворень - наприклад, імені масиву до покажчика або значення змінної до константи або навпаки

Якщо відповідна функція не знайдена, то здійснюється пошук такої функції, щоб для відповідності формальних і фактичних аргументів досить було використати тільки такі стандартні перетворення, що не спричиняють перетворень цілих типів до типів з плаваючою крапкою і навпаки. При цьому підбирається функція, для якої число таких перетворень було б мінімальним.

Далі здійснюється пошук функції, для якої аргументи можна одержати за допомогою всіх перетворень, розглянутих до цього, а також перетворень типів, визначених самим програмістом. Якщо й у цьому випадку єдина потрібна функція не знайдена, то на останньому етапі компілятор пробує знайти відповідність з урахуванням списку невизначених аргументів. Так, виклик функції func (1, 2, 3) може бути співставлений лише з функцією, що оголошена як іnt func(float, ...).

Мова С++ пропонує безпечну заміну макросам – вбудовані (іnlіne) функції. На відміну від макросів, вбудовані функції не піддаються помилкам подвійного обчислення. Крім того типи аргументів перевіряються компілятором і при потребі виконуються всі необхідні перетворення. Таким чином, якщо у вас є маленька функція (кілька рядків коду), яку доводиться часто викликати, то її можна оголосити як іnlіne. Оголошена в такий спосіб функція буде проаналізована компілятором на можливість її реалізації у вигляді вбудованої функції. Якщо компілятор вважатиме доцільним реалізувати її як вбудовану, то дана функція не буде викликатися. Замість цього тіло функції підставлятиметься в те місце програми, де здійснюється виклик. При цьому підвищується ефективність програми ціною збільшення розміру коду програми. Якщо ж компілятор вважатиме, що дана функція завелика для її ефективної реалізації як вбудованої, то вона буде реалізована як звичайна функція. Таким чином кінцеве рішення у питанні робити функцію позначену як inline вбудованою чи ні належить компілятору.

**Хід виконання роботи**

**Варіант 12**

Завдання : 12.Написати реалізацію перевантажених функцій :

int func(int\* arr, int length); int func(double \* arr, int length);

Функція func повертає кількість парних елементів послідовності.

//file ArrayWork.h

#pragma once

#include <iostream>

#include <math.h>

using std::cout;

using std::cin;

using std::endl;

int func(int\* array, int lenght) {

int numberOfEvenElements = 0;

for (int i = 0; i < lenght; i++) {

if (array[i] % 2 == 0) {

numberOfEvenElements += 1;

}

}

return numberOfEvenElements;

}

bool IsApproximatelyInt(double number1, double epsilone = 0.001) {

return number1 - (int)number1 <= epsilone;

}

int func(double\* array, int lenght) {

int numberOfEvenElements = 0;

for (int i = 0; i < lenght; i++) {

if (IsApproximatelyInt(array[i]) && (int)array[i] % 2 == 0) {

numberOfEvenElements++;

}

}

return numberOfEvenElements;

}

int\* IntArray(int lenght) {

int\* evenNums = new int[lenght];

for (int i = 0 , element = 1 ; i < lenght; i++ , element++) {

cout << "Enter " << element << " element: " << endl;

cin >> evenNums[i];

}

return evenNums;

}

double\* DoubleArray(int lenght) {

double\* doubleNums = new double[lenght];

for (int i = 0 , element = 1; i < lenght; i++ , element++) {

cout << "Enter " << element << " element: " << endl;

cin >> doubleNums[i];

}

return doubleNums;

}

//file OOP.cpp

#include <iostream>

#include"ArrayWork.h"

using std::cout;

using std::cin;

using std::endl;

int main(void) {

int lenght, choice;

cout << "Enter how much number in array?\n";

cin >> lenght;

cout << "Numbers even or fractional?\n1.Even.\n2.Fractional.\n";

cin >> choice;

int\* intNums;

double\* doubleNums;

switch (choice){

case 1:

intNums = IntArray(lenght);

cout << func(intNums, lenght) << " even numbers in array";

delete intNums;

break;

case 2:

doubleNums = DoubleArray(lenght);

cout << func(doubleNums, lenght) << " even numbers in array";

delete doubleNums;

break;

default:

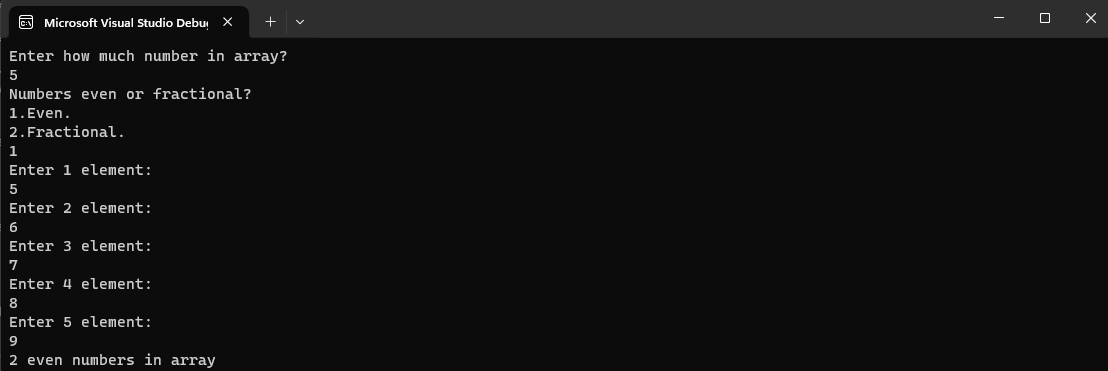
return 0;

break;

}

return 0;

}



**Висновок**

На цій лабораторній роботі я досліджував концепцію перевантаження функцій в об'єктно-орієнтованому програмуванні. Під час виконання роботи я ознайомився з основними принципами роботи перевантаження функцій, а саме можливістю створення кількох функцій з однаковим ім'ям, але різною кількістю аргументів або типами аргументів.

Я розглянув приклади використання перевантаження функцій для різних ситуацій, таких як обробка різних типів даних або виконання різних дій залежно від переданих аргументів. Зрозумів, що перевантаження функцій дозволяє покращити зручність та читабельність коду, оскільки дозволяє використовувати однакові назви функцій для схожих дій. Крім того, перевантаження функцій допомагає уникнути дублювання коду та спрощує його обслуговування.

В цілому, ця лабораторна робота дозволила мені глибше зрозуміти та практично застосувати концепцію перевантаження функцій, що є важливим елементом об'єктно-орієнтованого програмування.