Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»



Звіт

З лабораторної роботи №8

З дисципліни «Об’єктно-орієнтоване програмування »

На тему:

Шаблони

Виконав:

Куцевол В.С.

KI-109

Прийняв: асис. каф. ЕОМ

Гузинець Н.В

Львів – 2024

**Тема роботи:** Шаблони

**Мета.** : познайомитися із створенням шаблонів

**Теоретичні відомості**

Шаблони являють собою схематичний опис побудови класів та функцій. Використовуючи шаблони, з'являється можливість створювати узагальнені специфікації для класів та функцій, що найчастіше носять назву параметризованих класів (generic classes) та параметризованих функцій (generic functions). Шаблони не прив‘язані до конкретних типів даних і описують алгоритми, незалежно від типів даних. Дані алгоритми мають функціонувати однаково для різних типів даних. Такий опис дозволяє описати один раз функції, методи чи класи і на їх базі генерувати функції, методи і класи для кожного конкретного набору параметрів, що економить зусилля і час розробки програмного забезпечення. Після визначення загального шаблона, якщо для одного, кількох або всіх параметрів поведінка класу чи функції відрізнятиметься від описаної в загальному шаблоні, то створюється спеціалізація для конкретного набору параметрів. Спеціалізація може бути звичайною (неявною), явною або частковою.

Призначенням шаблонів є створення екземплярів (instantiating) шаблону, які вже є реальними функціями чи класами. При цьому відбувається прив'язування параметрів шаблону до даних визначеного типу. Цей процес називається конкретизацією. Cпроба компілятором створити екземпляр шаблону є генерацією програми. Тому зустрічаючи спробу створити екземпляр шаблону компілятор перемикається в режим його вивчення та запам'ятовування, а це - часові витирати.

Типи загального призначення, якими оперують шаблони, називаються шаблонними типами (template type), а їх сукупність параметрами шаблона (template parameters). Параметри шаблону як множина шаблонних типів може містити також преозначені і вбудовані типи С++.

Шаблонний тип Т є невизначеним узагальненим типом. По мірі використання шаблонів компілятор автоматично замінить тип Т іменем реального типу. Як правило, для імені шаблонного типу використовують ідентифікатори T чи Type. Проте це не обов'язково: ім'я можна декларувати будь-яким допустимим в С++ ідентифікатором. Шаблонний тип можна повноцінно використовувати в тілі шаблону, але це не є строгою вимогою.

Шаблон допускає використання параметрів, які ініціалізуються аргументами за замовчуванням, згідно з методологією оголошення і використання таких аргументів. Типи аргументів по замовчуванні можуть бути лише преозначеними або вбудованими. Використання шаблонних типів як аргументів по замовчуванню не допускається.

Під шаблон пам'ять не виділяється. Якщо екземпляр шаблону не створюється, то компілятор навіть не буде транслювати код шаблону. Це спричинює труднощі з використанням файлів заголовків, які містять лише оголошення шаблонів, а їх реалізація знаходиться у сpp-файлі. Для подолання цих недоліків треба підключати сpp-файл, а не файл заголовку, або код шаблону вносити у файл заголовку. Ранні версії С++ компіляторів не перевіряли синтаксис тіла незалежно від створення екземпляр шаблону. Сучасні компілятори відразу аналізують синтаксис коду тіла при першому знаходженні оголошення шаблону, а тому позбавлені цих недоліків.

Використання шаблонів може значно скоротити час створення програми. Це досягається тим, що з'являється можливість перенести незалежний від типу даних, один раз написаний і перевірений код спільний для множини різнорідних функцій в одну програмну конструкцію - шаблон.

Для виконання схожих операцій над різними типами даних часто використовуються перевантажені функції. Якщо ж для кожного типу даних повинні виконуватися ідентичні операції, то більш компактним і зручним рішенням є використання параметризованих (шаблонних) функцій. При цьому програміст повинен написати лише один опис шаблона функції. Базуючись на типах аргументів, використаних при виклику цієї функції, компілятор буде автоматично генерувати об'єктні коди функцій, що оброблятимуть кожен тип даних.

Параметризовані функції декларуються за допомогою ключового слова template. Це слово використовується для створення шаблону (каркасу), що в 138 загальних рисах описує призначення функції та надає опис операцій – сутність алгоритму, що може застосовуватися до даних різних типів.

За ключовим словом template слідує не порожній список параметрів шаблону, який складається з ідентифікаторів типу T, кожному з яких передує ключове слово class або typename (згідно новішого стандарту). Коли компілятор створюватиме конкретну версію функції, то автоматично замінить параметри конкретними типами даних. Цей процес носить назву інстанціювання шаблону.

Кожен формальний параметр з опису шаблона функції повинен з'явитися в списку параметрів функції принаймні один раз. Ім'я формального параметра може використовуватися в списку параметрів заголовка шаблона тільки один раз. Те ж ім'я формального параметра шаблона функції може використовуватися декількома шаблонами.

Шаблон функції може бути перевантажений, а саме можна визначити інші шаблони, що мають те ж ім'я функції, але різні набори параметрів. Також можна ввести не шаблонну функцію з тим же ім'ям та іншим набором параметрів функції.

Компілятор виконує процес узгодження, щоб визначити, який екземпляр функції відповідає конкретному викликові. Спочатку компілятор намагається знайти і використати функцію, що точно відповідає по імені та типам параметрів функції, що викликається. Якщо на цьому етапі компілятор зазнає невдачі, то він шукає шаблон функції, за допомогою якого він може згенерувати параметризовану функцію з точною відповідністю типів параметрів та імені функції; автоматичне перетворення типів не 139 забезпечується. І як останню спробу, компілятор послідовно виконує процес підбору перевантаженої функції.

Щоб створити із шаблона екземпляр конкретного класу, потрібно оголосити об'єкт, вказавши для його типу ім'я шаблона з набором конкретних аргументів (типів і констант). Кожен формальний тип у списку параметрів шаблона потрібно замінити на ім'я конкретного типу. Кожна формальна константа заміняється на константу зазначеного в шаблоні типу. Після того, як представник шаблонного класу створений, з ним можна поводитись як і з будьяким об'єктом, що належить до звичайного класу.

**Хід виконання роботи**

**Варіант 12**

Завдання : Контейнерний клас описує та забезпечує набір дій над даними параметризованого масиву, розмірність якого визначається під час роботи програми. Усі обчислення та перетворення повинні бути реалізовані у вигляді методів класу.

Дана прямокутна матриця. Визначити номер рядка, в якому знаходиться найдовша серія з різних елементів.

//file Matrix.h

#pragma once

#include <iostream>

using namespace std;

template <typename T >

class Matrix

{

private:

int columns;

int rows;

T\*\* matrix;

public:

Matrix(int columns, int rows) {

this -> columns = columns;

this -> rows = rows;

this -> matrix = new T\*[columns];

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

Matrix::matrix[i] = new T[rows];

}

}

void SetColomns(int colomns){

Matrix::columns = colomns;

}

int GetColomns() {

return Matrix::columns;

}

void SetRows(int rows) {

Matrix::rows = rows;

}

int GetRows() {

return Matrix::rows;

}

void SetMatrix(T\*\* matrix) {

Matrix::matrix = matrix;

}

T\*\* GetMatrix() {

return Matrix::matrix;

}

void FillMatrixWithElements() {

for (int i = 0; i < Matrix::columns; i++) {

for (int j = 0; j < Matrix::rows; j++) {

cout << "Enter element" << "[" << i + 1 << "]" << "[" << j + 1 << "]: ";

cin >> Matrix::matrix[i][j];

}

}

}

void PrintMatrix() {

cout << "Your matrix: " << endl;

for (int i = 0; i < Matrix::columns; i++) {

for (int j = 0; j < Matrix::rows; j++) {

cout << Matrix::matrix[i][j] << " ";

}

cout << endl;

}

}

void GetRowWithHighestNumberRow() {

int generalCounter = 0;

int counter = 1;

int rowWithHighest = 1;

for (int i = 0; i < Matrix::columns; i++) {

for (int j = 0; j < Matrix::rows; j++) {

counter = 1;

for (int k = j + 1; k < Matrix::rows; k++) {

if (Matrix::matrix[i][j] == Matrix::matrix[i][k]) {

counter++;

}

else

counter = 1;

if (counter > 1) {

rowWithHighest = i;

}

}

}

if (counter > generalCounter) {

generalCounter = counter;

}

}

cout << "Row with highest row of same numbers: " << rowWithHighest+1 << endl;

cout << "{ ";

for (int i = 0; i < Matrix::columns; i++) {

cout << Matrix::matrix[rowWithHighest][i] << " ";

}

cout << "}";

}

};

//File SwitchPParts.h

#pragma once

#include <iostream>

#include "Matrix.h"

void caseInt(int columns , int rows) {

Matrix<int>matrix(columns, rows);

matrix.FillMatrixWithElements();

matrix.PrintMatrix();

matrix.GetRowWithHighestNumberRow();

}

void caseFloat(int columns, int rows) {

Matrix <float> matrix(columns, rows);

matrix.FillMatrixWithElements();

matrix.PrintMatrix();

matrix.GetRowWithHighestNumberRow();

}

void caseDouble(int columns, int rows) {

Matrix <double> matrix(columns, rows);

matrix.FillMatrixWithElements();

matrix.PrintMatrix();

matrix.GetRowWithHighestNumberRow();

}

void caseChar(int columns, int rows) {

Matrix <char> matrix(columns, rows);

matrix.FillMatrixWithElements();

matrix.PrintMatrix();

matrix.GetRowWithHighestNumberRow();

}

//file main.cpp

#include <iostream>

#include "Matrix.h"

#include"SwitchParts.h"

int main() {

int userChoice;

int columns, rows;

cout << "Enter number of columns: ";

cin >> columns;

cout << "Enter number of rows: ";

cin >> rows;

cout << "\nChoose the type of data of your matrix:\n1.Int.\n2.Float\n3.Double\n4.Char\n\nChoice: ";

cin >> userChoice;

switch (userChoice)

{

case 1:

caseInt(columns, rows);

break;

case 2:

caseFloat(columns, rows);

break;

case 3:

caseDouble(columns, rows);

break;

case 4:

caseChar(columns, rows);

break;

default:

break;

}

return 0;

}



**Висновок**

У ході цієї лабораторної роботи я вивчив основні концепції шаблонів у мові програмування C++. Шаблони є потужним інструментом, який дозволяє писати загальні та перевикористовувані фрагменти коду. Я зрозумів, що шаблони дозволяють створювати функції та класи, які можуть працювати з будь-яким типом даних, що надає гнучкість та ефективність у програмуванні.

Під час виконання лабораторної роботи я розглянув різні типи шаблонів, такі як шаблони функцій та класів, а також вивчив, як використовувати шаблони для створення загальних алгоритмів та структур даних. Я також отримав досвід у використанні параметрів шаблонів та уникненні шаблонних антипатернів.

Завдяки цій роботі я навчився ефективно використовувати шаблони у своїх програмах, що дозволить мені писати більш чистий, зрозумілий та переносний код. Я також зрозумів важливість дотримання найкращих практик при використанні шаблонів для забезпечення читабельності та підтримки моїх програм у майбутньому.

Top of Form