Міністерство освіти і науки України

Національний університет „Львівська політехніка”

Кафедра “Електронних обчислювальних машин”



**Звіт з лабораторної роботи №8**

**Програмування, частина 2**

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

на тему:

“Шаблони”

Варіант - 25

**Виконала:**

ст. гр. КІ-15

Романішина А. Ю.

**Перевірив:**

асист. каф. ЕОМ

Козак Н. Б.

Львів - 2020

**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.

Основним застереженням при роботі з шаблонами є правильне використання операцій до змінних шаблонних типів. Тобто усі операції, які використовувались до змінної шаблонної типу, повинні мати місце для того типу, яким буде заміщений даний шаблонний тип.

Таким чином, за допомогою реалізації узагальнених функцій можна зменшити розмір та складність програми. Особливо корисними шаблони є саме в бібліотеках класів – тут вони вказують програмісту необхідні специфікації, приховуючи при цьому деталі справжньої реалізації.

*Параметризовані функції :*

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 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.

Синтаксис виклику шаблонної функції є таким самим як і виклик звичайної функції і має наступний вигляд:

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 забезпечується. І як останню спробу, компілятор послідовно виконує процес підбору перевантаженої функції.

**Індивідуальне 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.

*Варіант 25*

У довільному масиві обчислити кількість елементів масиву, що розташовані по спаданню і стоять на парних позиціях.

**Код програми:**

#include <iostream>

#include <iomanip>

#include <Windows.h>

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

**template** <**typename** type>

**class** Matrix

{

**private**:

**int** x, y;

type\*\* Arr;

**int** size\_x, size\_y;

**public**:

Matrix() {

}

Matrix(**int** size\_x, **int** size\_y)

{

x = size\_x;

y = size\_y;

Arr = **new** type \* [size\_x];

**for** (**int** i = 0; i < size\_x; i++)

{

Arr[i] = **new** type[size\_y];

**for** (**int** j = 0; j < size\_y; j++)

{

Arr[i][j] = 0;

}

}

}

**void** create(**int** size\_x, **int** size\_y)

{

Arr = **new** type \* [size\_x];

**for** (**int** i = 0; i < size\_x; i++)

{

Arr[i] = **new** type[size\_y];

**for** (**int** j = 0; j < size\_y; j++)

{

Arr[i][j] = 0;

}

}

}

~Matrix()

{

**for** (**int** i = 0; i < size\_x; i++)

{

**delete** Arr[i];

}

**delete**[] Arr;

}

**friend** ostream& **operator**<<(ostream& output, Matrix<type>& matr)

{

**for** (**int** i = 0; i < matr.size\_x; i++)

{

**for** (**int** j = 0; j < matr.size\_y; j++)

{

output << matr.Arr[i][j] << "\t";

}

output << endl << endl;

}

output << endl;

**return** output;

}

**friend** istream& **operator**>>(istream& is, Matrix<type>& matr)

{

matr.~Matrix();

cout << "Введіть кількість рядків: ";

is >> matr.size\_x;

cout << "Введіть кількість стовпців: ";

is >> matr.size\_y;

matr.Arr = **new** type \* [matr.size\_x];

cout << "Для рандомного заповнення введіть - 1\nДля ручного заповнення введіть - 2\n";

matr.create(matr.size\_x, matr.size\_y);

**int** l = 1;

cin >> l;

**switch** (l)

{

**case** 1:

**for** (**int** i = 0; i < matr.size\_x; i++)

{

**for** (**int** j = 0; j < matr.size\_y; j++)

{

matr.Arr[i][j] = rand() % 99;

}

}

**break**;

**case** 2:

**for** (**int** i = 0; i < matr.size\_x; i++)

{

**for** (**int** j = 0; j < matr.size\_y; j++)

{

cout << "a[" << i << "]" << "[" << j << "]" << "=";

is >> matr.Arr[i][j];

}

}

**break**;

**return** is;

}

}

**int** search()

{

**int** n = 0;

**for** (**int** i = 0; i < size\_x - 2; i += 2)

{

**for** (**int** j = 0; j < size\_y - 2; j += 2)

{

**if** (Arr[i][j] > Arr[i][j + 2])

{

n++;

}

}

}

**return** n;

}

};

**template** <**typename** type>

**class** Array

{

**friend** istream& **operator**>> (istream& input, Array& obj)

{

cout << "Для рандомного заповнення введіть - 1\nДля ручного заповнення введіть - 2\n";

**int** l = 1;

cin >> l;

**switch** (l)

{

**case** 1:

**for** (**int** ix = 0; ix < obj.size; ix++)

{

obj.ptr[ix] = rand() % 99;

}

**break**;

**case** 2:

**for** (**int** ix = 0; ix < obj.size; ix++)

{

cout << "a[" << ix << "]" << "=";

input >> obj.ptr[ix];

}

**break**;

}

**return** input;

}

**friend** ostream& **operator**<< (ostream& output, **const** Array& obj)

{

**for** (**int** ix = 0; ix < obj.size; ix++)

{

output << setw(5) << obj.ptr[ix];

}

output << endl;

**return** output;

}

**private**:

**int**\* ptr;

**int** size;

**public**:

Array()

{

size = 10;

ptr = **new** **int**[size];

**for** (**int** ix = 0; ix < size; ix++)

{

ptr[ix] = 0;

}

}

Array(**int** arraySize)

{

size = (arraySize > 0 ? arraySize : 10);

ptr = **new** **int**[size];

**for** (**int** ix = 0; ix < size; ix++)

{

ptr[ix] = 0;

}

}

~Array()

{

**delete**[] ptr;

}

**int** getSize() **const**

{

**return** size;

}

**int**& **operator**[] (**int** subscript)

{

**if** (subscript < 0 || subscript >= size)

{

std::cerr << "\n Ошибка индекса: " << subscript << std::endl;

exit(1);

}

**return** ptr[subscript];

}

**int** search()

{

**int** n = 1, max;

**for** (**int** i = 0; i < size - 2; i++, i++)

{

**if** (ptr[i] > ptr[i+2])

{

n++;

max = n;

}

**else**

{

max = n;

n = 1;

}

}

**return** max;

}

};

**int** main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

Array<**int**> Arr(10);

cin >> Arr;

cout << Arr;

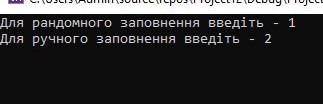
cout << endl << Arr.search() << endl;

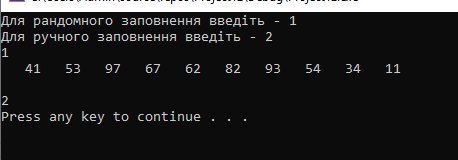
system("pause");

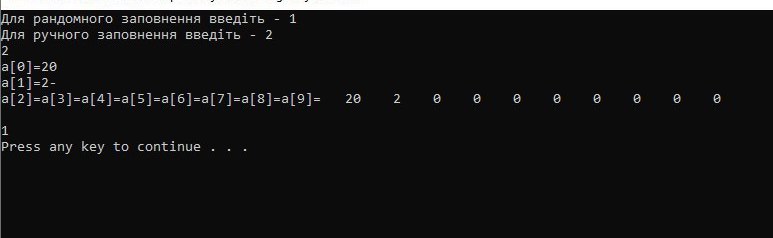
**return** 0;

}

**Результат виконання функції:**

****

****

****

**Висновок**: на даній лабораторній роботі я познайомилась з створенням шаблонів.