**Лабораторна робота №20.**  **Розробка програм із використанням шаблонів функцій та класів.**

**Мета:** Набуття навичок в розробці програм, де використовується шаблони функцій.

**Завдання**

1. Опрацювати теоретичні відомості та переглянути лекції № 27 "Шаблони функцій та класів. Класи колекцій стандартної бібліотеки*"*.
2. Створити шаблон функції, яка міняє місцями значення двох змінних. Викликати функцію для змінних цілого, дійсного, символьного типів. Вивести результати до і після обміну, використовуючи форматоване виведення.
3. Створити клас банківської транзакції, що містить два коди рахунків та код операції, які задаються параметрами, суму переводу (дійсне), а також містить метод, який виводить інформацію про транзакцію. В головній функції викликати функцію для екземплярів класів, де 1) рахунки задаються змінними типу string, а код операції цілим; 2) рахунки задаються цілим, а код змінною типу string. Продемонструвати виконання.
4. Пп. 2, 3 оформити однією програмою.

Результати надсилати на електронну адресу викладача

[t.i.lumpova@gmail.com](mailto:t.i.lumpova@gmail.com)у вигляді заголовкового файлу та cpp-файлу з іменем у форматі

Для заголовкового файлу

**<Прізвище англійською>.h**

Наприклад, Ivanov.h

Для cpp-файлу:

**<Номер групи> <Номер лабораторної><Прізвище англійською>.cpp**

Наприклад, 31-01Ivanov.cpp.

Тему в заголовку листа записати

**ООП<Номер групи>-ЛР <Номер лабораторної>-<Прізвищеанглійською>**

**Строк відсилки ЛР ІПЗ-31 24.11.2022**

Всі запитання, що виникнуть, надсилайте на електронну адресу викладача, тему в заголовку листа записати

**ООП<Номер групи>-Запитання-<Прізвище англійською>**.

**ТЕОРЕТИЧНІ ВІДОМОСТІ**

Із шаблонами зв'язано декілька понять.

***Шаблон функції***, або ***шаблонна функція*** (template function) ще має назву - ***узагальнена функція*** (тобто функція, оголошена за допомогою ключового слова template). Ці терміни є синонімами.

Конкретна версія узагальненої функції, створювана компілятором, називається ***спеціалізацією*** (specialization) чи ***згенерованою фун***к***цією*** (generated function).

Процес генерації конкретної функції називається ***конкретизацією*** (instantiation). Іншими словами, згенерована функція є конкретним екземпляром узагальненої функції.

Тип T, що указується в кутових дужках, називається ***параметром шаблону*** (або шаблонним параметром), а тип, що указується в списку параметрів (наприклад, int) — ***параметром виклику***. При утворенні шаблонної функції компілятор може автоматично генерувати стільки ї різних варіантів, скільки існує способів виклику цієї функції в програмі.

**Виведення аргументів**

Під час виклику функції (наприклад, max) параметри шаблону визначаються аргументами, що передаються в функцію. Якщо в якості параметрів типу T const& передається два значення int, компілятор робить висновок, що замість T слід підставити int. Автоматичне перетворення типів в шаблонних функціях не дозволяється. Відповідність типів параметрів і аргументів повинна бути точною.

**// Приклад вірного і помилкового вживання параметрів**

**template <typename T>**

**void max(T& a, T& b)**

**...**

**max(4,5); // Вірно: T == int для обох аргументів**

**max(4,5.5); // Помилка:перший T==int, другий T==double**

Існує три способи виправлення цієї помилки.

1. Привести обидва аргументи до одного типу: max(static\_cast<double>(4), 5.5);

2. Указати тип T явно max<double>(4, 5.5);

3. Задати різні типи параметрів шаблонів.

**Функція з кількома узагальненими типами**

Використовуючи список, елементи якого розділені комами, можна визначити кілька узагальнених типів даних в операторі template. Наприклад, у наступній програмі створюється шаблонна функція, що має два узагальнених типи.

**template <typename T1, typename T2>**

**inline T1 max (T1 const& a, T2 const& b)**

**{ return a < b? b : a; }**

Кількість параметрів шаблону необмежена, але в шаблонах функцій (на відміну від шаблонів класів) не можна використовувати аргументи шаблону за умовчанням. Можливість задавати декілька параметрів шаблону дозволяє розв’язати проблему виводу аргументів, але породжує нові. Проблема полягає в тому, що ми повинні оголосити тип значення, що повертається. Якщо для цього використати один із двох типів параметрів T1 або T2, аргумент для іншого параметру повинен конвертуватися в цей же тип, незалежно від волі програміста. В С++ немає способу задати правило вибору “найбільш потужного типу”. Отже, залежно від порядку слідування аргументів під час виклику можна отримати як найбільше число серед пари 4 і 5.5 і double, і int (тобто, 5.5 або 5). Крім того, при конвертуванні типу другого параметру в тип значення, що повертається, утворюється новий локальний тимчасовий об’єкт, що унеможливлює повертання результату за посиланням. Отже, в нашому прикладі, типом значення, що повертається, повинен бути T1, а не T1 const&.

Оскільки типи параметрів виклику конструюються із параметрів шаблону, вони зазвичай пов’язані один з одним. Ця концепція називається ***виводом аргументів шаблону функції*** і забезпечує можливість викликати шаблонну функцію так само, як і звичайну. В тих випадках, коли цей зв’язок відсутній, аргумент шаблона під час виклику необхідно задавати явно. Наприклад, можна ввести третій тип параметра шаблона, який задає тип значення, що повертає функція.

**template <typename T1, typename T2, typename RT >**

**inline RT max(T1 const& a, T2 const& b);**

Але механізм виводу аргументів шаблона не розповсюджується на типи значень, що повертаються, а серед типів параметрів виклику функції RT відсутній. Отже, для його визначення необхідно явно задати список аргументів шаблону.

**max<int,double,double>(4,5.5); //Вірно, але занадто обтяжливо**

Слід явно задавати всі типи аргументів, які не можна визначити неявно.

Отже, якщо в нашому прикладі змінити порядок слідування параметрів шаблону, то під час виклику знадобиться указати лише тип значення, що повертається.

**template < typename, RT typename T1, typename T2 >**

**inline RT max(T1 const& a, T2 const& b);**

**...**

**max(4,5.5); // Вірно, повертається double**

В даному випадку RT задається явно, а типи T1 і T2 виводяться із аргументів виклику як int і double.

Жодна з наведених версій не дає суттєвих переваг, отже, краще зупинитися на найпростішому варіанті — версії max() з одним параметром.

**Явна спеціалізація узагальненої функції**

Незважаючи на те что узагальнена функція перевантажує сама себе, її можна перевантажити явно. Цей процес називається ***явною спеціалізацією*** (explicit specialization). Перевантажена функція заміщає (чи “маскує”) узагальнену функцію, зв'язану з даною конкретною версією. Розглянемо модифіковану версію програми, призначеної для перестановки двох змінних.

// Перевантаження шаблонної функції.

**#include <iostream>**

**using namespace std;**

**template <typename T>**

**T max(T &a, T &b) { return a < b ? b : a; }**

**int max(int &a, int &b) { return a < b ? b : a; }**

**int main() {**

**int i=10, j=20;**

**double x=10.5, y=25.5;**

**char a='a', b='z';**

**cout << "i ? j: " << max(i,j) << '\n';**

**cout << "x ? y: " << max(x,y) << '\n';**

**cout << "a ? z: " << max(a,b) << '\n';**

**return 0; }**

Ця програма виводить на екран наступні рядки.

i ? j: 20

x ? y:

25.5 a ? z: z

Існує альтернативна синтаксична конструкція, призначена для позначення явної спеціалізації функції. Цей метод використовує ключове слово template. Наприклад, перевантажену функцію max() з попереднього прикладу можна переписати в такий спосіб.

**#include <iostream>**

**using namespace std;**

**template <typename T>**

**T max(T &a, T &b) { return a < b ? b : a; }**

**template<> int max< int > (int &a, int &b)**

**{ return a < b ? b : a; }**

**…**

Як бачимо, новий спосіб визначення спеціалізації містить конструкцію template<>. Тип даних, для якого призначена спеціалізація, вказується усередині кутових дужок після імені функції. Для спеціалізації будь-якого іншого типу узагальненої функції використовується така ж синтаксична конструкція. В даний час обидва способи визначення спеціалізації еквівалентні.

**Явна конкретизація узагальненої функції**

***Конкретизація шаблонів*** — це процес, під час якого на основі узагальненого визначення шаблонів генеруються типи і функції. ***Спеціалізація*** — це конкретний екземпляр шаблона. Коли компілятор зустрічає використання спеціалізації шаблону, від утворює його, підставлюючи замість параметрів шаблону необхідні аргументи. Ці дії виконуються автоматично і не вимагають жодних указівок в коді або визначенні шаблону. Такий процес називають ***неявною***, або ***автоматичною конкретизацією***.

***Точка конкретизації*** утворюється в тому випадку, коли деяка конструкція вихідного коду посилається на спеціалізацію шаблона таким чином, що для цієї спеціалізації потрібно виконати конкретизацію шаблона. Точка конкретизації — це місце коду, в яке можна вставити шаблон с підставленими аргументами.

Існує три способи явної конкретизації.

**#include <iostream>**

**using namespace std;**

**template <typename T>**

**T max(T &a, T &b) { return a < b ? b : a; }**

**// Перший спосіб:**

**template char max(char &a, char &b);**

**// Другий спосіб :**

**template double max<>(double &a, double &b);**

**// Третій спосіб ;**

**template float max < float > (float &a, float &b);**

**int main() {**

**int i=10, j=20;**

**double x=10.5, y=25.5;**

**char a='a', b='z';**

**cout << "i ? j: " << max(i,j) << '\n';**

**cout << "x ? y: " << max(x,y) << '\n';**

**cout << "a ? z: " << max(a,b) << '\n';**

**return 0; }**

В програмі повинно бути не більше однієї явної конкретизації для визначеної спеціалізації шаблону.

Розглянемо ситуацію, в якій реалізується бібліотека. Нехай перша версія шаблону функції виглядає так.

**// Файл max.hpp**

**template <typename T>**

**T max(T const& x, T const& y)**

**{ return a < b ? b : a; }**

Користувач бібліотеки може включити наведений вище заголовний файл і явно конкретизувати шаблон, що в ньому міститься.

**// Код користувача**

**#include "max.hpp"**

**template int max(int, int);**

**Перевантаження шаблонної функції**

Для того щоб перевантажити специфікацію узагальненої функції, достатньо створити ще одну версію шаблона, що відрізняється від інших своїм списком параметрів.

// Перевантаження шаблонної функції.

**#include <iostream>**

**using namespace std;**

**// Перша версія шаблонної функції f().**

**template <typename T>**

**void f(T a) { cout << "Inside f(T a)\n"; }**

**// Друга версія шаблонної функції f().**

**template <typename T, typename Y >**

**void f(T a, Y b) { cout << "Inside f(T a, Y b)\n"; }**

**int main() {**

**f(10); // Виклик функції f(T).**

**f(10, 20); // Виклик функції f(T, Y).**

**return 0; }**

**Використання стандартних параметрів шаблонних функцій**

При визначенні шаблонної функції можна змішувати стандартні й узагальнені параметри. У цьому випадку стандартні параметри нічим не відрізняються від параметрів будь-яких інших функцій. Розглянемо приклад.

**// Застосування стандартних параметрів у шаблонній функції.**

**#include <iostream>**

**using namespace std;**

**const int TABWIDTH = 8; // Виводить на екран дані в позиції tab.**

**template <typename T> void tabOut(T data, int tab)**

**{ for(; tab; tab--)**

**for(int i=0; i< TABWIDTH; ++) cout <<' ';**

**cout << data << "\n"; }**

**int main() {**

**tabOut("Перевірка", 0);**

**tabOut(100, 1);**

**tabOut('T', 2);**

**tabOut(10/3, 3);**

**return 0; }**

Програма виводить на екран наступні повідомлення.

Перевірка

100

T

3

**Обмеження на узагальнені функції**

Узагальнені функції нагадують перевантажені, але на них накладаються ще більш жорсткі обмеження. При перевантаженні усередині тіла кожної функції можна виконувати різні операції. У той же час узагальнена функція повинна виконувати ту саму універсальну операцію для усіх версій, розрізнятися можуть лише типи даних. Розглянемо перевантажену функцію на наступному прикладі. Ці функції не можна заміняти узагальненими, оскільки вони мають різне призначення.

**#include <iostream>**

**#include <cmath>**

**using namespace std;**

**void myfunc(int i)**

**{ cout << "Значення = " << i << "\n"; }**

**void myfunc(double d) {**

**double intpart, fracpart; f**

**racpart = modf(d, &intpart);**

**cout << "Дробова частина = " << fracpart << endl;**

**cout << "Ціла частина = " << intpart; }**

**int main() { myfunc(1); myfunc(12.2); return 0; }**

**Узагальнені класи**

**Приклад використання двох узагальнених типів даних**

Шаблонний клас може мати декілька шаблонних типів. Для цього їх достатньо перелічити в списку шаблонних параметрів в оголошенні template. Наприклад, наступна програма створює клас, що використовує два узагальнених типи.

**/\* Приклад класу, що використовує два узагальнених типи\*/**

**#include <iostream>**

**using namespace std;**

**template < Type1, Type2> class myclass**

**{ Type1 i; Type2 j;**

**public: myclass(Type1 a, Type2 b) { i = a; j = b; }**

**void show() { cout << i << ' ' << j << '\n'; } };**

**int main()**

**{ myclass ob1(10, 0.23);**

**myclass ob2('X', "Шаблони — могутній механізм."); ob1.show(); // Виводимо ціле і дійсне число.**

**ob2.show(); // Виводимо символ і покажчик на символ. return 0; }**

Ця програма виводить наступні результати.

10 0.23

X Шаблони — могутній механізм.

У програмі з'являються об'єкти двох типів. Об'єкт ob1 використовує цілі і дійсні числа. Об'єкт ob2 використовує символ і покажчик на символ. В обох випадках при створенні об'єктів компілятор автоматично генерує відповідні дані й функції.