**Лекція №27. Шаблони функцій та класів. Класи колекцій стандартної бібліотеки**

**Шаблони функцій**

У процесі розв’язання багатьох задач необхідно використовувати функції, в яких алгоритм обчислення однаковий, а типи даних відрізняються. Прикладом є задачі пошуку і сортування. Особливістю програмування таких задач мовою С++ є використання шаблонів функцій. Шаблонні функції і шаблони типів є основними елементами *узагальненого програмування* у C++.

*Шаблонні функції* (template functions) призначені для запису узагальнених функцій, що можуть працювати з даними різних типів.

Шаблони функцій — потужний засіб параметризації. За допомогою шаблона функції можна визначити алгоритм, який буде застосовуватися до даних різних типів, а конкретний тип даних передається функції у вигляді параметра на етапі компіляції.

Шаблон функції — це деяка узагальнена функція (родова функція) для сімейства функцій, призначених для розв’язання даної задачі. Визначається така шаблонна функція у заголовному файлі і має такий вигляд:

**template <class Т>**

**type\_func my\_func (type paraml, type param2, …)**

**{**

**// оператори тіла функції**

**}**

Де **template <class T>** — зарезервований вираз (заголовок шаблону), який вказує компілятору оголошений користувачем ідентифікатор типу Т;

**type\_func** — тип шаблонної функції;

**my\_func** — довільний ідентифікатор шаблонної функції;

**type param1, type param2** — формальні параметри, з яких хоча б один повинен мати або наведений у заголовку **(template cclass type>)** тип **Т**, або покажчик**\*param** на змінну типу **Т (Т \*param)**, або посилання **&param** на змінну типу**Т (Т&param)**;

**оператори тіла функції** — схема реальних операторів, що генеруються компілятором у відповідну функцію, враховуючи тип даних, вказаних при виклику.

Список формальних параметрів шаблона не може бути порожнім.

У шаблоні функції може бути оголошено декілька формальних типів даних, а також використано параметри означених раніше типів. Наприклад:

**template <class ТІ, class T2>**

**typefunc my\_func(Ti a,double x,T2 b,int c.char s)**

**{**

**//оператори тіла функції**

**}**

Таким чином, оголошення шаблонів функцій завжди починається з ключового слова **template (шаблон)**, за ним у кутових дужках визначається список формальних типів, перед кожним з яких вказується ключове слово **class** (або **typename**, що більше відповідає сучасному стандарту) за яким йде ідентифікатор. Ім'я формального параметра в списку повинне бути унікальним. Далі йде звичайний опис функції. При цьому формальні типи, представлені у заголовку шаблону, можна використовувати в опису функції для задання типів аргументів функції, типу значення, що повертається, а також для оголошення змінних усередині тіла функції.

Формальні параметри шаблонів можуть використовуватися для визначення типу результату і формальних параметрів шаблонної функції. У тілі шаблонної функції також можуть використовуватися формальні параметри шаблона.

**Приклад 1.** Написати шаблон функції, що повертає мінімальний елемент масиву, застосувати цю функцію для обробки масивів різних типів.

**/\* Використання шаблону функції для обчислення мінімального елемента масивів різних типів \*/**

**#include <iostream>**

**#include <conio.h> //- шаблон функції**

**#include <Windows.h>**

**using namespace std;**

**template <class T> T minmas (T \*a, int k)**

**{ T min = a[0];**

**for (int i = 1; i < k; i++)**

**if (a[i] < min) min = a[i];**

**return min;**

**}**

**int main() //--- головна функція**

**{system("color F0");**

**int b[ ]={1, 6, 8, 5, 9, -6, 4, -5, 2}; //масив цілих чисел**

**//---------------- виклик функції minmas() та виведення результатів**

**cout <<" min array b[ ]= "<<minmas(b, sizeof(b)/sizeof(int));**

**cout << endl;**

**float c[ ]={-4.5, 6.4, 7.0, -6.3, 2.1}; //масив дійсних чисел**

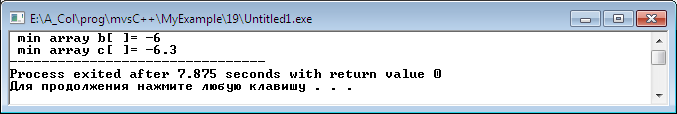
**cout <<" min array c[ ]= "<<minmas(c, sizeof(c)/sizeof(float));**

**getch();**

**return 0;**

**}**

Результат обчислень:



У заголовку шаблону цієї функції оголошено єдиний формальний параметр Т як тип даних, що повинні оброблятися функцією minmas(). У заголовку функції параметр Т використовується для завдання типу значення функції, що повертається (Т minmas), та для задання типу покажчика \*а. Усередині функції параметр **Т** застосовано для визначення типу локальної змінної тіла. Завдяки цьому шаблону у програмі можна обробляти масиви різних типів.

**Опис та використання шаблонних функцій**

Як ще один приклад шаблона приведемо функцію підсумовування елементів масиву довільного типу. Головне, щоб для елементів масиву були визначені операції присвоювання, у тому числі присвоювання константи "нуль", і "+=".

**template** <**class** SomeType>

SomeType sumOfArray(SomeType \*a, **const** **int** size)

{

SomeType sum = 0;

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

{

sum += a[i];

}

**return** sum;

}

Крім того, можна визначити шаблон функції виведення в стандартний потік елементів масиву:

**template** <**class** SomeType>

**void** printArray(SomeType \*a, **const** **int** size)

{

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

{

cout << a[i] << ' ';

}

cout << endl;

}

Конкретні визначення функцій, що відповідають шаблону, компілятор генерує під час виклику шаблонної функції з параметрами конкретного типу. Для описаного вище приклада можна запропонувати таке використання шаблонних функцій:

**int** main()

{

**int** a[] = {1, 2, 3};

printArray(a, 3);

cout << sumOfArray(a, 3) << endl;

**double** b[] = {1.1, 2.2, 3.3, 4.5};

printArray(b, 4);

cout << sumOfArray(b, 4) << endl;

cin.get();

**return** 0;

}

Шаблонна функція може перевантажуватися за умови, що список формальних параметрів кожного варіанта відрізняється від інших або типами параметрів, або їхнім числом.

**Порядок звернення до шаблонних функцій**

Бувають випадки, коли для якихось конкретних типів потрібно дати особливе визначення шаблонної функції. У цьому випадку програміст повинен задати свій спеціальний варіант функції. Наприклад, шаблон функції min() працює для типів, для яких визначена операція "<":

**template** <**class** Type>

Type min(Type a, Type b)

{

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

}

Цей шаблон не підходить для варіанта порівняння рядків. Для них визначається спеціальний варіант функції:

**char**\* min(**char**\* s1, **char**\* s2)

{

**return** strcmp(s1, s2) < 0 ? s1 : s2;

}

Порядок виклику функцій буде таким.

1. Досліджуються всі нешаблонні варіанти функції.
2. Досліджуються всі шаблонні варіанти функції.
3. Повторно досліджуються всі нешаблонні варіанти функції, з застосуванням перетворення типів.

Для того, щоб можна було конкретизувати шаблон, компілятор повинен бачити не тільки оголошення, але і визначення функції. Тому визначення шаблонних функцій можна і треба поміщати в заголовні файли.

Під час виклику функції фактичний параметр шаблона можна вказати явно, наприклад:

**int** i = min<**int**>(2, 3);

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.

**Шаблони класів**

Досить часто при використанні ООП виникає необхідність введення великої кількості класів, які виконують однакові дії і відрізняються лише типами даних, по відношенню до яких ці дії застосовуються. Для спрощення виконання цієї задачі в С++ передбачені шаблони класів. ***Шаблони класів*** це елементи мови програмування, які дозволяють визначити структуру сімейства класів, за якою компілятор самостійно створює потрібні класи, грунтуючись на параметрах настройки, що задаються. Цей механізм аналогічний механізму шаблонів функцій.

*Шаблон класу* (class template) можна використовувати для створення сімейства класів, які відрізняються типами або константними значеннями всередині опису.

Попереднє оголошення і визначення шаблону класу починається зі службового слова **template**. За ним іде список формальних параметрів шаблона типу. Цей список не може бути порожнім. Наприклад:

**template** <**class** T> **class** X

{

T t;

**public**:

**void** set(T t1) { t = t1; }

};

Шаблон класу не є класом. *Інстанціювання шаблону* (template instantiation) - це створення певних типів з шаблону. Такі класи мають назву *екземплярів шаблону* (template instances):

X<**int**> xi;

X<**double**> xd;

Примітка: шаблони класів іноді умовно називають *узагальненими класами* (generic classes).

Параметрами шаблонів можуть бути параметри-типи, параметри звичайних типів і параметри-шаблони. У шаблона може бути кілька параметрів. Цілі аргументи використовуються найчастіше для завдання розмірів і границь масивів. Цілий аргумент шаблона повинен бути константою. Можливе завдання усталених параметрів шаблона:

**template** <**class** T, **int** size = 64> **class** Y

{

. . .

};

Фактичне значення цілого параметру повинно бути константним виразом:

**const int** N = 128;

**int** i = 256;

Y<**int**, 2\*N> b1;// OK

Y<**float**, i> b2;// Помилка: i не константа

Найбільш типовий приклад використання шаблонів класів – це створення контейнерних класів, наприклад, векторів для розміщення об'єктів довільних типів.

Приклад:

template <class T>//шаблон класу вектор

class Vector

{

private:

T \*elements;

int size;

public:

Vector(int razm=0); //конструктор, його реалізація має особливості

//деструктор, його реалізація також може мати особливості

~Vector(){delete elements;}

//перевантажений оператор для класу

T& operator[](int i){return elements[i];}//перевантажений оператор-метод

//метод, його реалізація має особливості

void print\_contents();

};

//конструктор, його реалізація має особливості

template <class T>

Vector<T>::Vector(int razm)

{

elements=new T[razm];

for(int i=0; i< razm; elements[i]=(T) 0 i++);

size=razm;

};

//метод, реалізація якого має особливості

template <class T>

void Vector<T>::print\_contents()

{

cout << "elements num-"<<size<<"\n";

for(int i=0; i<size; i++)

cout <<"el["<<i<<"]="<<elements[i] <<"\n";

}

//головна функція

//зверніть увагу на визначення типу для кожного об’єкту

int main()

{

int razmer=10;

Vector <int> i(razmer);

Vector <float> x(razmer);

Vector <char> p(razmer);

…

return 0;

}

Зверніть увагу на те, що заголовок шаблону класу починається з ключового слова template і містить вказівку на те, що тип наперед невідомий і повинен вказуватись при об’вленні об’єкту <class T>. Замість літери Т може бути використана інша літера. Головне, щоб у всіх методах і полях класу, де буде оброблюватись інформація даного типу стояла та ж сама літера. Це дасть змогу компілятору правильно сформувати об’єкт класу для заданого типу. При розробці шаблонів класів часто виникає проблема перевантаження операторів. Це пов’язано з тим, що з одного боку для розроблених програмістом класів, як правило, немає стандартних операторів, а з іншого боку дуже зручно, коли аналогічні операції для різних типів позначаються у програмі однаковими операторами.

Шаблони класів допускають використання механізму спадкування. Можливе створення похідного шаблону класу як від шаблона, так і від класу, що не є шаблоном.

Функція-елемент шаблонного класу вважається неявною шаблонною функцією, а параметри шаблона типу для її класу - параметрами її шаблона. Для деяких типів стандартні функції-елементи не підходять. У таких випадках можна явно задавати реалізацію функції, розрахованої на конкретний тип. Перед реалізацією таких функцій потрібно спеціальне оголошення **template**<> без параметрів. Крім того, можна дати особливе визначення шаблонного класу, розраховане на конкретний тип. Функція-друг для шаблона типу не є неявною шаблонною функцією.

Шаблон класу може мати статичні елементи. Кожен клас, згенерований по шаблону, має свою копію статичних елементів.

**Шаблони класів і відповідність типів**

Створені за одним шаблоном типи будуть різними і між ними неможливе відношення спадкування крім єдиного випадку, коли в цих типів ідентичні параметри шаблона. Наприклад:

**template**<**class** T> **class** X

{

/\* ... \*/

}

X<**int**> x1;

X<**short**> x2;

X<**int**> x3;

Тут x1 і x3 одного типу, а x2 - зовсім іншого. Автоматичне приведення типів не здійснюється:

x2 = x3; // помилка

**Переваги використання шаблонів класів**

Оголошення шаблону класу дає такі переваги:

* уникнення повторюваності написання програмного коду для різних типів даних. Програмний код (методи, функції) пишеться для деякого узагальненого типу T;
* зменшення текстової частини програмного коду, і, як наслідок, підвищення читабельності програм;
* забезпечення зручного механізму передачі аргументів у шаблон класу з метою їх обробки методами класу.

Приклад. Оголошується шаблон класу, що містить методи, які виконують наступні операції над деяким числом:

* множення числа на 2;
* ділення одного числа на інше. Для цілих типів виконується ділення націло;
* взяття квадрату числа (степінь 2).

Оголошення шаблону має вигляд

// шаблон класу, що реалізує число різних типів

template <class T>

class MyNumber

{

public:

// конструктор

MyNumber(void) { }

// метод, що множить на 2 число

void Mult2(T\* t);

// метод, що повертає квадрат числа для деякого типу T

T MySquare(T);

// метод, що ділить два числа типу T і повертає результат типу T

T DivNumbers(T, T);

};

// реалізація методу, що множить число на 2

template <class T> void MyNumber<T>::Mult2(T\* t)

{

\*t = (\*t)\*2;

}

// реалізація методу, що повертає квадрат числа

template <class T> T MyNumber<T>::MySquare(T number)

{

return (T)(number\*number);

}

// метод, що ділить 2 числа і повертає результат від ділення

template <class T> T MyNumber<T>::DivNumbers(T t1, T t2)

{

return (T)(t1/t2);

}

Використання шаблону класу MyNumber в іншому програмному коді

MyNumber <int> mi; // об'єкт mi класу працює з типом int

MyNumber <float> mf; // об'єкт mf працює з типом float

int d = 8;

float x = 9.3f;

// множення числа на 2

mi.Mult2(&d); // d = 16

mf.Mult2(&x); // x = 18.6

// взяття квадрату з числа

int dd;

dd = mi.MySquare(9); // dd = 81 - ціле число

double z;

z = mf.MySquare(1.1); // z = 1.21000... - дійсне число

// ділення чисел

long int t;

float f;

t = mi.DivNumbers(5, 2); // t = 2 - ділення цілих чисел

f = mf.DivNumbers(5, 2); // f = 2.5 - ділення дійсних чисел

**Оголошення шаблону класу, який приймає аргументи**

Бувають випадки, коли в шаблоні класу потрібно використовувати деякі аргументи. Ці аргументи можуть використовуватись методами, що описуються в шаблоні класу.

Загальна форма шаблону класу, що містить аргументи, наступна:

template <class T, ***type1*** *var1,* ***type2*** *var2, ...,* ***typeN*** *varN*> class ClassName

{

// тіло шаблону класу

// ...

}

де

* *T* – деякий узагальнений тип даних;
* *type1*, *type2*, …, *typeN* – конкретні типи аргументів з іменами *var1*, *var2*, …, *varN*;
* *var1*, *var2*, …, *varN* – імена аргументів, що використовуються у шаблоні класу.

Загальна форма оголошення об’єкту шаблонного класу, який містить один аргумент:

ClassName <***type***, *arg*> *objName*;

де

* *ClassName* – ім’я шаблонного класу;
* *type* – конкретний тип даних, для якого формується реальний клас;
* *arg* – значення аргументу, яке використовується у шаблоні класу;
* *objName* – ім’я об’єкту шаблонного класу.

Приклад використання шаблону класу, який приймає два аргументи

У прикладі реалізується шаблон класу CMyArray, що містить методи обробки масиву чисел. Тип елементів масиву може бути дійсним або цілим.

Шаблон класу отримує два цілих числа:

* count – кількість елементів масиву. Використовується при ініціалізації класу з допомогою конструктора з 1 параметром;
* num – число, що служить для проведення операцій над масивом.

Ці числа використовуються у методах для виконання операцій над масивом. Шаблон класу містить такі дані та методи:

* кількість елементів масиву n;
* масив елементів (чисел) A заданої розмірності (10);
* метод Power(), що здійснює піднесення елементів масиву A у степінь num, що є вхідним параметром (аргументом);
* метод CalcNum(), що здійснює підрахунок кількості елементів, що є більше заданого параметру num.

Текст шаблону класу наступний:

// шаблон класу, що отримує 2 параметри

template <class TT, int count, int num> class CMyArray

{

private:

int n; // кількість елементів масиву

TT A[10]; // масив елементів

public:

// конструктор класу без параметрів

CMyArray()

{

// кількість елементів беремо з вхідного параметра count

n = count;

// заповнити масив довільними значеннями

for (int i=0; i<n; i++)

A[i] = (TT)(i\*2);

}

// конструктор класу з 1 параметром

CMyArray(int cnt)

{

if (cnt<=10) n = cnt;

else n = 0;

// заповнення масиву довільними значеннями

for (int i=0; i<n; i++)

A[i] = (TT)(i\*2);

}

// методи доступу

int GetN(void) { return n; }

void SetN(int n)

{

if (n<=10) this->n = n;

else n=0;

for (int i=0; i<n; i++)

A[i] = (TT)(i\*2);

}

// метод, що читає комірку масиву з заданим індексом

TT GetItem(int index) { return (TT)A[index]; }

// методи, що виконують операції над масивом A

// піднесення елементів масиву до степеня num

void Power(void);

// підрахунок кількості елементів, що є більше заданого аргументу

int CalcNum(void);

};

// метод, що підносить елементи масиву до степеня num

template <class TT, int count, int num>

void CMyArray<TT, count, num>::Power(void)

{

if (n<0) return;

for (int i=0; i<n; i++)

A[i] = System::Math::Pow(A[i], num);

}

// метод, що визначає кількість елементів масиву,

// які більше заданого числа num (num - вхідний параметр)

template <class TT, int count, int num>

int CMyArray<TT, count, num>::CalcNum(void)

{

int k = 0;

// цикл підрахунку

for (int i=0; i<n; i++)

if (A[i] > num)

k++;

return k;

}

Використання шаблону у деякому іншому програмному коді (функції, методі)

// використання шаблону класу CMyArray

// масив цілих чисел, параметри: count=7, num=2

CMyArray <int, 7, 2> ai1;

// масив цілих чисел, виклик конструктора з 1 параметром

CMyArray <int, 8, -3> ai2(6); // к-сть елементів count = 6, num=-3

// масив дійсних чисел типу double, виклик конструктора без параметрів

CMyArray <double, 4, 5> ad1;

// перевірка

int n, t;

double x;

n = ai1.GetN(); // n = 7

n = ai2.GetN(); // n = 6

n = ad1.GetN(); // n = 4

// перевірка масиву

t = ai1.GetItem(3); // t = 6

t = ai2.GetItem(0); // t = 0

x = ad1.GetItem(2); // x = 4.0

// виклик методів обробки масиву та перевірка результату

ai1.Power(); // піднесення кожного ел-ту масиву до степеня num=2

t = ai1.GetItem(3); // t = 6^2 = 36

// підрахунок к-сті елементів, які більше за -3

// всього у масиві класу ai2 6 елементів

t = ai2.CalcNum(); // t = 6

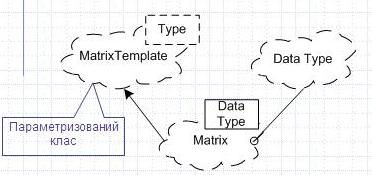
// робота з класом, що обробляє тип double

x = ad1.GetItem(3); // x = 6.0

ad1.Power(); // піднести усі числа масиву до степеня num = 5

x = ad1.GetItem(3); // x = 6.0^5 = 7776

**Відношення інстанціювання в нотації Буча**



*Для самостійного вивчення*: Поглибити матеріал лекції за наданою літературою. Вивчення лекційного матеріалу та додаткових джерел. Розгляд запитань і виконання завдань для самостійної роботи, запропонованих на лекції.

*Література*

1. Джейс Либерти Освой самостоятельно С++ за 21 день: 3-е изд. пер. с англ.: Уч. пос. – М.: Издательский дом „Вильямс”, 2001. – 816 с.: ил..
2. Павловская Т.А. С/С++. Программирование на языке высокого уровня. СПб.: Питер, 2003. – 461 с. URL:  <http://www.ph4s.ru/bookprogramir_1.html>
3. Липпман С. Б., Лажойе Ж. Язык программирования С++: Вводный курс. — М.: ДМК, 2001. URL: <http://www.insycom.ru/html/metodmat/inf/Lipman.pdf>
4. Дейтел Х., Дейтел П. Основы программирования на С++. – М.: Бином, 1999. – 1024 с. URL:  <http://ijevanlib.ysu.am/wp-content/uploads/2018/03/deytel.pdf>
5. Бадд Т. Объектно-ориентированное программирование в действии. [2-е изд.] – СПб.: Изд-во "Питер". 1997.  URL: <http://khizha.dp.ua/library/Timothy_Budd_-_Introduction_to_OOP_(ru).pdf>.
6. *Герб Саттер,Андрей Александреску* "Стандарты программирования на С++ ", [*Вильямс, 2005*](http://www.williamspublishing.com/)*; 304 с.*
7. *Скотт Мейерс.* Эффективное использование C++. 50 рекомендаций по улучшению ваших программ и проектов. *"ДМК", 2000; 240 с.*
8. *Скотт Мейерс.* Наиболее эффективное использование C++. 35 новых рекомендаций по улучшению ваших программ и проектов.*"ДМК",2000;304 с.*
9. Г.Буч. Об‘єктно-орієнтоване проектування з прикладами застосування. – К.: Видавничий центр "Академія". 2002. ­ 499 с.
10. Страуструп Б. Язык программирования С++· Киев: "ДиаСофт", 1993. - 256 с.

*Контрольні запитання для самоперевірки*.

1. Для чого використовуються шаблони функцій?
2. В якому випадку доцільно використовувати шаблони класів?
3. Які переваги дає використання шаблонів класів?