**Практична робота №14.** Написання програм з використанням шаблонів функцій

**Завдання.**

Написати на мові С++ програму введення даних для розрахунку формул з передачею параметрів за різними механізмами.

1. Ознайомитися з теоретичною частиною.
2. Написати програму, яка використовує дві шаблонні функції.

2.1. Створити шаблон функції, яка міняє місцями значення двох змінних. Викликати функцію для змінних цілого, дійсного, символьного типів. Вивести результати до і після обміну.

2.2. Створити шаблон функції, яка знаходить максимальне значення. Викликати функцію для змінних цілого, дійсного, символьного типів. Вивести результати до і після обміну.

1. Вхідні дані ввести, а результати вивести, використовуючи потокове введення-виведення даних. В першому рядку кожної програми записати

*// ОПІ-41 Група № Прізвище Номер ЛР*

1. Результати надсилати на електронну адресу викладача [**t.i.lumpova@gmail.com**](mailto:t.i.lumpova@gmail.com)у вигляді cpp-файлу з іменем у форматі

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

Наприклад, МІВТ-МНТ-ЕТ-41-07Ivanov.cpp.

Іншим рішенням є надсилання поштою посилання на текст програми за URL адреси, яку надає C++Shell, вказавши в темі листа, номер групи прізвище студента та номер ПР.

В темі листа вказати, номер групи, прізвище студента та номер ПР як "ПР№5".

**Строк відсилки ЛР для МНТ/ЕТ - 41 30.03.2024**

Всі запитання, що виникнуть, надсилайте на електронну адресу викладача, В темі листа вказати, номер групи, прізвище студента та номер ПР як "ПР№7 В темі листа вказати, номер групи, прізвище студента, номер ПР та фразу "Запитання".

**Теоретичні відомості.**

**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(TІ 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 завдання 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**

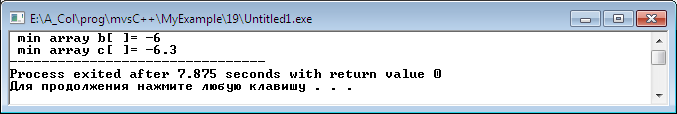
**cout <<" min array c[ ]= "<<minmas(c, sizeof(c)/sizeof(float));**

**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 завдання 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 що 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, підставляючи 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; і++) 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.

**Виклик функції з використанням параметра-посилання** здійснює передачу до функції не самої змінної, а тільки посилання на неї. У цьому випадку забезпечується доступ до переданого параметра за допомогою визначення його альтернативного імені, бо посилання є синонімом імені параметра. Тоді всі дії, що відбуваються над посиланням, є діями над самою змінною. Такий спосіб передачі параметрів і повернення результату передбачає запис у списку фактичних параметрів імені змінної, а у списку формальних — параметрів-посилань. Наприклад:

//-------------- **використання параметра-посилання**

**void fun (int &p)** //---------- функция fun()

**{ ++p;**

**cout <<** **"p = " << p << endl;** **}**

**void main** **( )**

**{ int x** **= 10;**

**fun (x);** //---------- вызов функции fun()

**cout <<** **"x="** **<<** **x << endl; }**

Одержимо результат попереднього фрагмента, тобто **р = 11 і х=11.**

При застосуванні вказаних вище параметрів у функцію передаються не значення змінних, а їхні адреси, тому можливо міняти значення цих змінних усередині функції і передавати за її межі (в інші функції). Коли необхідно, щоб деякі параметри не змінювали свої значення всередині функції, їх слід оголосити як параметри-константи, використовуючи модифікатор **const**.

**Покажчики** — це змінні, котрі містять адресу пам’яті, розподіленої для об’єкта відповідного типу. При оголошенні змінної-покажчика слід вказати тип даних, адресу яких буде містити змінна, та ім’я покажчика з символом «\*».

Загальний формат опису покажчика має вигляд:

**тип \*ім’я;**

де **тип** — тип значень, на який вказує покажчик;  
**ім’я** — ім’я змінної-покажчика;  
«\*» — операція над типом, що читається «покажчик на тип».

Наприклад:

**int \*рn** – покажчик на ціле значення;  
**float \*pf1, \*pf2;** — два покажчики на дійсні значення.

Покажчики не прив’язують дані до якого-небудь визначеного імені змінної і можуть містити адреси будь-якого неіменованого значення. Існує адресна константа **NULL**, що означає порожню адресу.

Мова C++ налічує лише дві операції, які стосуються адрес змінних, а саме:

**«&»** — **операція взяття адреси** («адреса значення»);

**«\*»** — **операція розіменування** («значення за адресою»).

Операція взяття адреси **«&»** застосовується разом зі змінною і повертає адресу цієї змінної. Операція розіменування «\*» використовується разом з покажчиками і бере значення, на яке вказує змінна-покажчик, розташована безпосередньо після символу «\*».

Оголошення покажчиків можна здійснити одним з таких способів:

**<тип> \*ptr;**  
**<тип> \*ptr = <змінна-покажчик>;**  
**<тип> \*ptr = &<ім’я змінної>;.**

Наприклад:  
**int \*ptx, b; float у;** — оголошені змінна-покажчик **ptx** та змінні **b і у**;

**float \*sp = &у;** — покажчику **sp** присвоюється адреса змінної **у**;

**float \*р = sp;** — покажчику **р** присвоюється значення (адреса значення), яке міститься в змінній **sp**, тобто адреса змінної **у**.

При оголошенні покажчиків символ «\*» може знаходитися перед ім’ям покажчика або відразу після оголошення типу покажчика і поширювати свою дію тільки на одну змінну-покажчик, перед якою він записаний:

**long \*pt;   long\*Uk;   int \*ki, x, h;** — оголошення описів.

**Виклик функції з використанням параметра-посилання** здійснює передачу до функції не самої змінної, а тільки посилання на неї. У цьому випадку забезпечується доступ до переданого параметра за допомогою визначення його альтернативного імені, бо посилання є синонімом імені параметра. Тоді всі дії, що відбуваються над посиланням, є діями над самою змінною. Такий спосіб передачі параметрів і повернення результату передбачає запис у списку фактичних параметрів імені змінної, а у списку формальних — параметрів-посилань. Наприклад:

//-------------- **використання параметра-посилання**

**void fun (int &p)** //---------- функция fun()

**{ ++p;**

**cout <<** **"p = " << p << endl;** **}**

**void main** **( )**

**{ int x** **= 10;**

**fun (x);** //---------- вызов функции fun()

**cout <<** **"x="** **<<** **x << endl; }**

Одержимо результат попереднього фрагмента, тобто **р = 11 і х=11.**

При застосуванні вказаних вище параметрів у функцію передаються не значення змінних, а їхні адреси, тому можливо міняти значення цих змінних усередині функції і передавати за її межі (в інші функції). Коли необхідно, щоб деякі параметри не змінювали свої значення всередині функції, їх слід оголосити як параметри-константи, використовуючи модифікатор **const**.

В C++ передача за посиланням може здійснюватися двома способами:

* використовуючи безпосередньо посилання;
* за допомогою вказівників.

Синтаксис передачі з використанням посилань має на увазі застосування як аргумент посилання на тип об'єкта. Наприклад, функція

**double** Glue ( **long**& x, **int**& y ) ;

одержує два посилання на змінні типу **long** і **int**. При передачі у функцію параметра за посиланням компілятор автоматично передає у функцію адресу змінної, зазначеної в якості аргументу. Ставити знак амперсанта перед аргументом у виклику функції не потрібно. Наприклад, для попередньої функції виклик з передачею параметрів за посиланням виглядає в такий спосіб:

c = Glue (a, b) ;

Приклад прототипу функції при передачі параметрів через вказівник:

**void** Setnumber ( **int**\*, **long**\* ) ;

Тоді виклик функції має наступний вигляд:

Setnumber (&n,&a ) ;

Приклад функції, яка приймає в якості параметра дві змінні, та міняє їх місцями(параметри передаються з використанням посилань):

void swap(int &x, int &y)

{

int temp=x;

x=y;

y=temp;

}

int \_tmain()

{

int a=3,b=4;

swap(a,b);

cout<<"a="<<a<<"b="<<b<<'\n';

system("pause");

return 0;

}

Приклад функції, яка приймає в якості параметра дві змінні, та міняє їх місцями(параметри передаються з використанням вказівників):

void swap (int \*x, int \*y)

{

int temp=\*x;

\*x=\*y;

\*y=temp;

}

int \_tmain()

{

int a=3,b=4;

swap (&a,&b);

cout<<"a="<<a<<"b="<<b<<'\n';

system("pause");

return 0;

}

Якщо передати параметри за значенням, зміни не будуть збережені, бо в функцію буде передано копії змінних, а не їх адреси.