Нові можливості функцій в мові C++

Вбудовані функції. Це невеликі за обсягом коду функції, які позначаються службовим словом **inline** (і у декларації, і у визначенні функції). Ідея їх використання полягає в тому, що код такої функції просто додається до програми (“вбудовується”) в точці виклику. Виграш при використанні **inline**-функції на відміну від звичайної, полягає: у скорочення часу виконання програми. Програш теж очевидний – зростає об'єм програми. При порівнянні із макровизначеннями – виграш очевидний і безперечний – зникають всі проблеми з параметрами, вони передаються звичайним чином – за значенням.

Втім, слід зазначити, що службове слово **inline** – це лише прохання до компілятора, яке він може проігнорувати, якщо вважатиме, що затрати на виклик функції менші за її включення в код. Безперечно, не буде вбудованою рекурсивна функція.

Приклад.

```
inline double cube (double x) {return x*x*x;}
int main ()
{
  double res, s = 9;
  res = cube (++s);  // вірно: s = 10, res = 1000
}
```

Змінні-посилання. Зазвичай унарна операція **&** означає взяття адреси об'єкта. Вона ж може бути використана при визначенні так званих ***змінних-посилань***.

Приклад.

```
int val = 10; // Створили цілу змінну
// Створили змінну-посилання на val
int & r_val = val;
```

Тепер **val** та **r_val** синоніми, проте різних типів. Вони мають одне й те саме значення і одну й ту саму адресу в пам’яті.

Зверніть увагу, проініціалізувати змінну-посилання можна лише в момент визначення:

```
int val = 10;
int & r_val;
r_val = val; // помилка!
```

Змінні-посилання як аргументів функцій.

Як відомо, класична мова С дозволяє передачу параметрів у функції **лише за значенням**. В разі необхідності зміни значень параметрів функцій потрібні вказівники на них. Тепер же змінні-посилання означають передачу **by-reference** (за посиланням).

Приклад.

```
// Функція переставляє свої параметри
void change (int &a, int &b)
//В C# : void change (ref int a, ref int b)
{
 int temp = a; a = b; b = temp;
}
int main ()
{
 int x = 10, y = 20;
 change (x, y);
 cout << x << y; // x і y реально переставлені!
}
```

На відміну від передачі параметрів за значенням при такому способі в стеку не створюється копія відповідного аргументу, а передається змінна-посилання, яка є псевдонімом цього аргументу.

Змінні-посилання можуть використовуватись і як результат функції.

Відмінності у різних способах передачі параметрів функцій

1) Передача параметрів за значенням (умовчання)

```
void change (int a, int b)
{
    int temp = a; a = b; b = temp;
}
int main ()
{
    int x = 10, y = 20;
    // передача значень змінних
    change (x, y);
    // x та y не змінились!
    cout << x << y;
}
```

Відмінності у різних способах передачі параметрів функцій

2) Передача посилань на параметри (C++)

```
void change (int &a, int &b)
{
    int temp = a; a = b; b = temp;
}
int main ()
{
    int x = 10, y = 20;
    // передача посилань на змінні
    change (x, y);
    // x та y змінились!
    cout << x << y;
}
```

Відмінності у різних способах передачі параметрів функцій

3) Передача вказівників на параметри (C та C++)

```
void change (int *pa, int *pb)
{
    int temp = *pa; *pa = *pb; *pb = temp;
}
int main ()
{
    int x = 10, y = 20;
    // передача вказівників змінних
    change (&x, &y);
    // x та y не змінились!
    cout << x << y;
}
```

Коли має сенс використовувати змінні-посилання:

- якщо необхідно, щоб функція змінювала дані, які передає їй функція, що її викликає;
- якщо треба підвищити швидкодію програми – за рахунок передачі функції посилань замість об'єктів даних.

Функція <u>не змінює</u> передані їй у виклику аргументи у таких випадках:

- якщо розміри об'єкту даних невеликі, наприклад це вбудований тип даних або невелика структура – варто передавати його за значенням;
- якщо об'єктом даних є масив – використовуйте вказівник, проте для захисту даних задекларуйте його як вказівник на **`const`**;
- якщо об'єктом даних є велика структура – використовуйте вказівник на **`const`** або **`const`**-посилання для економії часу та пам'яті (об'єкт не копіюватиметься у стек);
- якщо об'єктом даних є екземпляр класу – використовуйте **`const`**-посилання для економії часу та пам'яті .

Функція <u>змінює</u> передані їй у виклику аргументи у таких випадках:

- якщо об'єктом даних є вбудований тип – використовуйте вказівник;
- якщо об'єктом даних є масив – єдиним правильним варіантом є вказівник;
- якщо об'єктом даних є структура – використовуйте вказівник або посилання;
- якщо об'єктом даних є екземпляр класу – використовуйте посилання.

Перевантаження функцій – простий поліморфізм.

Віднині функції можуть мати однакові ідентифікатори при умові, що вони розрізняються ***сигнатурами***.

До сигнатури входять ідентифікатор функції та список її параметрів.

Приклад.

```
void fun (int i);
int fun (int i);  // не перевантажується!
void fun (int i, int j);
void fun (double x);
void fun (int *pi);
void fun (int &i);  // не перевантажується!
void fun (char *s);
void fun (const char *s);
```

Параметри функцій за умовчанням.

Мається на увазі можливість задавати значення параметрам функцій, таким чином при виклику відповідні аргументи, якщо вони пропущені, будуть замінені значеннями за умовчанням.

Приклад.

```
// Функція ініціалізації текстового режиму на екрані
void InitScreen (int width = 80,
                 int height = 24, char background =‘ ’)
{
// код функції
}
int main ()
{
// Можливі виклики:
 InitScreen (60);
 InitScreen (60, 20);
 InitScreen (60, 20, ‘#’);
// Неможливий виклик:
 InitScreen (‘*’);
}
```