Функції. Формальні та фактичні параметри.

1. Що таке функція? Визначення функції. Переваги використання функцій

При написанні програм середнього та високого рівня складності виникає потреба в їх розбитті на частини. Розбиття великої програми на частини дозволяє зменшити ризик виникнення помилок, підвищує читабельність програмного коду завдяки його структуруванню.

Крім того, якщо деякий програмний код повторюється декілька разів (або є близьким за змістом), то є доцільним організувати його у вигляді функції, яку потім можна викликати багатократно за її іменем. Таким чином, відбувається економія пам'яті, зменшення вихідного коду програми, тощо.

Функція – це частина програми, яка має такі властивості чи ознаки:

- є логічно самостійною частиною програми;
- має ім'я, на основі якого здійснюється виклик функції (виконання функції). Ім'я функції підпорядковується правилам задавання **імен ідентифікаторів мови С++**;
- може містити список параметрів, які передаються їй для обробки або використання. Якщо функція не містить списку параметрів, то така функція називається функцією без параметрів;
- може повертати (не обов'язково) деяке значення. У випадку, якщо функція не повертає ніякого значення, тоді вказується ключове слово void;
- має власний програмний код, який береться у фігурні дужки { } і вирішує задачу, яка поставлена на цю функцію. Програмний код функції, реалізований в фігурних дужках, називається "тіло функції".

Використання функцій у програмах дає такі переваги:

- компактна організація програми шляхом зручного виклику програмного коду за його іменем, який у програмі може зустрічатись декілька разів (повторюватись);
- економія пам'яті, розміру вихідного та виконавчого коду і т.д.;
- зменшення ризику виникнення помилок для великих наборів кодів;
- підвищення читабельності програмного коду.

2. Яка загальна форма опису функції?

Загальна форма опису функції виглядає наступним чином:

```
тип ім'я_функції(список_параметрів або void)
{
   тіло_функції
   [return] (вираз);
```

}
де

- ***тип*** – тип значення, яке повертає функція. Якщо поле “*тип*” містить ключове слово void, то функція не повертає ніякого значення;
- ***ім’я_функції*** – це безпосередньо ім’я функції. За цим іменем відбувається виклик програмного коду, реалізованого в *тілі_функції*. Крім того, *ім’я_функції* є **покажчиком на цю функцію**. Значенням покажчика є адреса точки входу в функцію;
- ***список_параметрів*** – параметри, які передаються в функцію. Функція може отримувати будь-яку кількість параметрів. Якщо описується функція без параметрів, то в дужках вказується слово void;
- ***тіло_функції*** – набір операторів програмного коду, що реалізують алгоритм обчислення всередині функції;
- **return (*вираз*)** – ключове слово return вказує, що функція повертає значення задане в (вираз). Слово return може зустрічатись в декількох місцях тіла функції залежно від алгоритму (повторюватись).

3. Приклади опису та використання функцій, що не повертають значення

Якщо функція не повертає значення, тоді вона повинна починатися з ключового слова void.
Приклад 1. Функція MyFunc1() без параметрів, яка не повертає значення. Якщо в тілі деякого класу або модуля описати функцію:

```
// опис функції, яка не отримує і не повертає параметрів
void MyFunc1(void)
{
   // тіло функції - вивід тексту на форму в компонент label1
   label1->Text = "MyFunc1() is called";
   return; // повернення з функції
}
```

тоді викликати цю функцію можна наступним чином:

```
...

// виклик функції з іншого програмного коду (наприклад, обробника події)
MyFunc1();

...
```

4. Які є способи передачі параметрів у функцію? Приклад

В C++ існує **3 способи** передачі параметрів у функцію:

- передача параметру **за значенням** (Call-By-Value). Це є проста передача копій змінних в функцію. У цьому випадку зміна значень параметрів в тілі функції не змінить значення, що передавались у функцію ззовні (при її виклику);
- передача параметру **за адресою** змінної. У цьому випадку функції в якості параметрів передаються не копії змінних, а копії адрес змінних, тобто покажчик на змінну. Використовуючи цей покажчик функція здійснює доступ до потрібних комірок пам'яті де розташована передана змінна і може змінювати її значення. Загальні відомості про покажчики наведено **тут**;
- передача параметру **за посиланням** (Call-By-Reference). Передається посилання (покажчик) на об'єкт (змінну), що дозволяє синтаксично використовувати це посилання як покажчик і як значення. Зміни, внесені в параметр, що переданий за посиланням, змінюють вихідну копію параметра викликаючої функції.

Приклад. Цей приклад демонструє відмінність між передачею параметрів за значенням, передачею параметрів за адресою та передачею параметрів за посиланням. Описується функція, що отримує три параметри. Перший параметр (x) передається за значенням. Другий параметр (y) передається за адресою (як покажчик). Третій параметр (z) передається за посиланням.

```
// функція MyFunction
// параметр x - передається за значенням (параметр-значення)
// параметр y - передається за адресою
// параметр z - передається за посиланням
void MyFunction(int x, int* y, int& c)
{
   x = 8; // значення параметра змінюється тільки в межах тіла функції
   *y = 8; // значення параметра змінюється також за межами функції
   c = 8; // значення параметра змінюється також за межами функції
   return;
}
```

Демонстрація виклику функції MyFunction() з іншого програмного коду:

```
int a, b, c;

a = b = c = 5;

// виклик функції MyFunction()
// параметр a передається за значенням a->x
// параметр b передається за адресою   b->y
// параметр c передається за посиланням c->z
MyFunction(a, &b, c); // на виході a = 5; b = 8; c = 8;
```

Як видно з результату, значення змінної a не змінилось. Тому що, змінна a передавалась у функцію MyFunction() з передачею значення (перший параметр).

Однак, значення змінної b після виклику функції MyFunction() змінилось. Це пов'язано з тим, що в функцію MyFunction() передавалось значення адреси змінної b. Маючи адресу змінної b в пам'яті комп'ютера, всередині функції MyFunction() можна змінювати значення цієї змінної з допомогою покажчика y.
Також змінилось значення **c** після виклику функції. Посилання є адресою об'єкту в пам'яті. З допомогою цієї адреси можна мати доступ до значення об'єкта.

5. Що таке формальні та фактичні параметри функції? Приклад

Формальні параметри – це змінні, що приймають значення аргументів (параметрів) функції. Якщо функція має декілька аргументів (параметрів), їх тип та імена розділяються комою ' , '.
При виклику функції, що має аргументи, компілятор здійснює копіювання копій формальних аргументів в стек.

Приклад 1. Функція MyAbs(), що знаходить модуль числа має один формальний параметр x.

```
// функція, що знаходить модуль дійсного числа
float MyAbs(float x) // x - формальний параметр
{
   if (x<0) return (float)(-x);
   else return x;
}
```

Виклик функції з іншого програмного коду (іншої функції)

```
// виклик функції з іншого програмного коду
float res, a;

a = -18.25f;
res = MyAbs(a); // res = 18.25f; змінна a - фактичний параметр
res = MyAbs(-23); // res = 23; константа 23 - фактичний параметр
```

При виклику функції з іншого програмного коду фігурує фактичний параметр. У даному прикладі фактичний параметр це змінна a та константа 23.
При виклику функції фактичні параметри копіюються в спеціальні комірки пам'яті в стеку (стек – частина пам'яті). Ці комірки пам'яті відведені для формальних параметрів. Таким чином, формальні параметри (через використання стеку) отримують значення фактичних параметрів.

Оскільки, фактичні параметри копіюються в стек, то зміна значень формальних параметрів в тілі функції не змінить значень фактичних параметрів (тому що це є копія фактичних параметрів).

6. Яка область видимості формальних параметрів функції? Приклад

Область видимості формальних параметрів функції визначається межами тіла функції, в якій вони описані. У наведеному нижче прикладі формальний параметр n цілого типу має область видимості в межах фігурних дужок { }.
Приклад. Функція, що знаходить факторіал цілого числа n.

```
// функція, що знаходить n!
unsigned long int MyFact(int n) // початок області видимості формального параметру n
{
   int i;
   unsigned long int f = 1; // результат

   for (i=1; i<=n; i++)
      f = f*i;

   return f;   // кінець області видимості формального параметру n
}
```

Виклик функції з іншого програмного коду (іншої функції):

```
// виклик функції з іншого програмного коду
int k;
unsigned long int fact;
k = 6;
fact = MyFact(k); // fact = 6! = 720
```

⇑

7. Що таке прототип функції? Приклади

Прототип функції – це повідомлення компілятору та іншим функціям про те, що така функція існує. Іншими словами, це повідомлення компілятору та іншим функція про те, що у програмі існує функція з заданою сигнатурою. Прототип функції складається з скороченого опису без тіла функції. Прототип функції містить:

- ім’я функції;
- параметри функції або типи цих параметрів;
- тип значення, яке повертається функцією.

Загальна форма оголошення прототипу функції наступна:

```
return_type FuncName(parameters);
```

де

- *return_type* – тип, що повертається функцією;
- FuncName – ім’я функції;
- *parameters* – параметри функції.

При вказанні прототипу функції необов’язково вказувати імена параметрів. Наприклад, для функції

```
double Average(int A[], int size);
```

можна задати наступний прототип

```
double Average(int[], int);
```

У випадках, коли функція, яку викликають, оголошена перед викликаючою її функцією можна обійтись без використання прототипу. В інших випадках прототип функції необхідний.

Якщо функція описується в класі і викликається з методів цього класу, тоді подібних помилок не буде. Це пов’язане з тим, що в класі прототип функції відомий усім методам класу.