

Лекція 7.

Функції. Область видимості. Рекурсія.



План на сьогодні

1

Що таке функція?

2

Синтаксис функцій

3

Перевантаження функцій

4

Область видимості функцій

5

Рекурсивні функції

6

Функція `main()`



FACULTY OF APPLIED
MATHEMATICS AND
INFORMATICS
LVIV UNIVERSITY

Що таке функція?



FACULTY OF APPLIED
MATHEMATICS AND
INFORMATICS
LVIV UNIVERSITY

Що таке функції?

Функція в C++ - це іменований блок коду, який виконується лише тоді, коли він викликається.

У функцію можна передавати дані, відомі як параметри.

Є важливими для повторного використання коду: визначте код один раз і використовуйте його багато разів.

У C++ є два типи функцій: стандартної бібліотеки C++ та визначені користувачем

Оголошення функції

`void` - означає, що функція не повертає значення

`myFunction` - назва функції

`()` - параметри функції

```
void myFunction() {  
    // code to be executed  
}
```

Тіло функції: код, який буде виконуватись при виклику функції

Оголошена функція

```
// Create a function
```

```
void myFunction() {  
    cout << "I just got executed!";  
}
```

```
int main() {
```

```
    myFunction(); // call the function
```

```
    return 0;
```

```
}
```

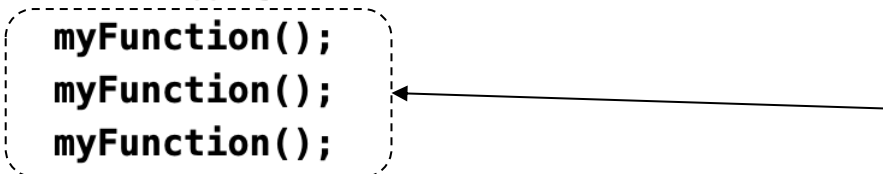
```
// Outputs "I just got executed!"
```

```
I just got executed!
```

Виклик функції

```
void myFunction() {  
    cout << "I just got executed!\n";  
}
```

```
int main() {  
    myFunction();  
    myFunction();  
    myFunction();  
    return 0;  
}
```



```
I just got executed!  
I just got executed!  
I just got executed!
```

Функції можна викликати багато разів

```
int main() {  
    myFunction();  
    return 0;  
}  
  
void myFunction() {  
    cout << "I just got executed!";  
}  
  
// Error
```

1. 5:3: error: 'myFunction' was not declared in this scope

В C++ якщо функція оголошена після функції main() - станеться помилка


```
// Function declaration  
void myFunction();
```

Прототип функції

```
// The main method  
int main() {  
    myFunction(); // call the function  
    return 0;  
}
```

```
// Function definition  
void myFunction() {  
    cout << "I just got executed!";  
}
```

Тіло функції

Оголошення функції і тіло функції можуть бути написаними окремо

Параметри та аргументи функції



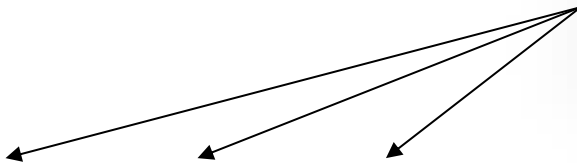
Параметри функції

Інформацію можна передавати у функції як параметри. Параметри виступають як змінні всередині функції.

Параметри задаються після назви функції, у круглих дужках. Потрібно задати типи та назви параметрів. Ви можете додати стільки параметрів, скільки потрібно, просто розділіть їх комами

Параметри функції

Параметри функції



```
void functionName(parameter1, parameter2, parameter3) {  
    // code to be executed  
}
```

Параметри функції

Формальні параметри функції

```
void myFunction(string(fname)) {  
    cout << fname << " Refsnes\n";  
}
```

```
int main() {  
    myFunction("Liam");  
    myFunction("Jenny");  
    myFunction("Anja");  
    return 0;  
}
```

Фактичні параметри функції
або аргументи

Liam Refsnes
Jenny Refsnes
Anja Refsnes

Коли параметр передається у функцію, він називається фактичним параметром або аргументом. Отже, з наведеного вище прикладу: **fname** є формальним параметром, а **Liam**, **Jenny** та **Anja** — це аргументи або фактичні параметри..

Параметри за замовчуванням

Можна також використовувати значення параметра за замовчуванням, використовуючи знак рівності (=).

```
void myFunction(string country = "Norway")
{
    cout << country << "\n";
}

int main() {
    myFunction("Sweden");
    myFunction("India");
    myFunction();
    myFunction("USA");
    return 0;
}

// Sweden
// India
// Norway
// USA
```



Sweden
India
Norway
USA

Кількість параметрів

```
void myFunction(string fname, int age) {  
    cout << fname << " Refsnes. " << age << " years old. \n";  
}  
  
int main() {  
    myFunction("Liam", 3);  
    myFunction("Jenny", 14);  
    myFunction("Anja", 30);  
    return 0;  
}
```

Параметрів може бути необмежена кількість

Значення, які повертають функції



Приклад повернення значення

```
int myFunction(int x) {  
    return 5 + x;  
}  
  
int main() {  
    cout << myFunction(3);  
    return 0;  
}  
  
// Outputs 8 (5 + 3)
```



8

Ключове слово **void**, використане в попередніх прикладах, вказує на те, що функція не повинна повертати значення. Якщо ви хочете, щоб функція повертала значення, замість **void** ви можете використовувати тип даних (такий як **int**, **string** тощо) і всередині функції використовувати ключове слово **return**

Приклад повернення значення

```
int addNumbers(int x, int y) {  
    return x + y;  
}  
  
int main() {  
    cout << addNumbers(5, 3);  
    return 0;  
}  
  
// Outputs 8 (5 + 3)
```



8

Приклад повернення значення

```
int myFunction(int x, int y) {  
    return x + y;  
}  
  
int main() {  
    int z = myFunction(5, 3);  
    cout << z;  
    return 0;  
}
```

8

Значення можна також зберегти в змінну

Приклад повернення вказівника

```
1  #include <iostream>
2  using namespace std;
3  int* createArray(int size) {
4      int* arr = new int[size]; // Dynamically allocate memory for an array
5      for (int i = 0; i < size; ++i) {
6          arr[i] = i * 10; // Initialize array elements
7      }
8      return arr; // Return the pointer to the array
9  }
10 void printArray(int *arr, int size){
11     for (int i = 0; i < size; ++i) {
12         std::cout << arr[i] << " ";
13     }
14     std::cout << std::endl;
15 }
16 int main() {
17     int size = 10;
18     int* myArray = createArray(size); // Function returns a pointer to the array
19     printArray(myArray, size);
20
21     delete[] myArray; // Don't forget to free the allocated memory
22     return 0;
23 }
```

Приклад повернення rvalue

Функція повертає тимчасовий об'єкт (**rvalue**), адреса якого копіюється у константне посилання на **string**, константне посилання продовжує життя тимчасового об'єкта. Слід зауважити що у даному випадку при поверненні з функції відбувається лише копіювання адреси а не цілого об'єкта **string** у змінну "greeting".

```
1  #include <iostream>
2  using namespace std;
3
4  std::string getGreeting() {
5      return "This is C++";
6  }
7
8  int main() {
9      const string& greeting = getGreeting();
10     cout << greeting << endl;
11     return 0;
12 }
```

Параметри-значення (Pass by Value)



Передача параметрів за значенням

У функцію передається локальна копія фактичного аргумента. При зміні цього параметру всередині функції, фактичний параметр у точці виклику функції не зміниться.

```
1  #include <iostream>
2  using namespace std;
3
4  void fun(int x) {
5      x = 30;
6  }
7
8  int main() {
9      int x = 20;
10     fun(x);
11     cout << "x = " << x;
12     return 0;
13 }
```

Формальний параметр

Фактичний параметр

Передача параметрів за значенням

Зверніть увагу, що у випадку передачі параметрів по значенню відбувається копія фактичних параметрів у формальні параметри функції, що є затратною операцією у випадку типу **string**.

```
1  #include <iostream>
2  using namespace std;
3
4  void printString(std::string str) {
5      std::cout << str << std::endl;
6  }
7
8  int main() {
9      std::string myString = "Hello C++ Functions!";
10     printString(myString);
11     return 0;
12 }
```

Формальний параметр

Фактичний параметр

Параметри-посилання (Pass by Reference)

Передача параметрів за адресою

У попередніх прикладах використовувались звичайні змінні, коли передавались параметри у функцію. Можна також передавати адресу фактичного параметру, тобто функція працює з фактичним аргументом безпосередньо. Це може бути корисно, коли потрібно змінити значення аргументів.

Посилання на фактичні параметри

```
void swap(int& a, int& b) {  
    a ^= b;  
    b ^= a;  
    a ^= b;  
}
```

`void function(const int &x)` – константне посилання – це обіцянка не змінювати об'єкт

Приклад 1

```
1  #include <iostream>
2  using namespace std;
3
4  void swap(int& a, int& b) {
5      a ^= b;
6      b ^= a;
7      a ^= b;
8  }
9
10 void main() {
11     int first = 1, second = 2;
12     cout << "Before swap: \n";
13     cout << first << '\t' << second << endl;
14     swap(first, second);
15     cout << "After swap: \n";
16     cout << first << '\t' << second << endl;
17 }
```

Before swap:

1 2

After swap:

2 1

Приклад 2

Якщо значення фактичного параметра не повинна змінювати функція, то його можна передавати через константне посилання. В такому випадку відбувається копіювання адреси фактичного параметра у формальний параметр функції, що є більш оптимальним ніж копіювання цілого об'єкта у випадку передачі за значенням.

```
1  #include <iostream>
2  using namespace std;
3
4  void printString(const std::string& str) { // Note the 'const' to prevent modification
5      std::cout << str << std::endl;
6  }
7
8  int main() {
9      std::string myString = "Hello C++ Functions!";
10     printString(myString);
11     return 0;
12 }
```

Приклад 3

Літерали, константи і аргументи, що вимагають перетворення типів, можна передавати як **const&-** аргументи і не можна – як не **const&-** аргументи

```
1  #include <iostream>
2  using namespace std;
3
4  void printString( std::string& str) { // Note the 'const' to prevent modification
5      std::cout << str << std::endl;
6  }
7
8  int main() {
9      std::string myString = "Hello C++ Functions!";
10     printString(myString); // OK
11     printString("Test");   // Compilation error
12     return 0;
13 }
```

Передача масивів у функції



FACULTY OF APPLIED
MATHEMATICS AND
INFORMATICS
LVIV UNIVERSITY

Одновимірні масиви

Оскільки `T[]` перетворюється до `T*`, то масиви не можуть передаватися у функцію як значення. Традиційно передається адреса початку масиву і кількість його елементів

```
4 void printArray(const int arr[], int size)
5 {
6     for (int i = 0; i < size; i++) {
7         cout << arr[i] << " ";
8     }
9     cout << endl;
10 }
11
12 int sum(const int* arr, int size)
13 {
14     int s = 0;
15     for (int i = 0; i < size; i++) {
16         s += arr[i];
17     }
18     return s;
19 }
20
21 int main()
22 {
23     int arr[] = { 10, 7, 8, 9, 1, 5 };
24     int size = sizeof(arr) / sizeof(arr[0]);
25     printArray( arr, size);
26     cout << "Array Sum=" << sum(arr, size);
27 }
```

```
10 7 8 9 1 5
Array Sum=40
```

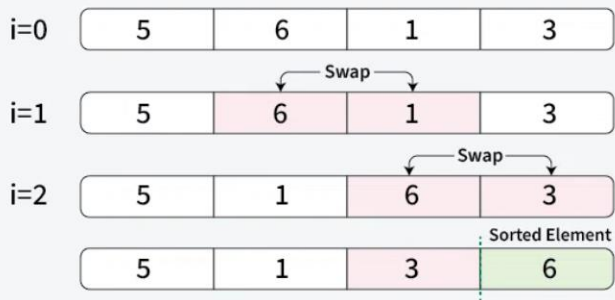
Двовимірні масиви

```
// Memory allocation
int** new_array(size_t rows, size_t columns) {
    int** arr = new int* [rows];
    for (size_t i = 0; i < rows; ++i)
        arr[i] = new int[columns];
    return arr;
}

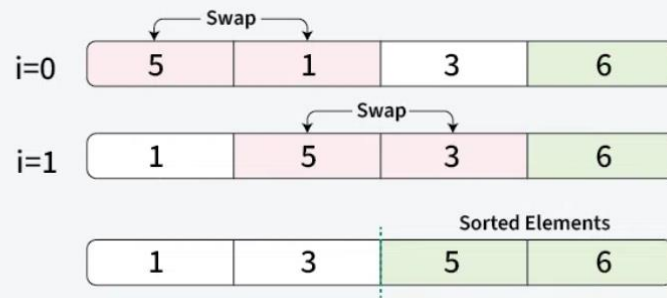
// Random initialization with number from 0 to 100
void random_array(int** arr, size_t rows, size_t columns) {
    srand(time(0)); // Use current time as seed for random generator
    const int RANGE_MAX = 100;
    for (size_t i = 0; i < rows; ++i)
        for (size_t j = 0; j < columns; j++)
            arr[i][j] = rand() % RANGE_MAX;
}
```


Сортування бульбашкою

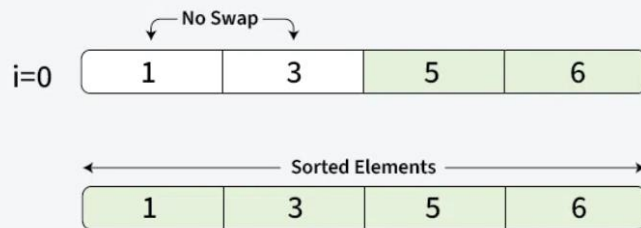
01 Step | Placing the 1st largest element at its correct position



02 Step | Placing 2nd largest element at its correct position



03 Step | Placing 3rd largest element at its correct position



Сортування бульбашкою

```
9 // An optimized version of Bubble Sort
10 void bubbleSort(int* arr, int size) {
11     bool swapped;
12     for (int i = 0; i < size - 1; i++) {
13         swapped = false;
14         for (int j = 0; j < size - i - 1; j++) {
15             if (arr[j] > arr[j + 1]) {
16                 swap(arr[j], arr[j + 1]);
17                 swapped = true;
18             }
19         }
20         // If no two elements were swapped, then break
21         if (!swapped)
22             break;
23     }
24 }
```

Вказівник на функцію



FACULTY OF APPLIED
MATHEMATICS AND
INFORMATICS
LVIV UNIVERSITY

Функція зворотного виклику

Функція зворотного виклику — це функція, яка передається як аргумент іншій функції

```
1  #include <iostream>
2  using namespace std;
3
4  // Define a callback function type
5  typedef void (*CallbackFunction)();
6
7  // Function that takes a callback function as an argument
8  void performAction(CallbackFunction callback) {
9      std::cout << "Performing some action...\n";
10     // Call the callback function
11     callback();
12 }
13
14 // Example callback function
15 void myCallback() {
16     std::cout << "Callback function \n";
17 }
18
19 int main() {
20     // Pass the callback function to performAction
21     performAction(myCallback);
22     return 0;
23 }
```

Масив вказівників функцій

```
4 int add(int a, int b) {
5     return a + b;
6 }
7 int subtract(int a, int b) {
8     return a - b;
9 }
10 int multiply(int a, int b) {
11     return a * b;
12 }
13
14 int main() {
15     // Declare and initialize an array of function pointers
16     int (*funcArray[3])(int, int) = { add, subtract, multiply };
17
18     // Variables to use as function parameters
19     int x = 2, y = 3;
20
21     // Access and call the functions using the array of function pointers
22     std::cout << "Add: " << funcArray[0](x, y) << std::endl;    // Calls add(10, 5)
23     std::cout << "Subtract: " << funcArray[1](x, y) << std::endl; // Calls subtract(10, 5)
24     std::cout << "Multiply: " << funcArray[2](x, y) << std::endl; // Calls multiply(10, 5)
25
26     return 0;
27 }
```

Перевантаження функцій



FACULTY OF APPLIED
MATHEMATICS AND
INFORMATICS
LVIV UNIVERSITY

Перевантаження функцій

За допомогою перевантаження функцій, кілька функцій можуть мати однакову назву, але з різними параметрами. Це означає, що функції з однаковою назвою можуть виконувати різні завдання в залежності від кількості або типу параметрів, які їм передаються.

Перевантажені функції

- ❖ знаходяться в одній області видимості,
- ❖ мають однакову назву,
- ❖ **повинні відрізнятись аргументами** (типом і кількістю),
- ❖ не можуть відрізнятись **лише типом повернення**.

```
int myFunction(int x)
float myFunction(float x)
double myFunction(double x, double y)
```

Перевантажена функції

Приклад без перевантаження

```
int plusFuncInt(int x, int y) {  
    return x + y;  
}  
  
double plusFuncDouble(double x, double y) {  
    return x + y;  
}  
  
int main() {  
    int myNum1 = plusFuncInt(8, 5);  
    double myNum2 = plusFuncDouble(4.3, 6.26);  
    cout << "Int: " << myNum1 << "\n";  
    cout << "Double: " << myNum2;  
    return 0;  
}
```

Int: 13

Double: 10.56

Приклад з перевантаженням

```
int plusFunc(int x, int y) {  
    return x + y;  
}  
  
double plusFunc(double x, double y) {  
    return x + y;  
}  
  
int main() {  
    int myNum1 = plusFunc(8, 5);  
    double myNum2 = plusFunc(4.3, 6.26);  
    cout << "Int: " << myNum1 << "\n";  
    cout << "Double: " << myNum2;  
    return 0;  
}
```

```
Int: 13  
Double: 10.56
```

Особливості перевантаження з const

```
1 #include <iostream>
2 using namespace std;
3
4 void fun(const int i)
5 {
6     cout << "fun(const int) called ";
7 }
8 void fun(int i)
9 {
10     cout << "fun(int ) called ";
11 }
12 int main()
13 {
14     const int i = 10;
15     fun(i);
16     return 0;
17 }
```

Entire Solution



2 Errors



0 Warnings



0 Messages

	Code	Description
✗	C2084	function 'void fun(const int)' already has a body
✗	C2065	'fun': undeclared identifier

```
1 #include <iostream>
2 using namespace std;
3
4 void fun(char* a){
5     cout << "non-const fun() " << a;
6 }
7 void fun(const char* a){
8     cout << "const fun() " << a;
9 }
10
11 int main(){
12     const char* ptr = "Hello C++ function";
13     fun(ptr);
14     return 0;
15 }
```

```
1 #include <iostream>
2 using namespace std;
3
4 void fun(const int& i)
5 {
6     cout << "fun(const int &) called ";
7 }
8 void fun(int& i)
9 {
10     cout << "fun(int &) called ";
11 }
12 int main()
13 {
14     const int i = 10;
15     fun(i);
16     return 0;
17 }
```

Помилки перевантаження

```
1  #include <iostream>
2  using namespace std;
3
4  int fun() { return 10; }
5  char fun() { return 'a'; }
6  // compiler error as it is a new declaration of fun()
7
8  int main()
9  {
10     char x = fun();
11     getchar();
12     return 0;
13 }
```

int fun(int *ptr);
int fun(int ptr[]); // redeclaration of fun(int *ptr)

void h(int ());
void h(int (*)()); // redeclaration of h(int())

Entire Solution

5 Errors

0 Warnings

0 Messages

7

Build + Inte

	Code	Description
abc	E0311	cannot overload functions distinguished by return type alone
	C2556	'char fun(void)': overloaded function differs only by return type from 'int fun(void)'
	C2371	'fun': redefinition; different basic types
	C2065	'fun': undeclared identifier
	C2440	'initializing': cannot convert from 'int (__cdecl *)(void)' to 'char'

Область видимості



FACULTY OF APPLIED
MATHEMATICS AND
INFORMATICS
LVIV UNIVERSITY

Область видимості

```
void myFunction() {  
    // Local variable that belongs to myFunction  
    int x = 5;  
  
    // Print the variable x  
    cout << x;  
}  
  
int main() {  
    myFunction();  
    return 0;  
}
```

Область видимості

5

Приклад з помилкою

```
void myFunction() {  
    // Local variable that belongs to myFunction  
    int x = 5;  
}
```

Область видимості

```
int main() {  
    myFunction();  
  
    // Print the variable x in the main function  
    cout << x;  
    return 0;  
}
```

За межами області
видимості

Приклад зі статичною змінною

static - об'єкт розміщується в області даних, значення зберігається при виході з блоку; ініціалізація явно, інакше неявно – нулем; глобальний час дії

```
int add(int myNumber) {  
    static int total = 0;  
    total += myNumber;  
    return total;  
}  
  
int main() {  
    cout << add(5) << "\\n";  
    cout << add(2) << "\\n";  
    cout << add(4) << "\\n";  
    cout << add(9) << "\\n";  
    return 0;  
}
```



```
5  
7  
11  
20
```

Глобальна область видимості

```
// Global variable x
```

```
int x = 5;
```

```
void myFunction() {  
    // We can use x here  
    cout << x << "\n";  
}
```

```
int main() {  
    myFunction();  
  
    // We can also use x here  
    cout << x;  
    return 0;  
}
```

Область видимості



5

5

Рекурсивні функції

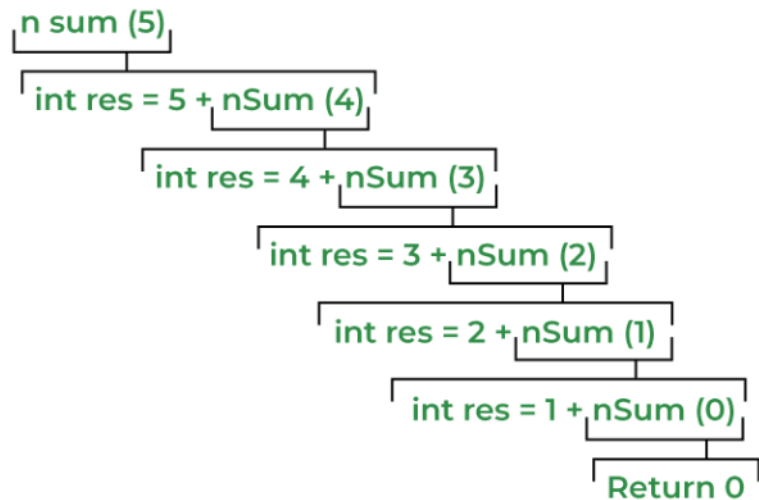
Рекурсія

Рекурсія — це техніка, коли функція викликає саму себе. Ця техніка дає можливість розбивати складні проблеми на простіші задачі, які легше вирішити.

Приклад 1

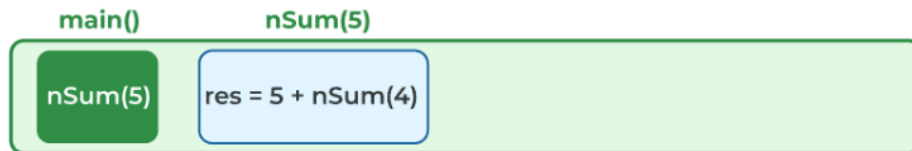
Рекурсія

```
1 #include <iostream>
2 using namespace std;
3
4 int nSum(int n)
5 {
6     // base condition to terminate the recursion when N = 0
7     if (n == 0) {
8         return 0;
9     }
10    // recursive case // recursive call
11    int res = n + nSum(n - 1);
12
13    return res;
14 }
15
16 int main()
17 {
18     int n = 5;
19     int sum = nSum(n);
20     cout << "Sum = " << sum;
21
22     return 0;
23 }
```

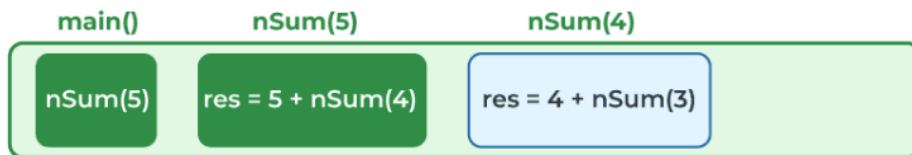


Наповнення стеку викликів

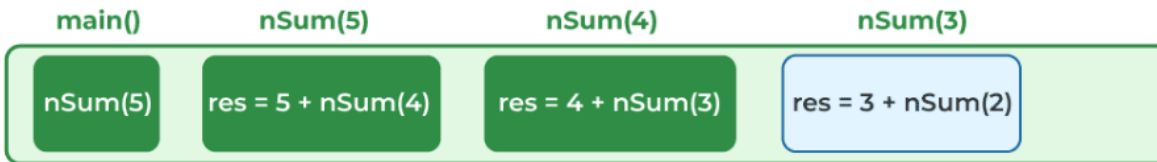
Крок 1



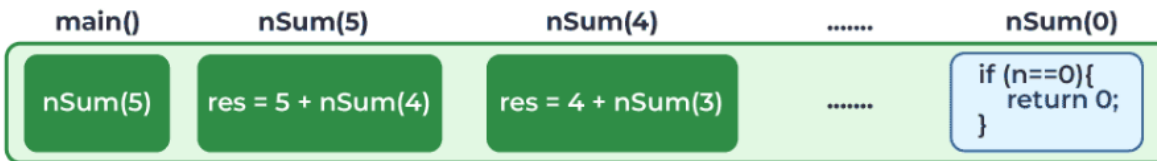
Крок 2



Крок 3

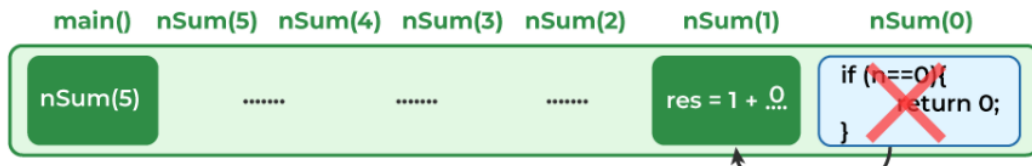


Крок 5



Зворотній хід викликів

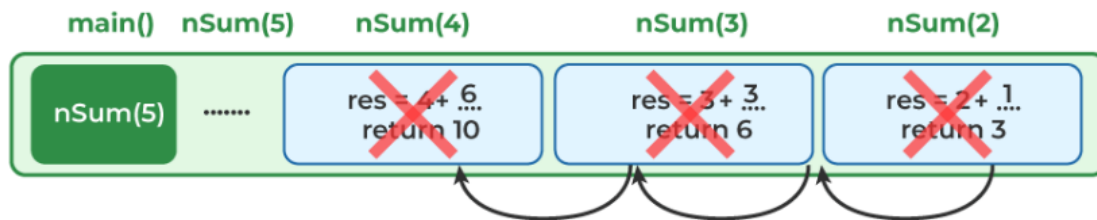
Крок 6



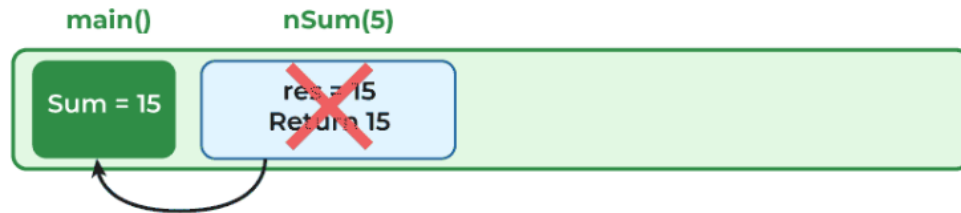
Крок 7



Крок 8



Крок 9



Output = 15

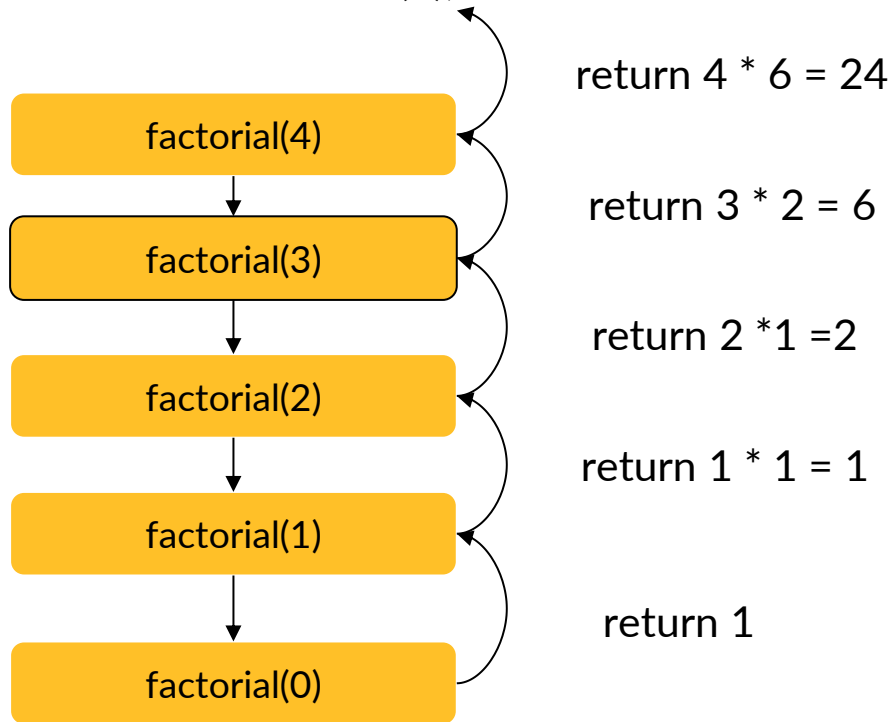
Приклад 2 (Факторіал)

```
int fact(int n)
{
    if (n == 0) {
        return 1;
    }

    return n * fact(n - 1);
}
```

$$4! = 1 * 2 * 3 * 4 = 120$$

int a = factorial(4); // 24

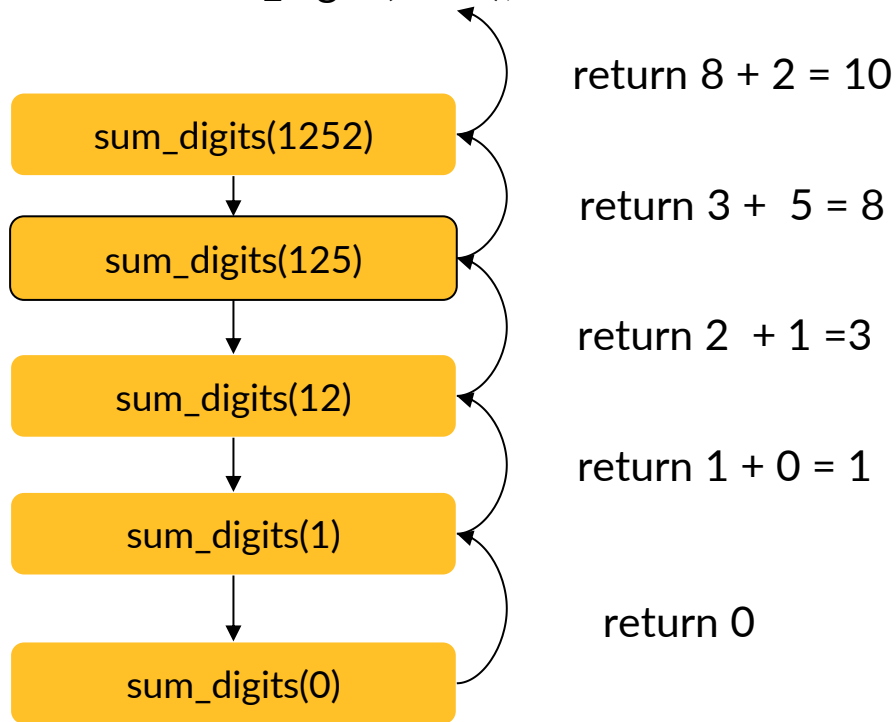


Приклад 2 (Сума цифр числа)

int a = sum_digits(1252); // 10

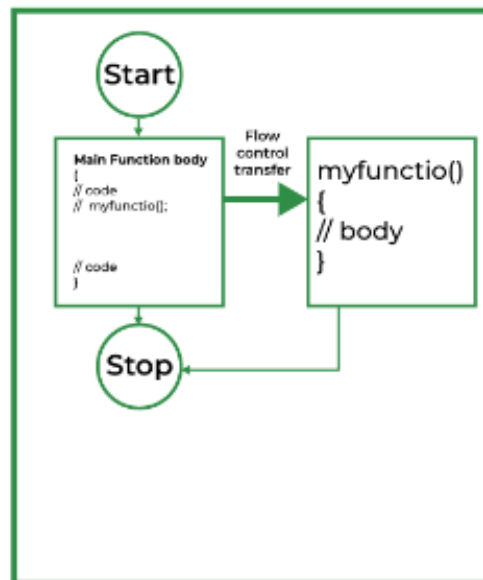
```
int sum_digits(int n)
{
    if (n == 0) {
        return 0;
    }
    return n % 10 + sum_digits(n / 10);
}
```

$$1 + 2 + 5 + 2 = 10$$

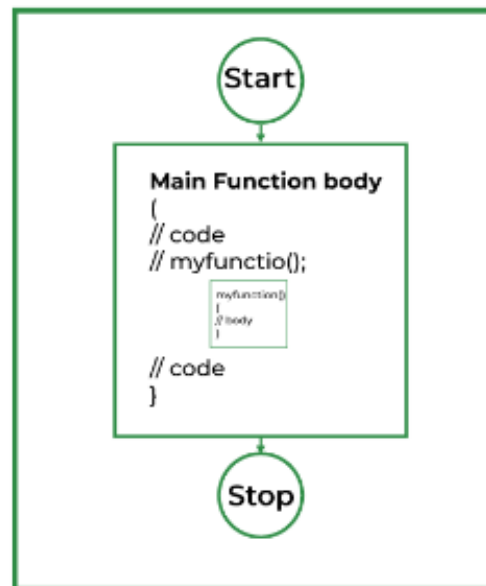


inline функції

Normal Function



Inline Function



```
1  #include <iostream>
2  using namespace std;
3
4  inline int cube(int s) {
5      return s * s * s;
6  }
7  int main() {
8      cout << "The cube of 3 is: " << cube(3) << "\n";
9      return 0;
10 }
```


inline функції

Компілятор може не виконувати **inline** за таких обставин, як:

- ❖ Якщо функція містить цикл.
- ❖ Якщо функція містить статичні змінні.
- ❖ Якщо функція рекурсивна.
- ❖ Якщо тип повернення функції інший, ніж void, і оператор return не існує в тілі функції.
- ❖ Якщо функція містить оператор switch або goto.

Функція `main()`

Функція main()

```
int main(int argc, char *argv[]) { /* ... */ }
```

- ❖ **argc (ARGument Count)** — кількість аргументів командного рядка.
- ❖ **argv (ARGument Vector)** — це масив переданих аргументів.
- ❖ **argv[0]** — ім'я програми.

```
1  #include <iostream>
2  using namespace std;
3
4  int main(int argc, char* argv[]) {
5      cout << "You have entered "<<argc <<" arguments:\n";
6      for (int i = 0; i < argc; i++) {
7          cout << argv[i] << endl;
8      }
9      return 0;
10 }
```

```
You have entered 4 arguments:
C:\Users\vtrushevskyy\source\repos\ConsoleApplication4\x64\Debug\ConsoleApplicationFunctions.exe
hello
C++
world
```

Дякую!