C++

پریسا حامد روح بخش

موسسه ی پارس پژوهان

فصل هفتم

- Functions
- Function Overloading
- Recursion

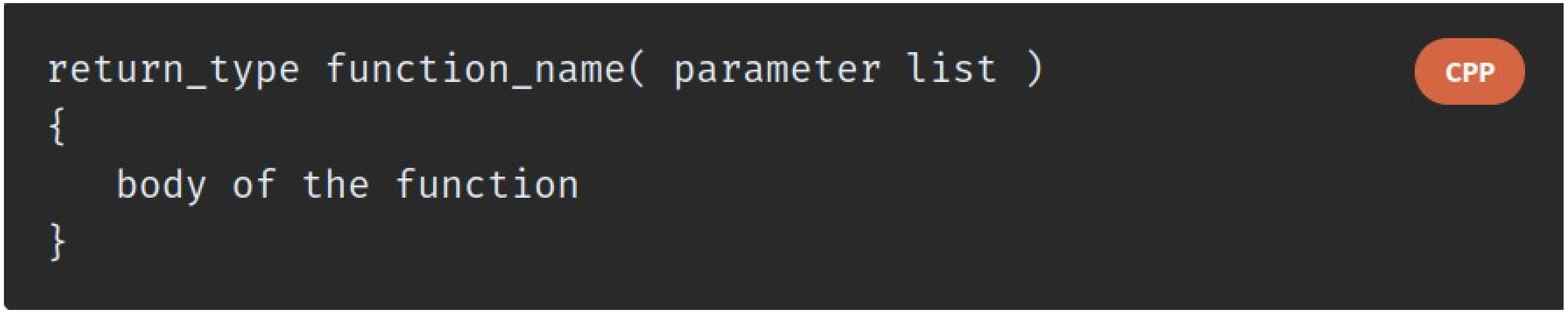
Pars Pajouhaan
پارس پژوهان
بامجوز رسمی از سازمان فنی و حرفه‌ای

Functions

- A function is a group of statements that perform a particular task.
- Using functions can have many advantages, including the following:
- - You can reuse the code within a function.
- - You can easily test individual functions.
- - If it's necessary to make any code modifications, you can make modifications within a single function, without altering the program structure.
- - You can use the same function for different inputs.

Functions

- **Every valid C++ program has at least one function - the main() function.**

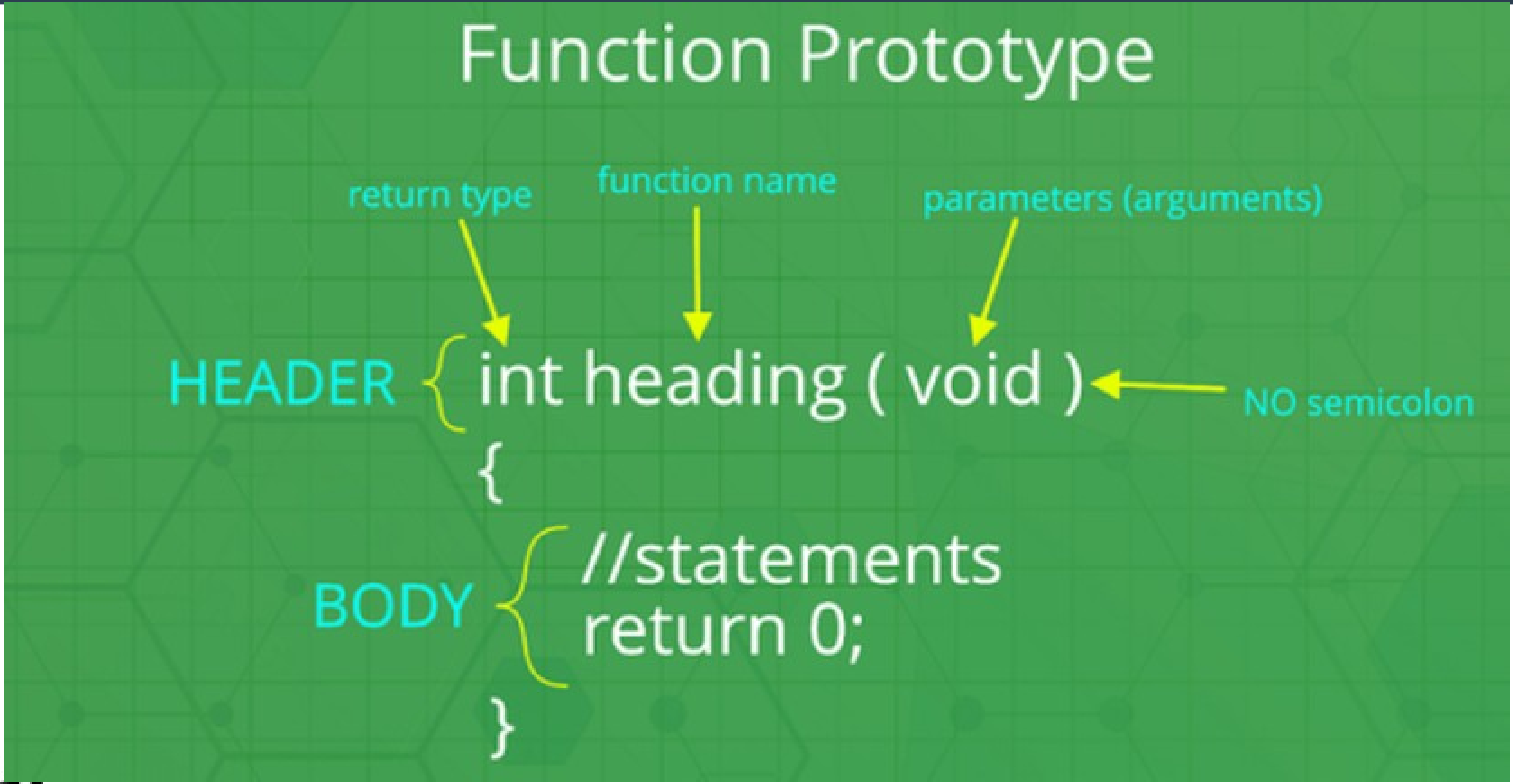


```cpp
return_type function_name( parameter list )
{
   body of the function
}
```

CPP

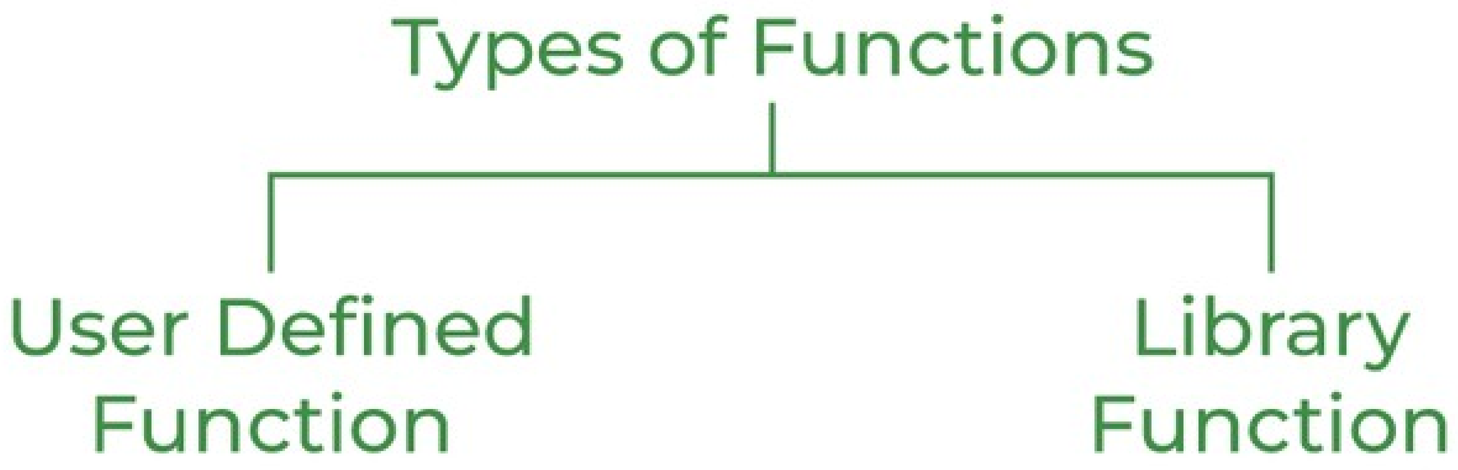
Parameters are **optional**; that is, you can have a function with no parameters.

Functions



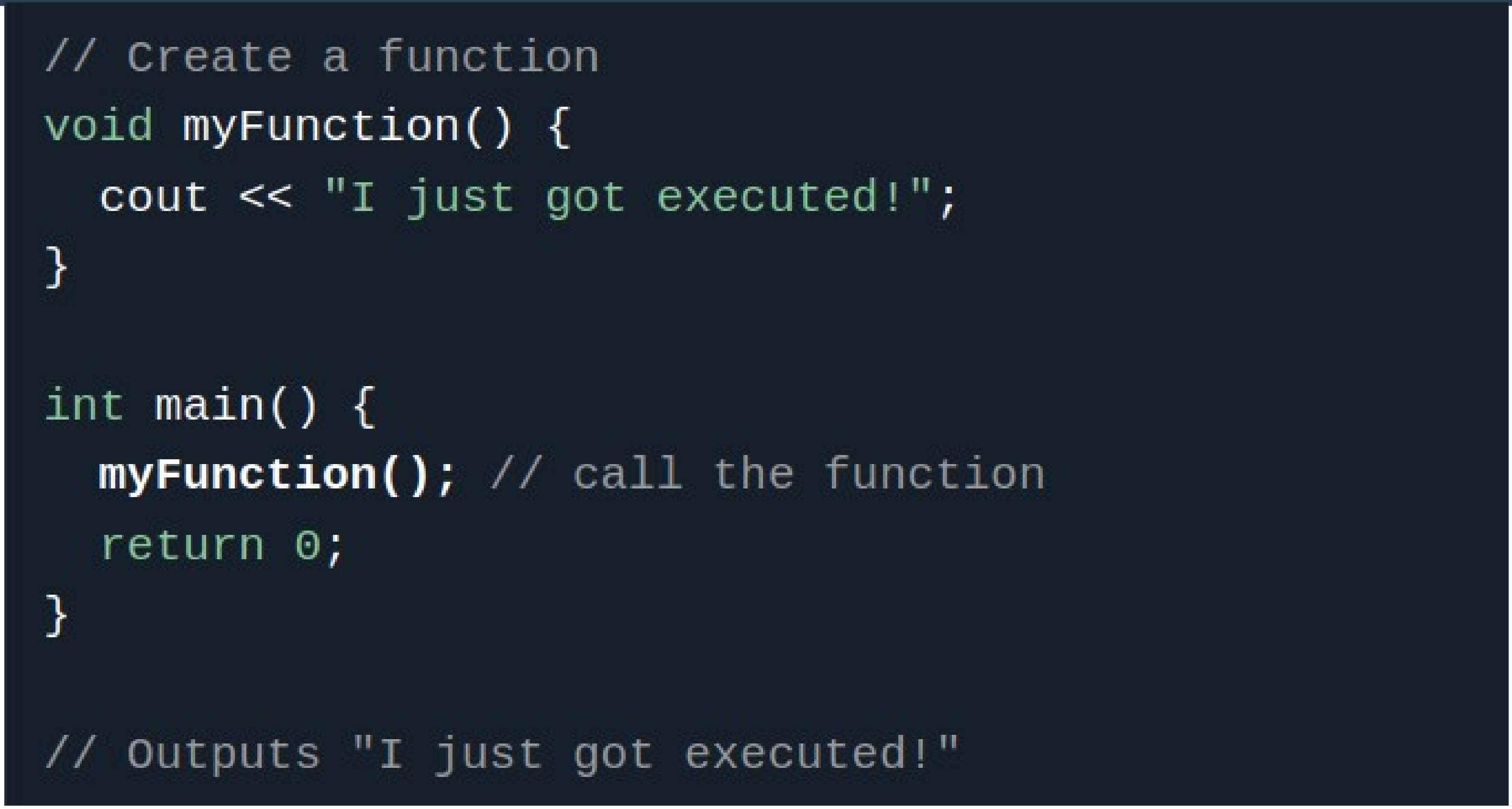
```
int heading ( void )
{
    //statements
    return 0;
}
```

Functions


Types of Functions
User Defined Function
Library Function

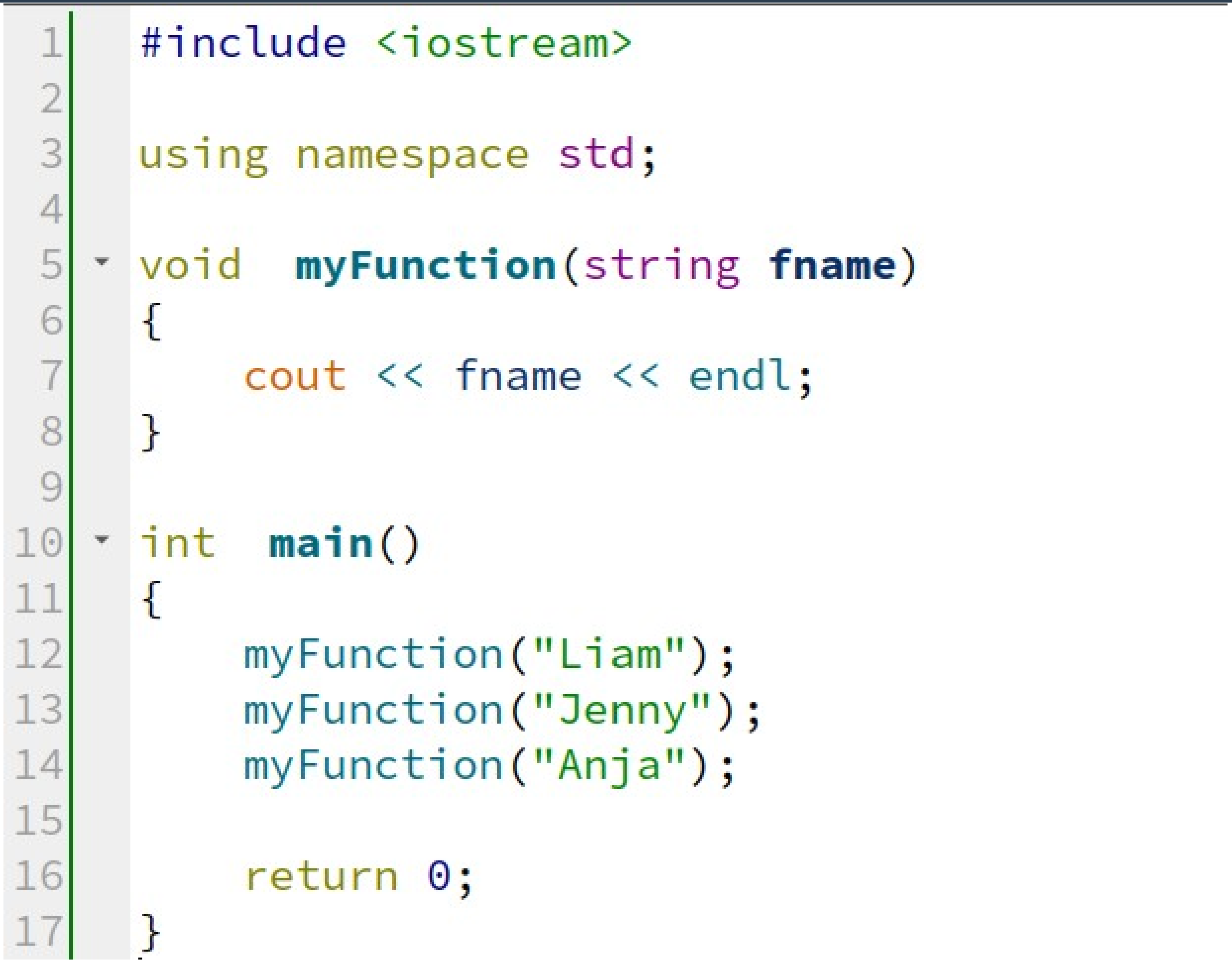
Functions

Example



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

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

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

Functions



```
#include <iostream>

using namespace std;

void  myFunction(string fname)
{
    cout << fname << endl;
}

int  main()
{
    myFunction("Liam");
    myFunction("Jenny");
    myFunction("Anja");

    return 0;
}
```

Functions

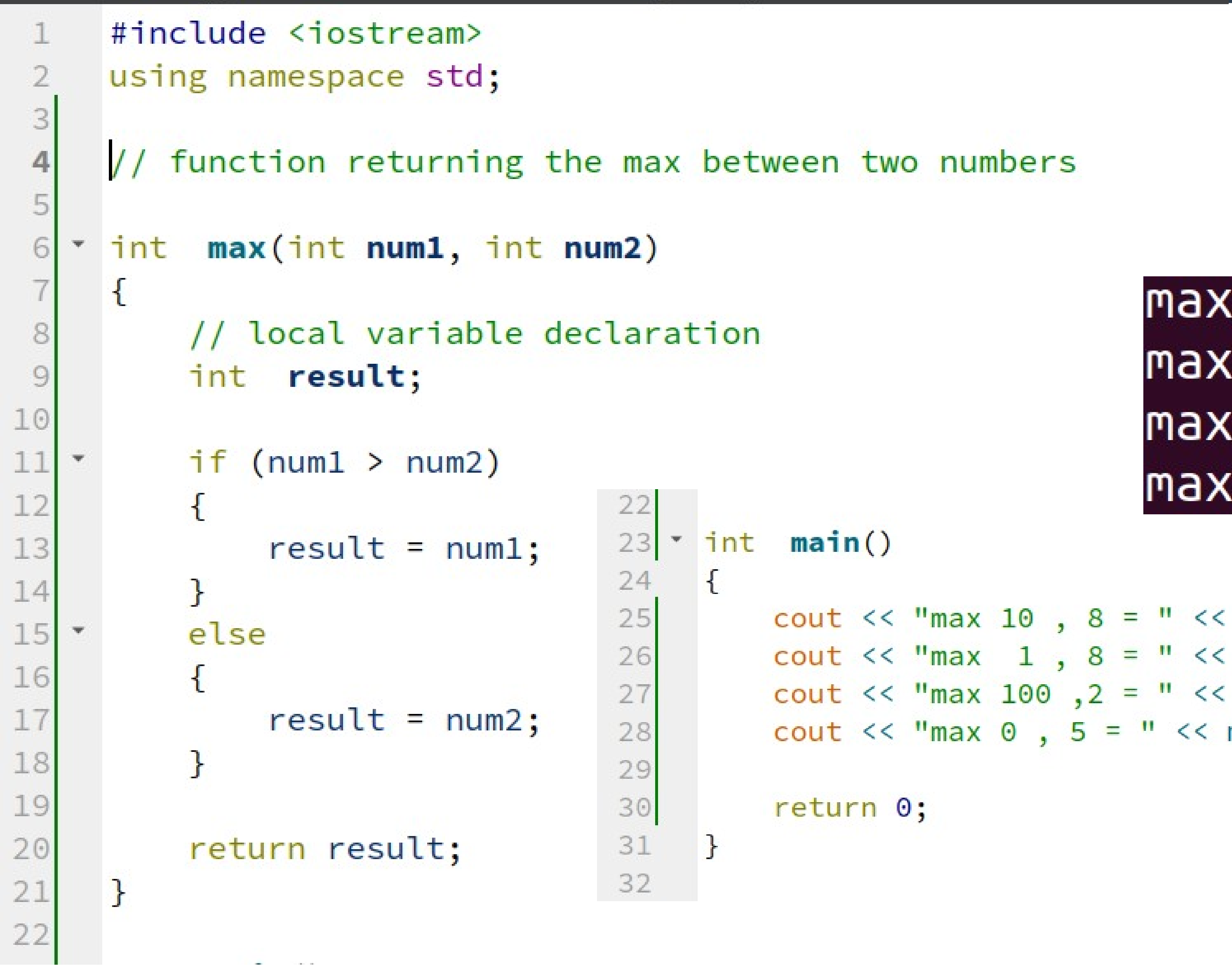
- **Return Type** – A function may **return** a value. The **return_type** is the data type of the value the function returns. Some functions perform the desired operations **without returning a value**. In this case, the return_type is the keyword void.

Pars Pajouhaan
پارس پژوهان
بامجوز رسمی از سازمان فنی وحرفه‌ای

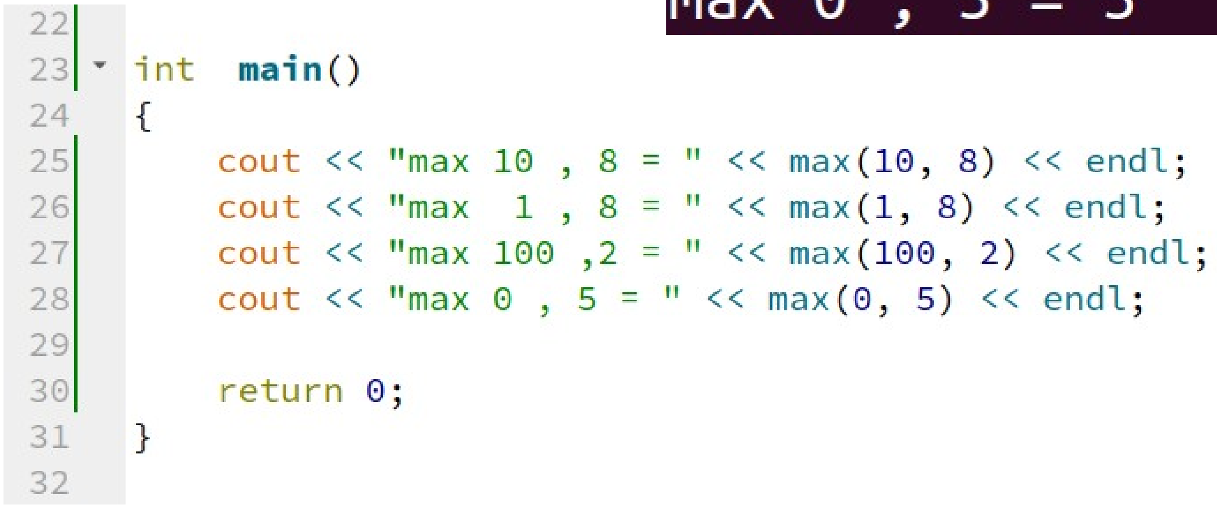
تمرین

- تابعی بنویسید که ماکسیمم دو عدد را محاسبه کند.
- تابعی بنویسید که ماکسیمم اعداد در یک آرایه را محاسبه کند.
- تابعی بنویسید که مینیمم دو عدد را محاسبه کند.
- تابعی بنویسید که مینیمم اعداد در یک آرایه را محاسبه کند.

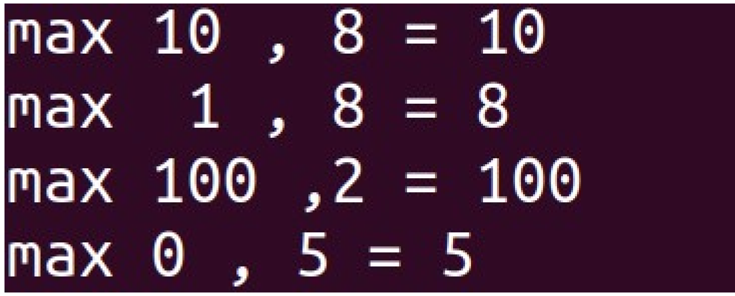
جواب



```cpp
#include <iostream>
using namespace std;

// function returning the max between two numbers

int  max(int num1, int num2)
{
    // local variable declaration
    int  result;

    if (num1 > num2)
    {
        result = num1;
    }
    else
    {
        result = num2;
    }

    return result;
}

```

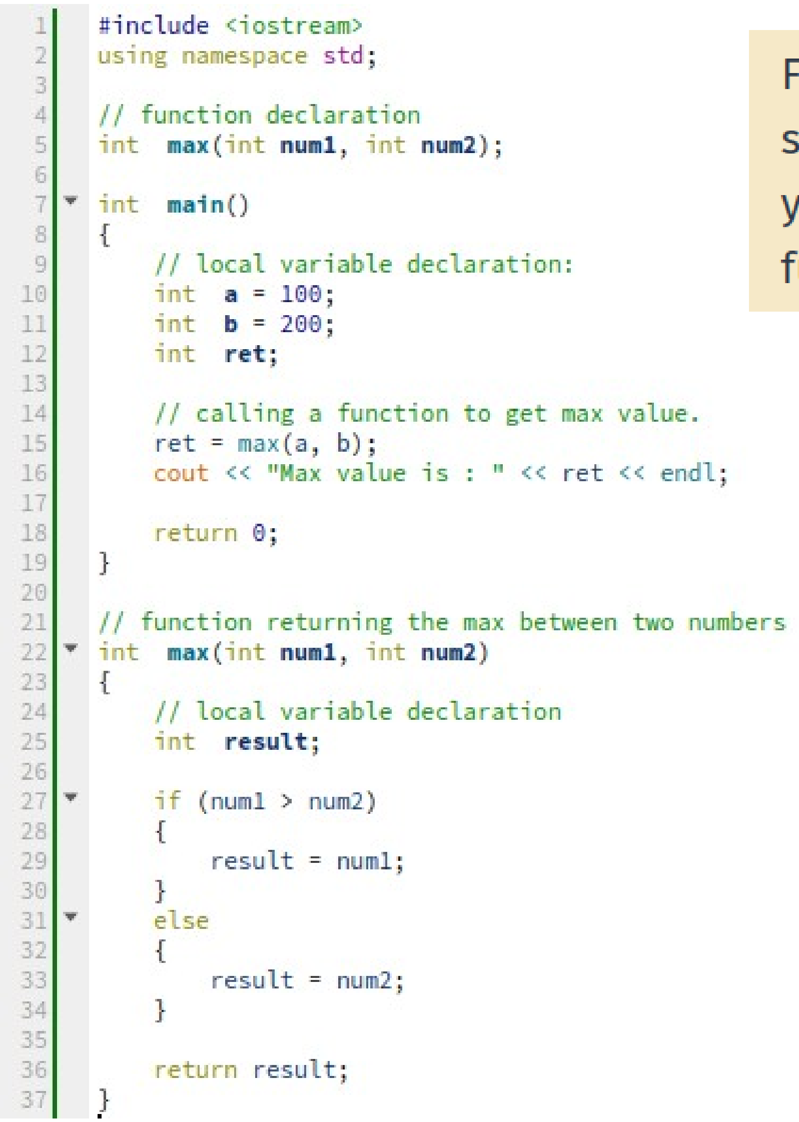


```cpp

int  main()
{
    cout << "max 10 , 8 = " << max(10, 8) << endl;
    cout << "max  1 , 8 = " << max(1, 8) << endl;
    cout << "max 100 ,2 = " << max(100, 2) << endl;
    cout << "max 0 , 5 = " << max(0, 5) << endl;

    return 0;
}

```



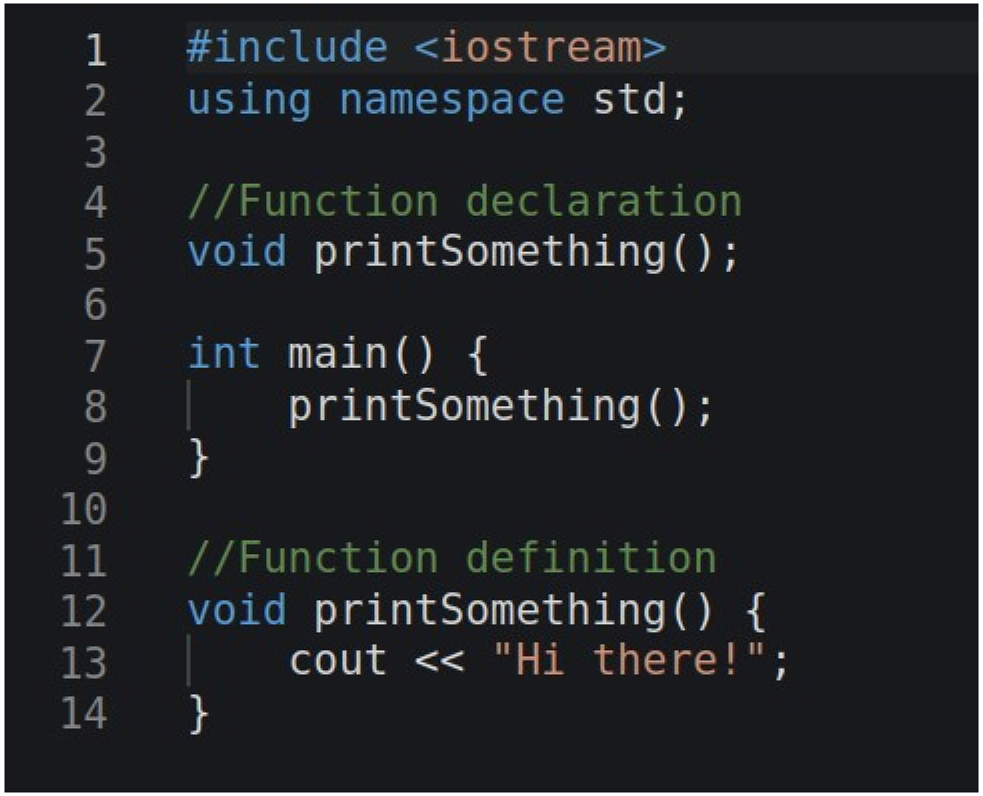
```
max 10 , 8 = 10
max  1 , 8 = 8
max 100 ,2 = 100
max 0 , 5 = 5
```

تمرین



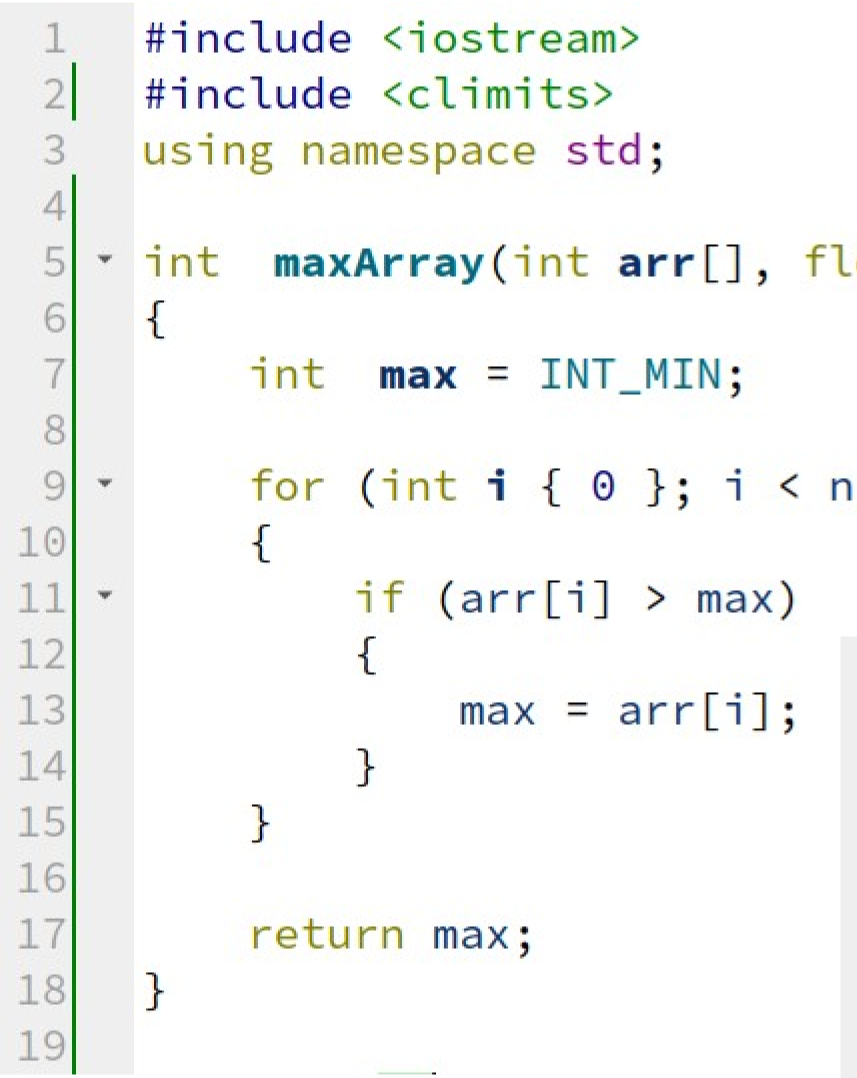
```cpp
#include <iostream>
using namespace std;

// function declaration
int  max(int num1, int num2);

int  main()
{
     // local variable declaration:
     int  a = 100;
     int  b = 200;
     int  ret;

     // calling a function to get max value.
     ret = max(a, b);
     cout << "Max value is : " << ret << endl;

     return 0;
}

// function returning the max between two numbers
int  max(int num1, int num2)
{
     // local variable declaration
     int  result;

     if (num1 > num2)
     {
        result = num1;
     }
     else
     {
        result = num2;
     }

     return result;
}
```

Function declaration is required when you define a function in one source file and you call that function in another file. In such case, you should declare the function at the top of the file calling the function.

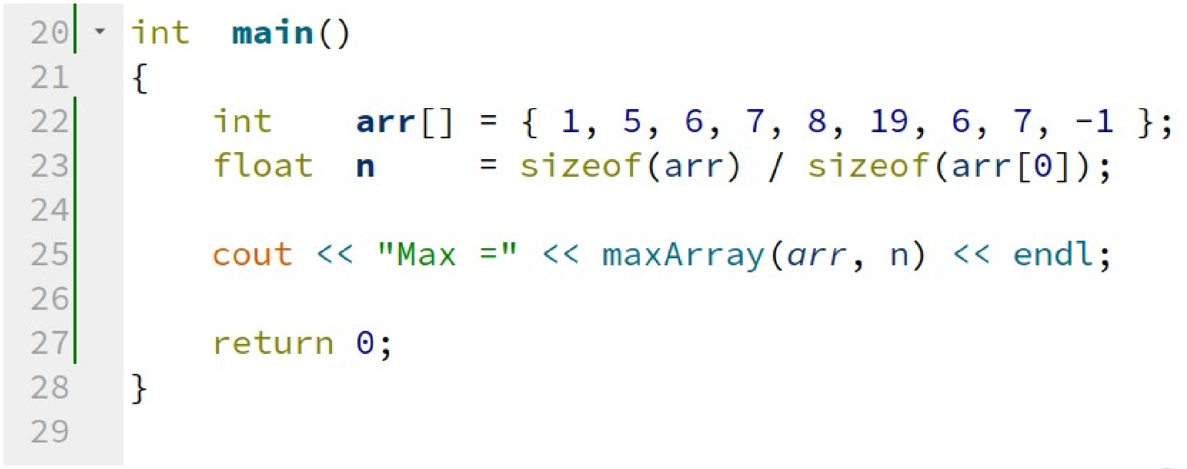


```cpp
#include <iostream>
using namespace std;

//Function declaration
void printSomething();

int main() {
    printSomething();
}

//Function definition
void printSomething() {
    cout << "Hi there!";
}
```

تمرین

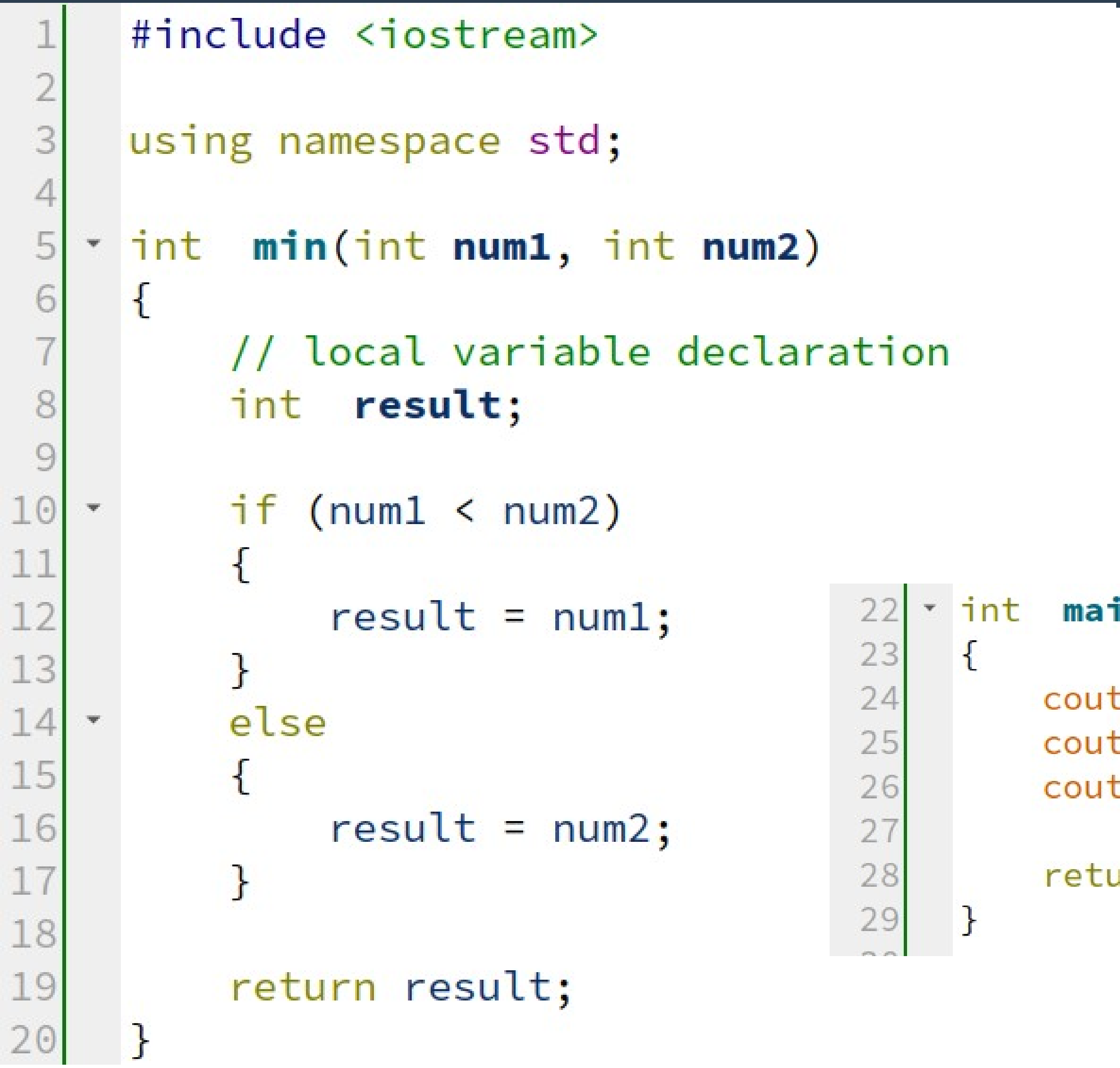


```cpp
#include <iostream>
#include <climits>
using namespace std;

int  maxArray(int arr[], float n)
{
    int  max = INT_MIN;

    for (int i { 0 }; i < n; i++)
    {
        if (arr[i] > max)
        {
            max = arr[i];
        }
    }

    return max;
}

```

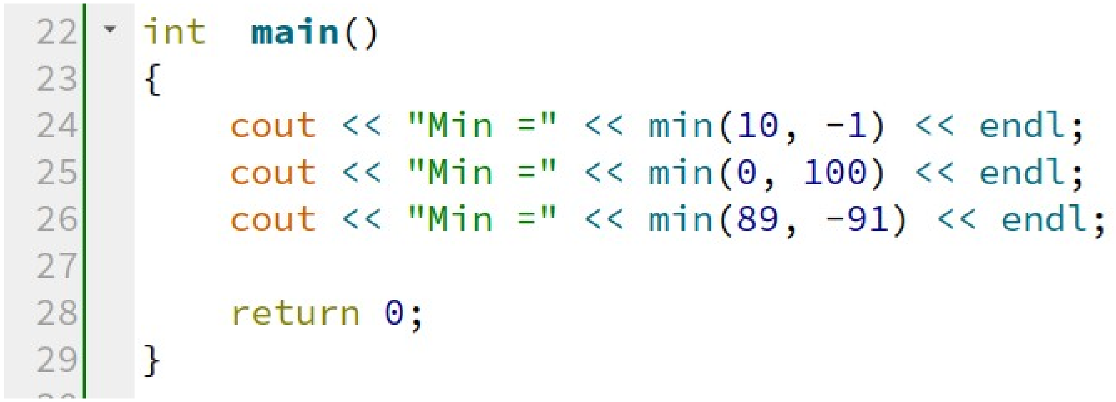


```cpp
int  main()
{
    int    arr[] = { 1, 5, 6, 7, 8, 19, 6, 7, -1 };
    float  n     = sizeof(arr) / sizeof(arr[0]);

    cout << "Max =" << maxArray(arr, n) << endl;

    return 0;
}

```

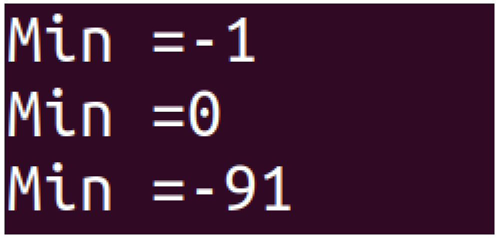
Functions



```cpp
#include <iostream>

using namespace std;

int  min(int num1, int num2)
{
    // local variable declaration
    int  result;

    if (num1 < num2)
    {
        result = num1;
    }
    else
    {
        result = num2;
    }

    return result;
}
```

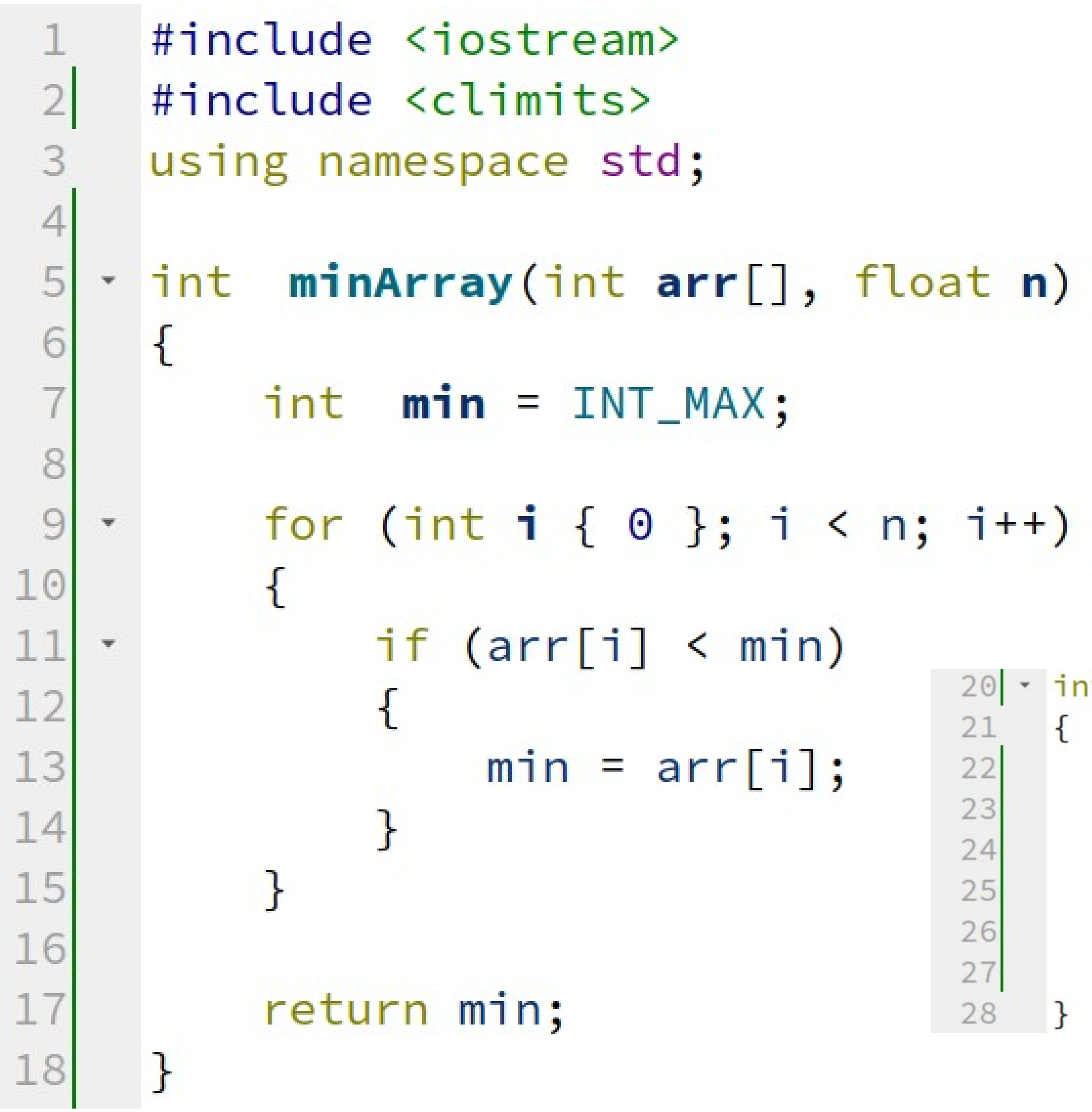


```cpp
int  main()
{
    cout << "Min =" << min(10, -1) << endl;
    cout << "Min =" << min(0, 100) << endl;
    cout << "Min =" << min(89, -91) << endl;

    return 0;
}
```

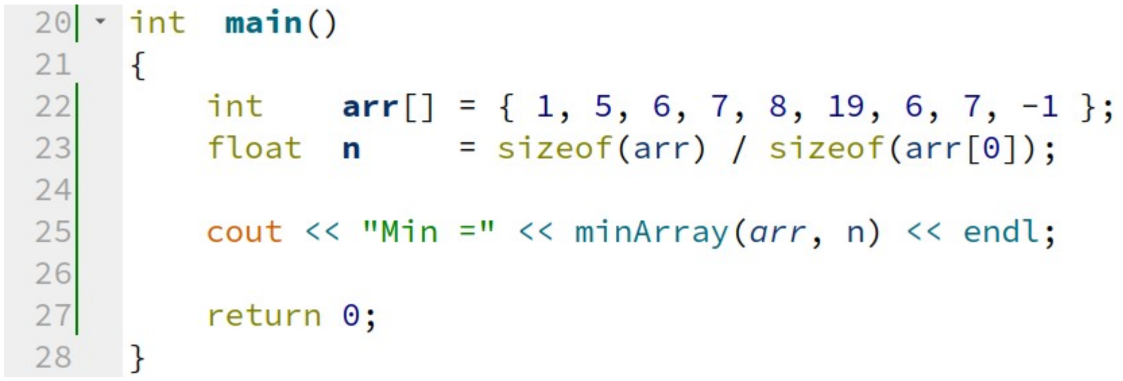


```
Min =-1
Min =0
Min =-91
```

تمرین

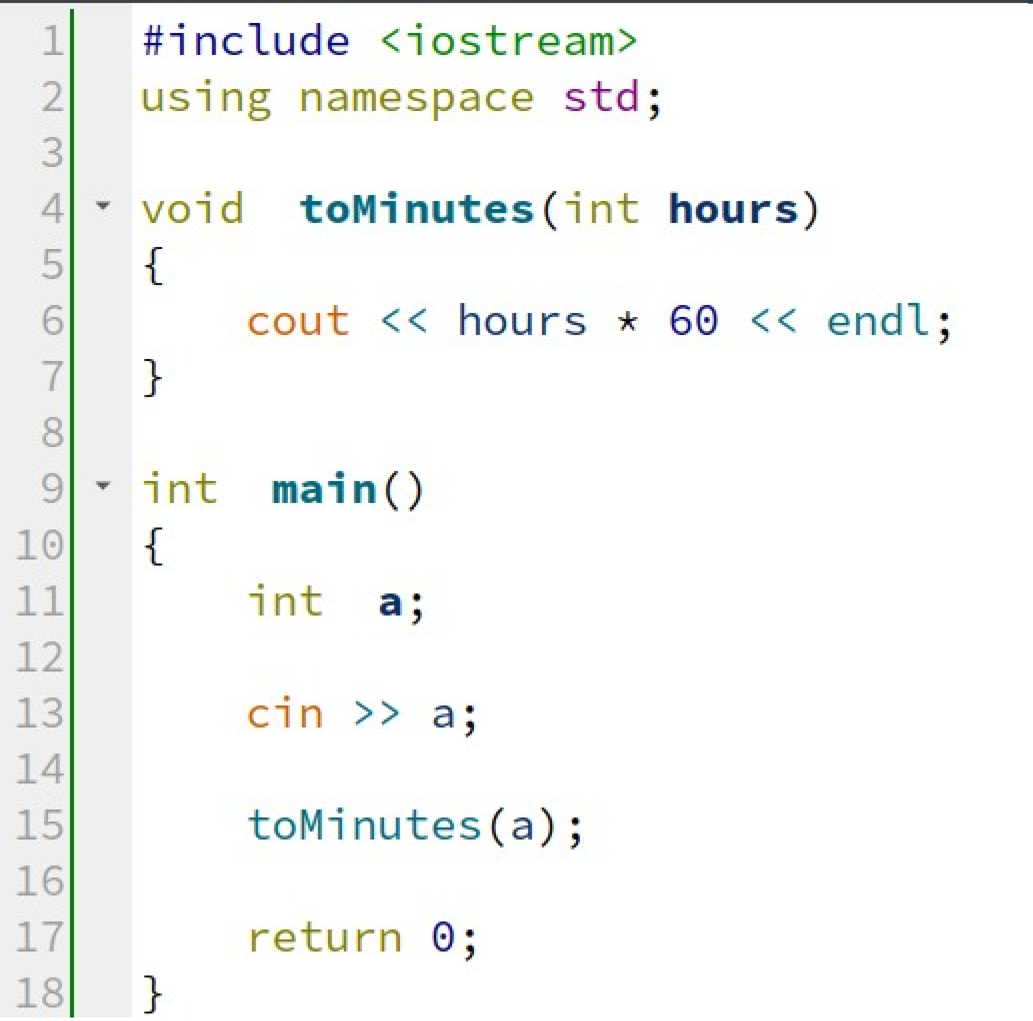


```cpp
#include <iostream>
#include <climits>
using namespace std;

int  minArray(int arr[], float n)
{
     int  min = INT_MAX;

     for (int i { 0 }; i < n; i++)
     {
          if (arr[i] < min)
          {
               min = arr[i];
          }
     }

     return min;
}
```



```cpp
int  main()
{
     int    arr[] = { 1, 5, 6, 7, 8, 19, 6, 7, -1 };
     float  n     = sizeof(arr) / sizeof(arr[0]);

     cout << "Min =" << minArray(arr, n) << endl;

     return 0;
}
```

تمرین



```cpp
#include <iostream>
using namespace std;

void  toMinutes(int hours)
{
    cout << hours * 60 << endl;
}

int  main()
{
    int  a;

    cin >> a;

    toMinutes(a);

    return 0;
}
```

Function Arguments

- **Call by Value**

  This method copies the actual value of an argument into the formal parameter of the function. In this case, changes made to the parameter inside the function have no effect on the argument.

Functions

- **Call by Pointer**

  This method copies the address of an argument into the formal parameter. Inside the function, the address is used to access the actual argument used in the call. This means that changes made to the parameter affect the argument.

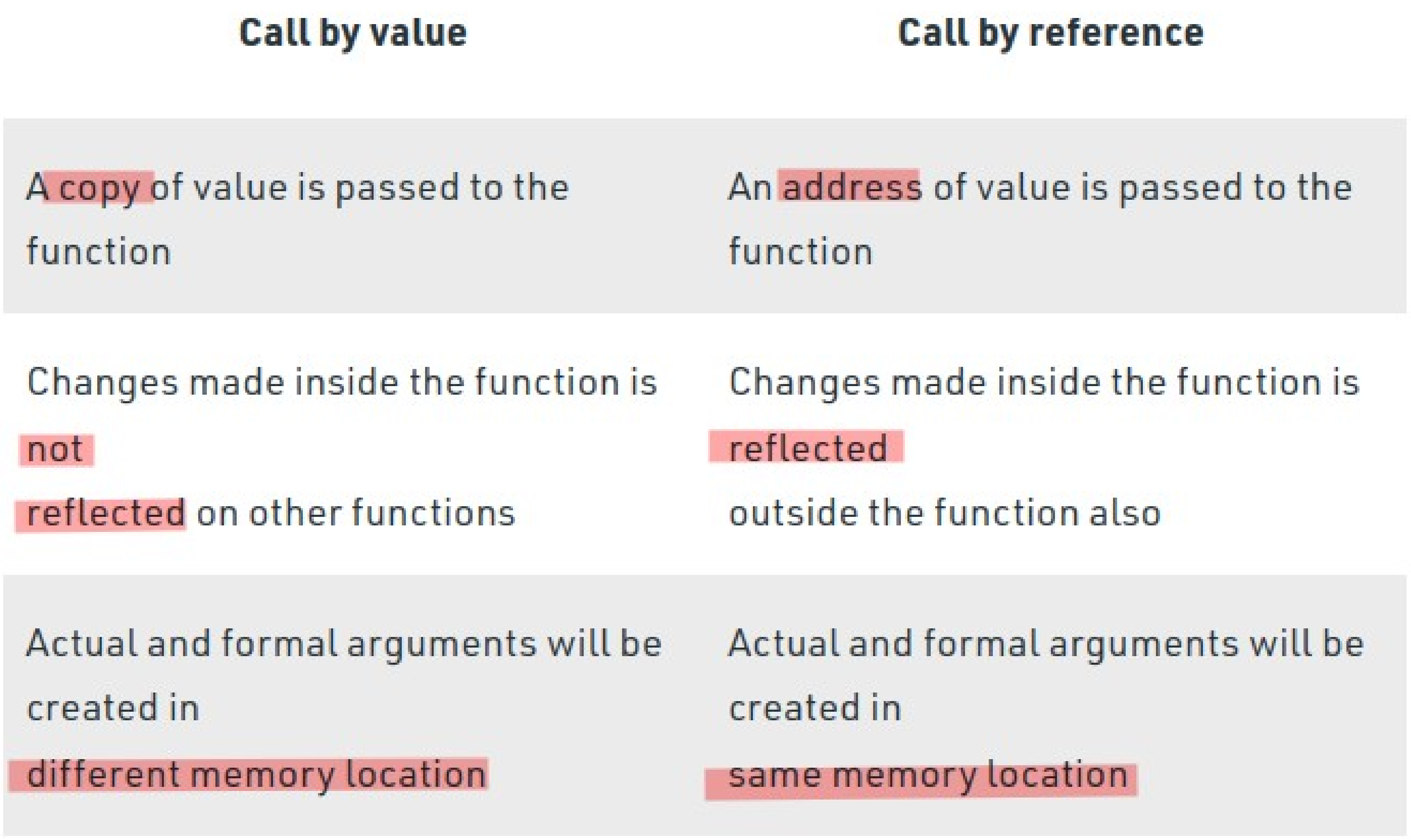
Functions

- Call by Reference

  This method copies the reference of an argument into the formal parameter. Inside the function, the reference is used to access the actual argument used in the call. This means that changes made to the parameter affect the argument.

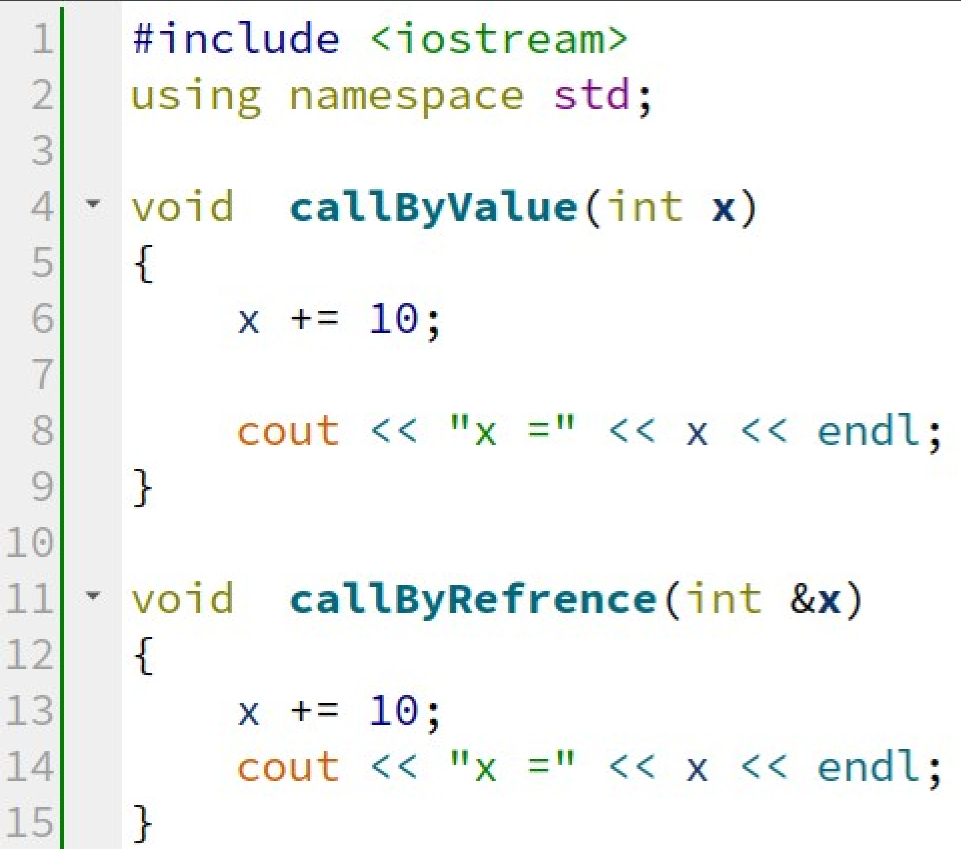
By default, C++ uses call by value to pass arguments.

Functions

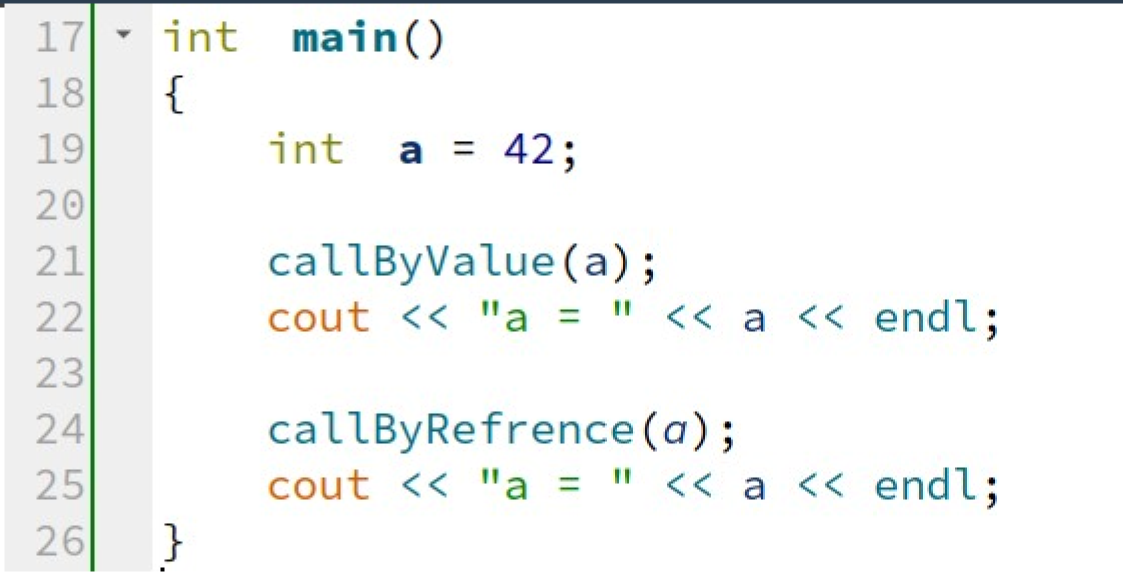


Call by value	Call by reference
A copy of value is passed to the function	An address of value is passed to the function
Changes made inside the function is not reflected on other functions	Changes made inside the function is reflected outside the function also
Actual and formal arguments will be created in different memory location	Actual and formal arguments will be created in same memory location

تمرین

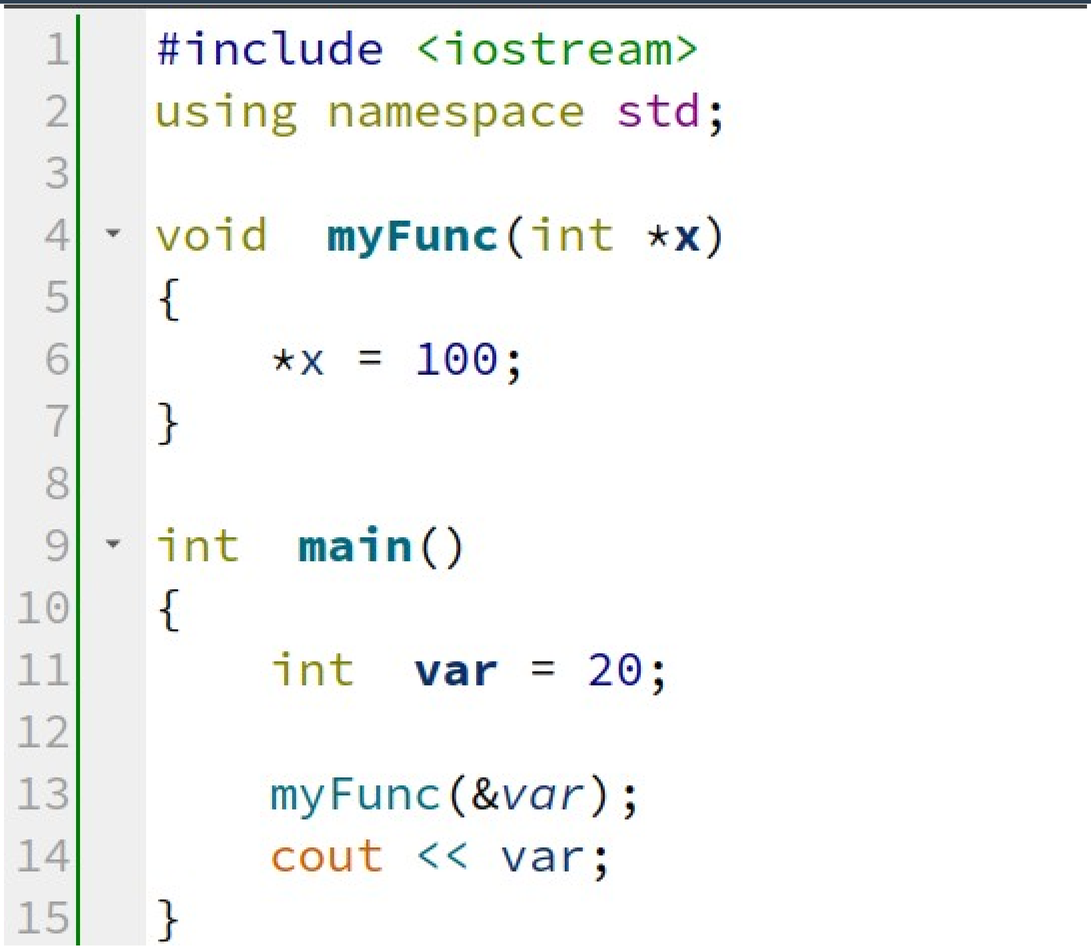


```cpp
#include <iostream>
using namespace std;

void  callByValue(int x)
{
    x += 10;

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

void  callByRefrence(int &x)
{
    x += 10;
    cout << "x =" << x << endl;
}
```



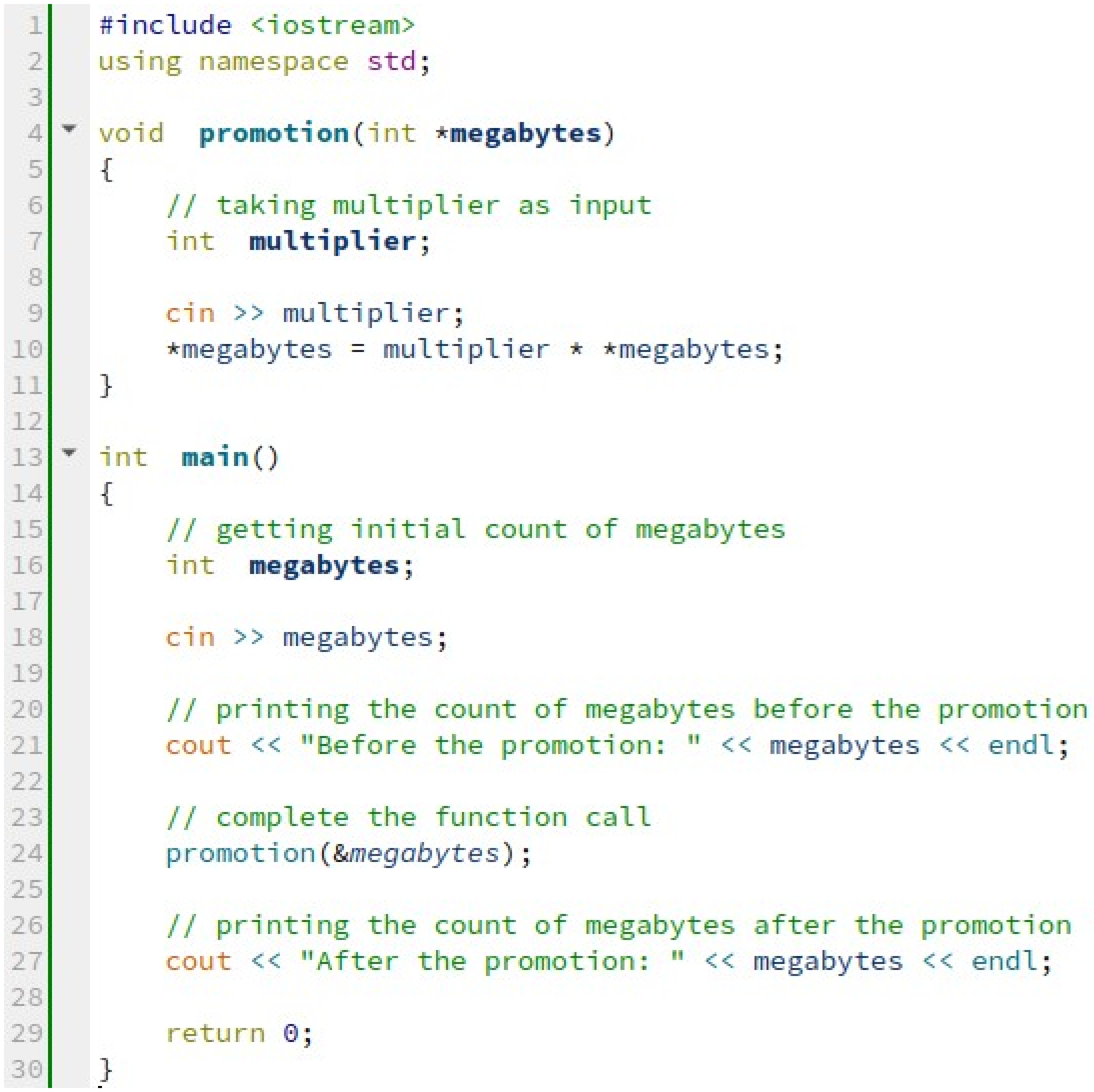
```cpp
int  main()
{
    int  a = 42;

    callByValue(a);
    cout << "a = " << a << endl;

    callByRefrence(a);
    cout << "a = " << a << endl;
}
```

تمرین



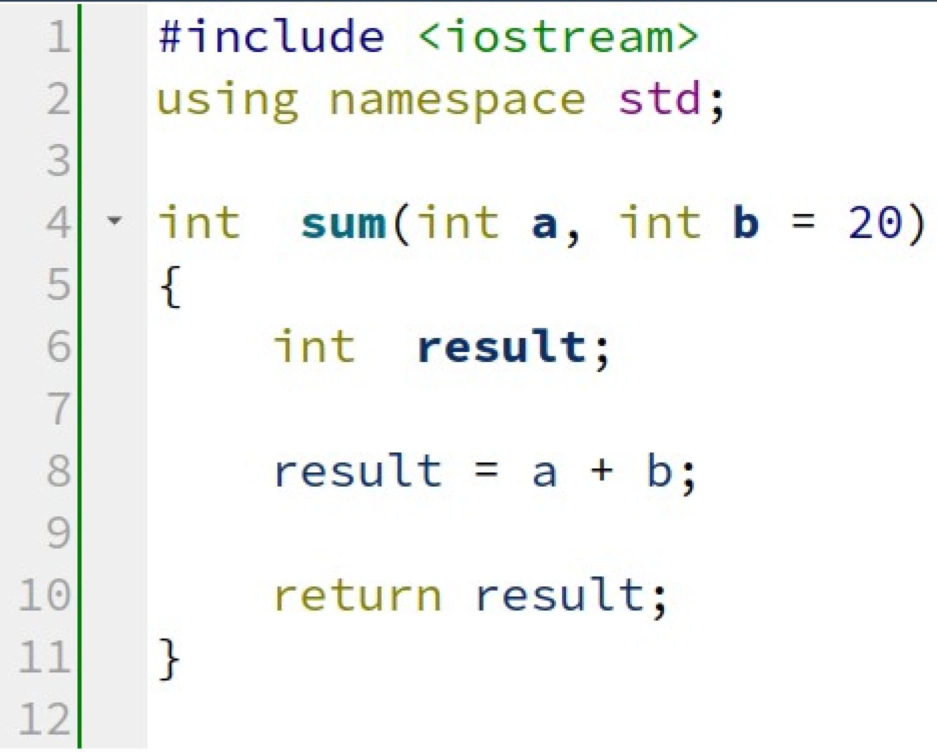
```cpp
#include <iostream>
using namespace std;

void  myFunc(int *x)
{
    *x = 100;
}

int  main()
{
    int  var = 20;

    myFunc(&var);
    cout << var;
}
```

تمرین

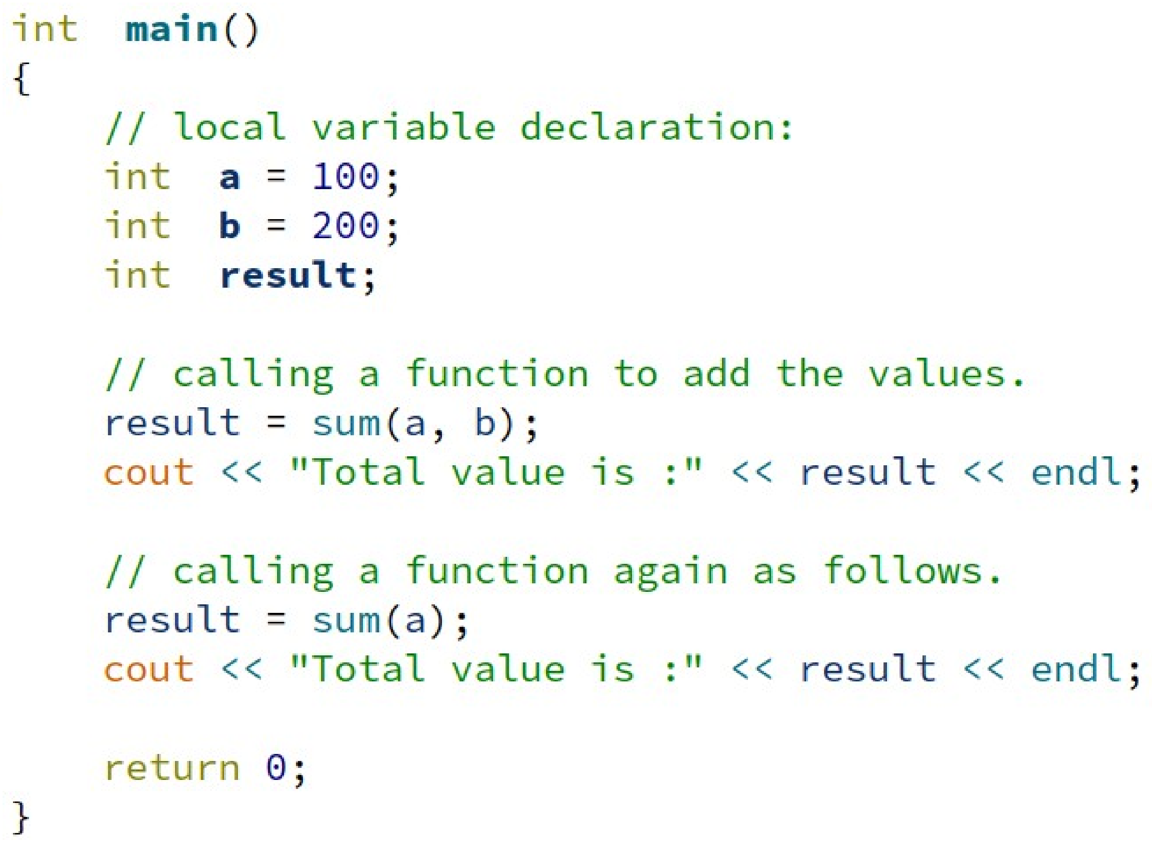


```cpp
#include <iostream>
using namespace std;

void  promotion(int *megabytes)
{
    // taking multiplier as input
    int  multiplier;

    cin >> multiplier;
    *megabytes = multiplier * *megabytes;
}

int  main()
{
    // getting initial count of megabytes
    int  megabytes;

    cin >> megabytes;

    // printing the count of megabytes before the promotion
    cout << "Before the promotion: " << megabytes << endl;

    // complete the function call
    promotion(&megabytes);

    // printing the count of megabytes after the promotion
    cout << "After the promotion: " << megabytes << endl;

    return 0;
}
```

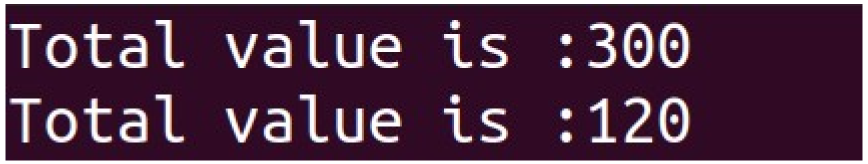
تمرین



```cpp
#include <iostream>
using namespace std;

int  sum(int a, int b = 20)
{
    int  result;

    result = a + b;

    return result;
}

```

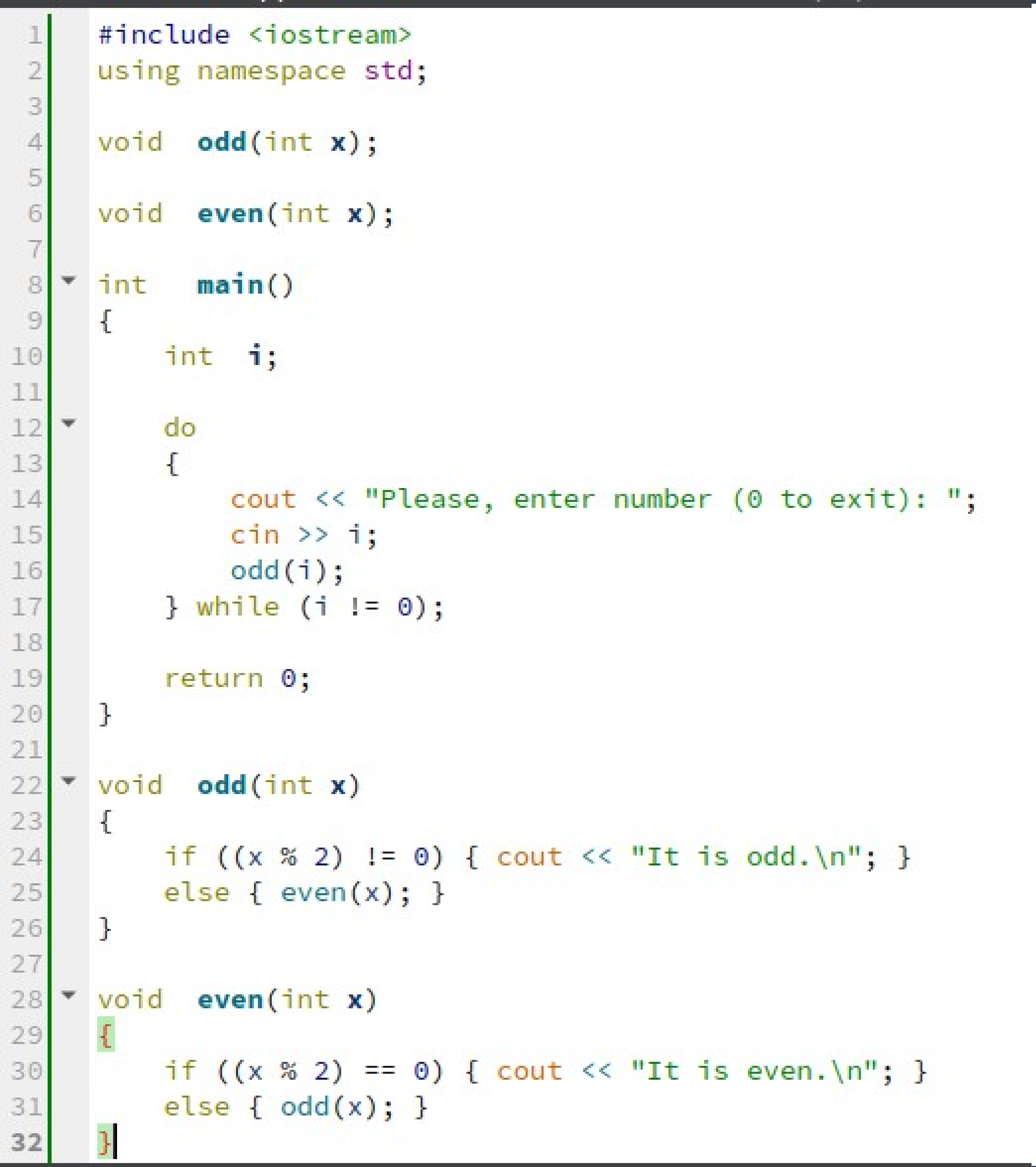


```cpp
int  main()
{
    // local variable declaration:
    int  a = 100;
    int  b = 200;
    int  result;

    // calling a function to add the values.
    result = sum(a, b);
    cout << "Total value is :" << result << endl;

    // calling a function again as follows.
    result = sum(a);
    cout << "Total value is :" << result << endl;

    return 0;
}
```



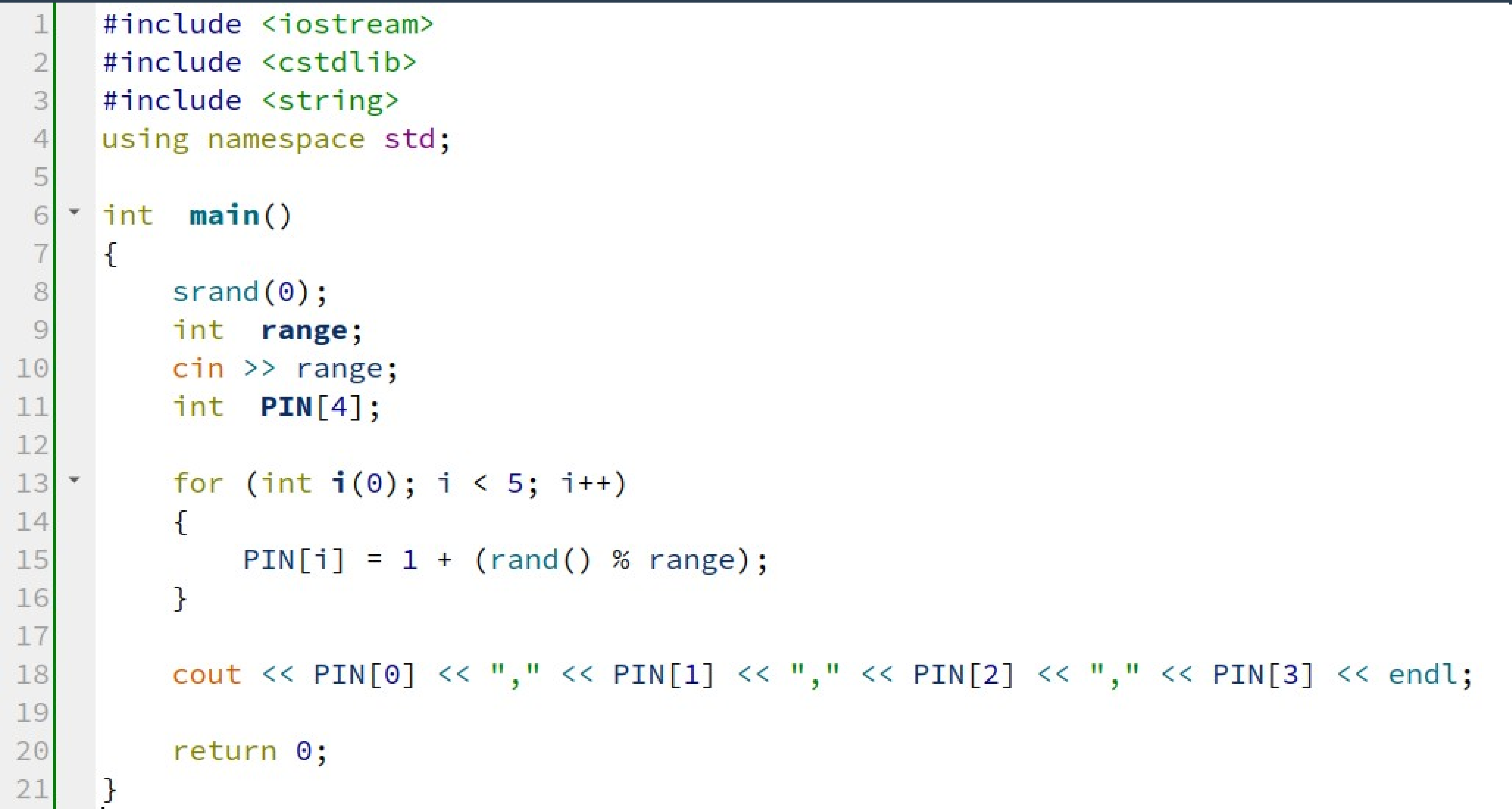
```
Total value is :300
Total value is :120
```

تمرین



```
#include <iostream>
using namespace std;

void  odd(int x);

void  even(int x);

int   main()
{
    int  i;

    do
    {
        cout << "Please, enter number (0 to exit): ";
        cin >> i;
        odd(i);
    } while (i != 0);

    return 0;
}

void  odd(int x)
{
    if ((x % 2) != 0) { cout << "It is odd.\n"; }
    else { even(x); }
}

void  even(int x)
{
    if ((x % 2) == 0) { cout << "It is even.\n"; }
    else { odd(x); }
}
```

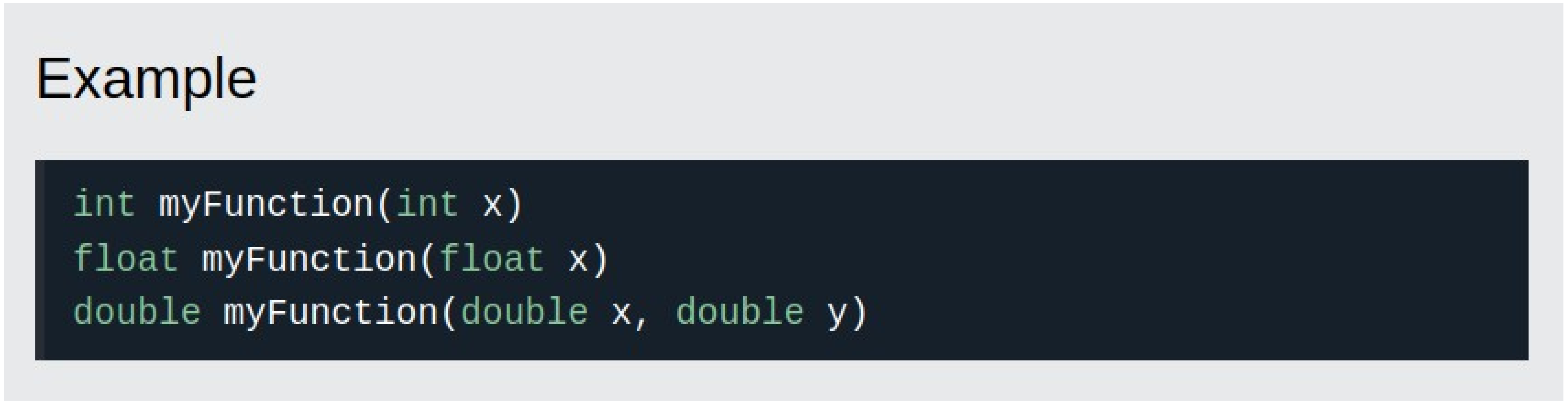
Functions



```cpp
#include <iostream>
#include <cstdlib>
#include <string>
using namespace std;

int  main()
{
    srand(0);
    int  range;
    cin >> range;
    int  PIN[4];

    for (int i(0); i < 5; i++)
    {
        PIN[i] = 1 + (rand() % range);
    }

    cout << PIN[0] << "," << PIN[1] << "," << PIN[2] << "," << PIN[3] << endl;

    return 0;
}
```

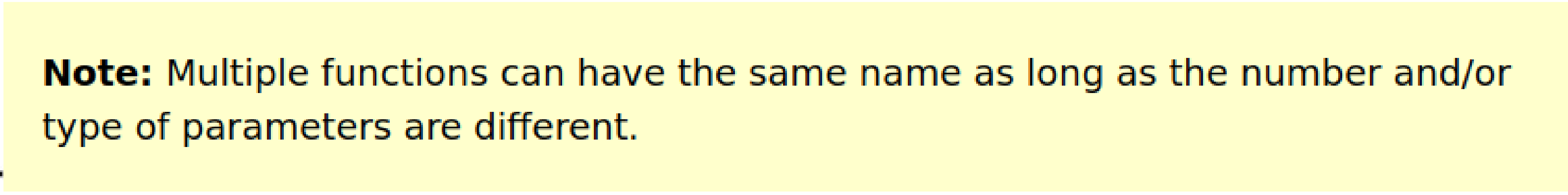
Function Overloading

- With function overloading, multiple functions can have the **same name** with **different parameters**:



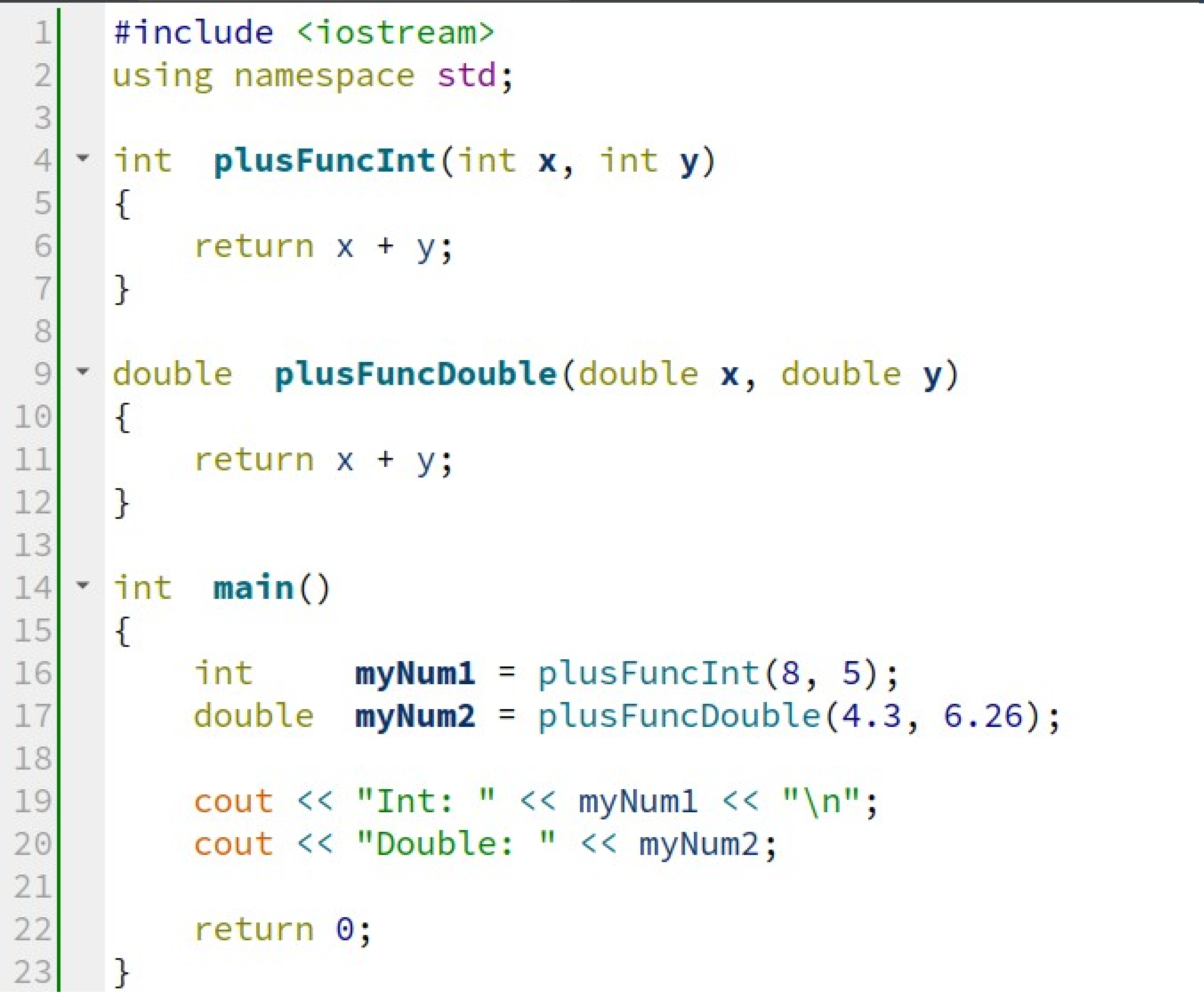
Example

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



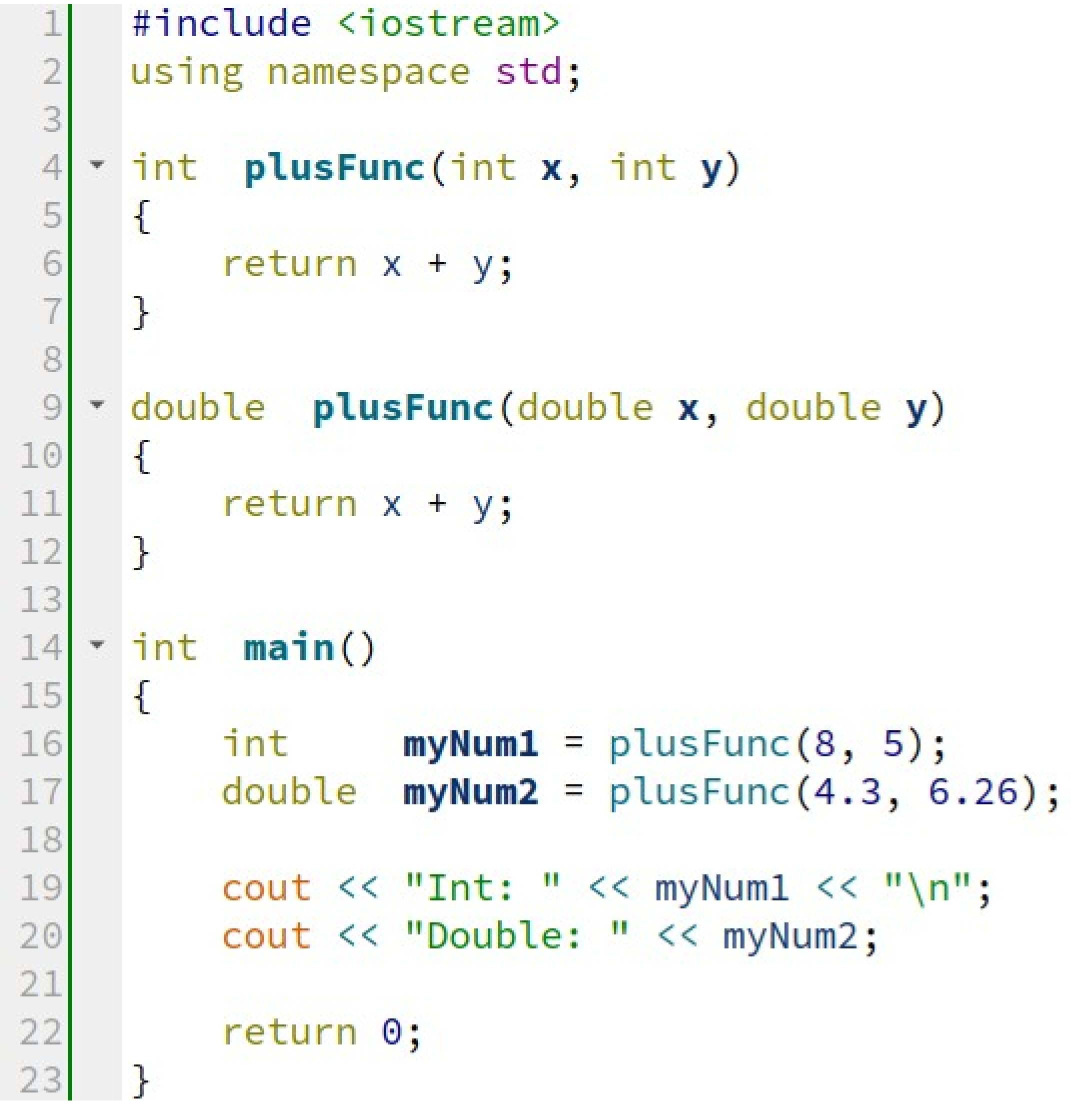
Note: Multiple functions can have the same name as long as the number and/or type of parameters are different.

Function Overloading



```cpp
#include <iostream>
using namespace std;

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;
}
```

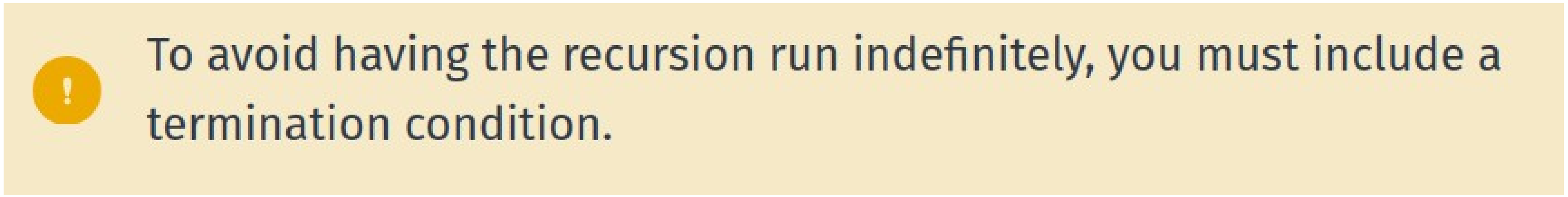
Function Overloading



```cpp
#include <iostream>
using namespace std;

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;
}
```

Recursion

- **Recursion is the technique of making a function call itself. This technique provides a way to break complicated problems down into simple problems which are easier to solve.**



! To avoid having the recursion run indefinitely, you must include a termination condition.

Recursion

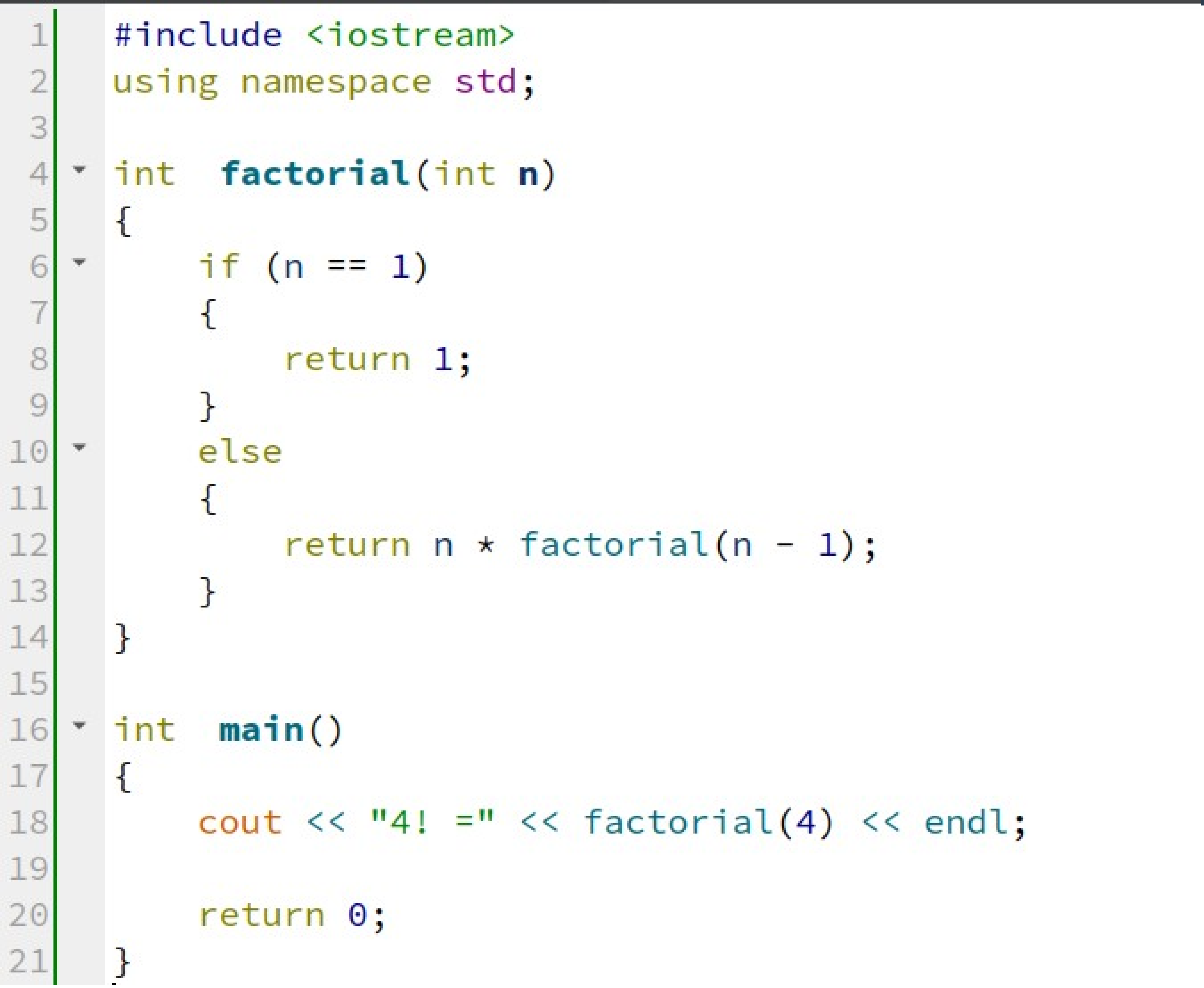
- To demonstrate recursion, let's create a program to calculate a number's factorial.

  In mathematics, the term factorial refers to the product of all positive integers that are less than or equal to a specific non-negative integer (n). The factorial of n is denoted as n!

  - For example:

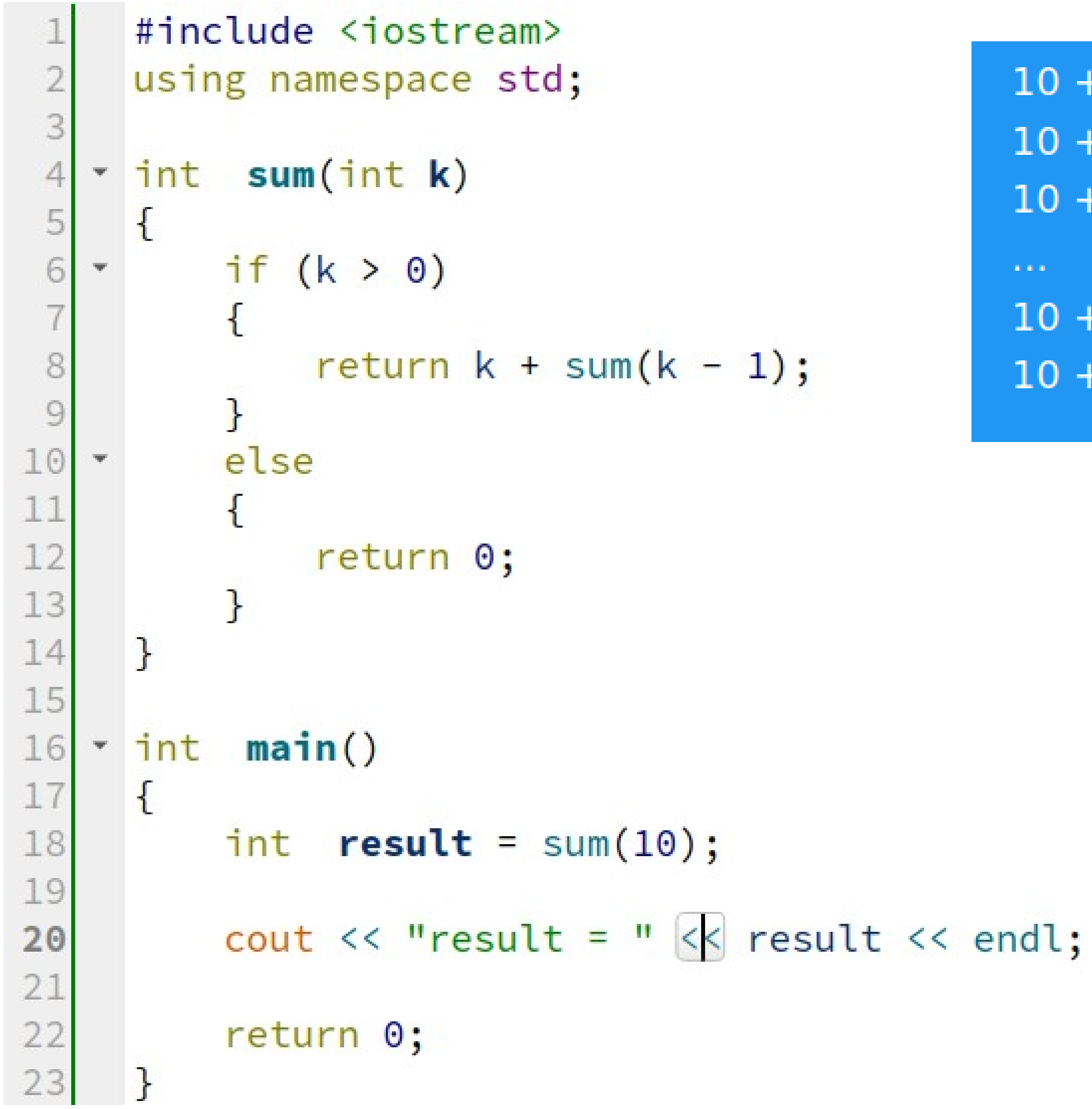
```
4! = 4 * 3 * 2 * 1 = 24
```

Recursion



```
#include <iostream>
using namespace std;

int  factorial(int n)
{
    if (n == 1)
    {
        return 1;
    }
    else
    {
        return n * factorial(n - 1);
    }
}

int  main()
{
    cout << "4! =" << factorial(4) << endl;

    return 0;
}
```

Recursion



```
#include <iostream>
using namespace std;

int  sum(int k)
{
    if (k > 0)
    {
        return k + sum(k - 1);
    }
    else
    {
        return 0;
    }
}

int  main()
{
    int  result = sum(10);

    cout << "result = " << result << endl;

    return 0;
}
```

10 + sum(9)
10 + (9 + sum(8))
10 + (9 + (8 + sum(7)))
…
10 + 9 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1 + sum(0)
10 + 9 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1 + 0

Question ?