Return value of main

Well, there is a catch: If the execution of main ends normally without encountering a return statement the compiler assumes the function ends with an implicit return statement:
return 0;
Note that this only applies to function main for historical reasons. All other functions with a return type shall end with a proper return statement that includes a return value, even if this is never used.

Functions

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

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

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

A function can be called multiple times..

Function Declaration and Definition

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

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

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

We can’t write function definition without the function declaration……………….

Parameters and Arguments

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

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

int addition (int a, int b)
{
  int r;
  r=a+b;.
  return r;
}

int main ()
{
  int z;
  z = addition (5,3);                                    //arguments
  cout << "The result is " << z;
}
```

Default values in parameters

In C++, functions can also have optional parameters, for which no arguments are required in the call

```
#include <iostream>
using namespace std;
int divide (int a, int b=2)
{
  int r;
  r=a/b;

  return (r);

}

int main ()

{

  cout << divide (12) << '\n';

  cout << divide (20,4) << '\n';

  return 0;
```

outputs:

```
6

5
```

```
void myFunction(string country = "Norway") {
//optional parameter = default value
  cout << country << "\n";
}

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

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

---

while we use the void function we can’t return the values…………..

```
int myFunction(int x, int y) {
  return x + y;
}

int main() {
  int z = myFunction(5, 3);
  cout << z;
  return 0;
}
// Outputs 8
```

Pass By Reference

As previous ,we used normal variables when we passed parameters to a function. We can also pass a reference to the function. This can be useful when you need to change the value of the arguments.[ampersand (&)]

```
void modifyStr(string &str) {
  str += " World!";
}

int main() {
  string greeting = "Hello";
  modifyStr(greeting);
  cout << greeting;
  return 0;
}
```

outputs:Hello World!

```
#include <iostream>
using namespace std;
void duplicate (int& a, int& b, int& c)
{
  a*=2;
  b*=2;
  c*=2;
}
int main ()
{
  int x=1, y=3, z=7;
  duplicate (x, y, z);
  cout << "x=" << x << ", y=" << y << ", z=" << z;
  return 0;
}
```

```
//Outputs : x=2,y=6,z=14
```

if we don’t use the & outputs will be 1,3,7

Pass Arrays as Function Parameters

```
void myFunction(int myNumbers[5]) {
  for (int i = 0; i < 5; i++) {
    cout << myNumbers[i] << "\n";
  }
}

int main() {
  int myNumbers[5] = {10, 20, 30, 40, 50};
  myFunction(myNumbers);
  return 0;
}
```

outputs:

```
10
20
30
40
50
```

Inline function

In C++, we can declare a function as inline. This copies the function to the location of the function call in compile-time and may make the program execution faster.

```
#include <iostream>

using namespace std;

inline void displayNum(int num) {

    cout << num << endl;

}

int main() {

    // first function call

    displayNum(5);

    // second function call
    displayNum(8);
    // third function call
    displayNum(666);
    return 0;
}
```

Outputs:

```
5
8
666
```

Excessive use of inline functions may increase the programs binary size ,which can negatively impact performance due to cache inefficiency.

Declaring functions

```
// declaring functions prototypes
#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);
}
```

```
Outputs:
Please, enter number (0 to exit): 9
It is odd.
Please, enter number (0 to exit): 6
It is even.
Please, enter number (0 to exit): 1030
It is even.
Please, enter number (0 to exit): 0
It is even.
```

Recursivity

Recursivity is the property that functions have to be called by themselves. It is useful for some tasks, such as sorting elements, or calculating the factorial of numbers.

```
// factorial calculator
#include <iostream>
using namespace std;
long factorial (long a)
{
  if (a > 1)
   return (a * factorial (a-1));
  else
   return 1;
}
```

```
int main ()
{
    int num;
   cout<<"Enter the num: \n";
  cin>>num;
long result = factorial(num);
cout<<result<<"\n";
  return 0;
}
```

Outputs:

```
Enter the num:5
120
```

Long data type cannot store such a large value because it exceeds its range.

Code :→

```
// Function to convert Fahrenheit to Celsius
float toCelsius(float fahrenheit) {
  return (5.0 / 9.0) * (fahrenheit - 32.0);
}

int main() {
  // Set a fahrenheit value
  float f_value = 98.8;

  // Call the function with the fahrenheit value
  float result = toCelsius(f_value);

  // Print the fahrenheit value
  cout << "Fahrenheit: " << f_value << "\n";

  // Print the result
  cout << "Convert Fahrenheit to Celsius: " << result << "\n";

  return 0;
}
```

SRAND()

The `srand()` function in C++ seeds the pseudo-random number generator used by the `rand()` function. It is defined in the cstdlib header file.

POST INCREMENT AND PRE INCREMENT

post increment (i++)and pre-increment(++i) are operators used to increase the value of a variable by 1.

---

1. Pre-increment (++i)

The variable is incremented before its value is used in the expression.

```
int i = 5; int x = ++i; // i is incremented first, then assigned to x

 cout << "i: " << i << ", x: " << x;

 // Output: i: 6, x: 6
```

2. Post-increment (i++)

The variable's current value is used in the expression first, then it is incremented.

```
int i = 5; int x = i++; // x is assigned the current value of i, then i is incremented

cout << "i: " << i << ", x: " << x;

// Output: i: 6, x: 5
```

WHAT DOES RETURN DOES IN C++ ?

```
// ******************** EXAMPLE 1 ********************

 #include <iostream>

double square(double length);

double cube(double length);

int main() {

double length = 6.0;
double area = square(length);
double volume = cube(length);
std::cout << "Area: " << area << "cm^2\n";
 std::cout << "Volume: " << volume << "cm^3\n";
return 0;
}
double square(double length){
 return length * length;
}
double cube(double length){
return length * length * length;
}
// ******************** EXAMPLE 2 ********************
#include <iostream>
std::string concatString(std::string string1, std::string string2);
 int main() {
std::string firstName = "Bro";
std::string lastName = "Code";
std::string fullName = concatString(firstName, lastName);
std::cout << "Hello " << fullName; return 0;
}
std::string concatString(std::string string1, std::string string2){
return string1 + " " + string2;
}
```

Here we return a standard string so the return type of this function would be a standard string

It’s the return keyword

SCOPE RESOLUTION OPERATOR

```
#include <iostream>
int myNum = 3; //global
void printNum();
int main() {
int myNum = 1; //local
printNum();
std::cout << "main: " << myNum << '\n'; //local
//std::cout << ::myNum << '\n'; //global
return 0;
}
void printNum(){
int myNum = 2; //local
std::cout << "printNum: "<< myNum << '\n'; //local
//std::cout << ::myNum << '\n'; //global
}
```

Its useful because functions can’t see inside of other functions.