Function Declarations

In this section, we will understand the purpose of functions in C++.

WE'LL COVER THE FOLLOWING

- Definition
- Rules of declaration
- Alternative function syntax
- Default arguments

Definition

> A **function** is a set of statements and operations. Like variables, functions have names. When a function is invoked or *called*, the sequence of statements inside it is executed.

Rules of declaration

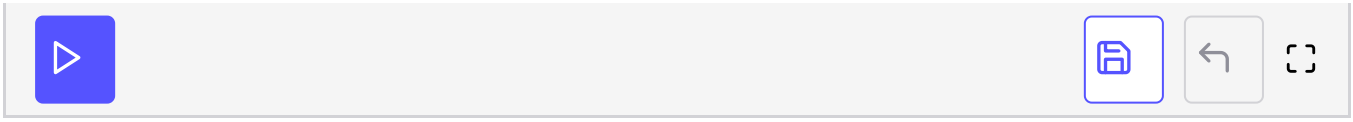
- The name of the function should hold some meaning as to what it actually does.
- A function declaration consists of a `return` type, a name, and a set of parameters (optional).
- The **signature** of a function is its name and parameters.
- A function can accept values, pointers, or references.
- Functions can be static, and their return values can be `const` or `volatile`.

Below, we can see an example of the `swap` function:

```
#include <iostream>

void swap(int& fir, int& sec){
  int tmp(fir);
  fir = sec;
  sec = tmp;
}

int main() {
  int a{2011};
  int b{2014};
  std::cout << "a: " << a <<", b: " << b << std::endl;
  swap(a, b); //  Function call
  std::cout << "After swap" << std::endl;
  std::cout << "a: " << a <<", b: " << b << std::endl;
}
```



> 💡 A function makes a copy of its arguments. So by default, the original variables are not altered outside the function scope. In order to modify them, the function must take them by reference, as is done in line 3 above.

Alternative function syntax

With the introduction of `auto`, there came a new way to write functions using the arrow syntax:

```
auto func( para ) -> returntype { functionbody }
auto sum(int a, int b) -> int { return a + b;}

int sum(int a, int b) { return a + b;} // Old syntax
```

In C++11, it is compulsory to use this syntax when the auto keyword is used in the function declaration.

As we'll see in the future, the new syntax allows us to make **lambda functions**.

Default arguments

Function parameters allow us to pass arguments to the function. These arguments can be used to perform operations within the function's body

arguments can be used to perform operations within the function's body.

For the purpose of convenience, C++ allows us to specify default arguments for a function. A parameter's default argument is used when we don't specify its value in the function call ourselves.

Each default argument is defined in the function's signature using the `=` operator.

```
bool isTempOK(const int t, const int low = 20, const int high = 50);
```

Here, `low` has a default value of `20`, whereas `high` has the default value of `50`. Consider the function call below:

```
isTempOK(100, 21);
```

`100` will be assigned to `t` and `21` will replace the default value of `low`. Since we have not specified the value for `high`, `50` will be used. This shows us that in a function call, values are assigned from **left to right**.

The parameters that have default arguments must be placed at the end. In other words, they must be on the right end of the list of parameters.

Let's look at a more comprehensive example in the next lesson.