**Functions**

1. **Built-in Functions**

#include <iostream> 🡪 cin cout

#include <cmath> 🡪 sqrt pow fabs(double) ceil floor

#include <string> 🡪 string

#include <cstdlib> 🡪 abs-(int) labs-long srand rand

#include <ctime> 🡪 time(NULL) for random number generator

Function declaration and definition is similar to C (must **declare** before int main(). Can **define** right when declared, or after int main()).

1. **Return**

A function can return the following data types: void, double, int, bool, char, std::string, std::vector<type>

1. **Inline Functions**

Using inline advises the compiler to insert the function’s body here the function call is, which sometimes helps with execution speed (and sometimes hinders speed) 🡪 should test while using

“inline functions” may also refer to member functions of a class.

Note that you should ALWAYS add the inline keyword if you are inlining functions in a header (unless you are dealing with member functions, which are automatically inlined for you).

1. **Default value**

**Must always define default values in function declarations (in header file)**

1. **Function Overload**

If you want a function to accept an argument that can be either int or double; some function parameters to be optional 🡪 Function overloading

In the process of function overloading, you can *give multiple C++ functions the same name*. However, at least one of these conditions are true:

* Each has different type parameters.
* Each has a different number of parameters.

🡪 We can change the way a function behaves based on the arguments passed.

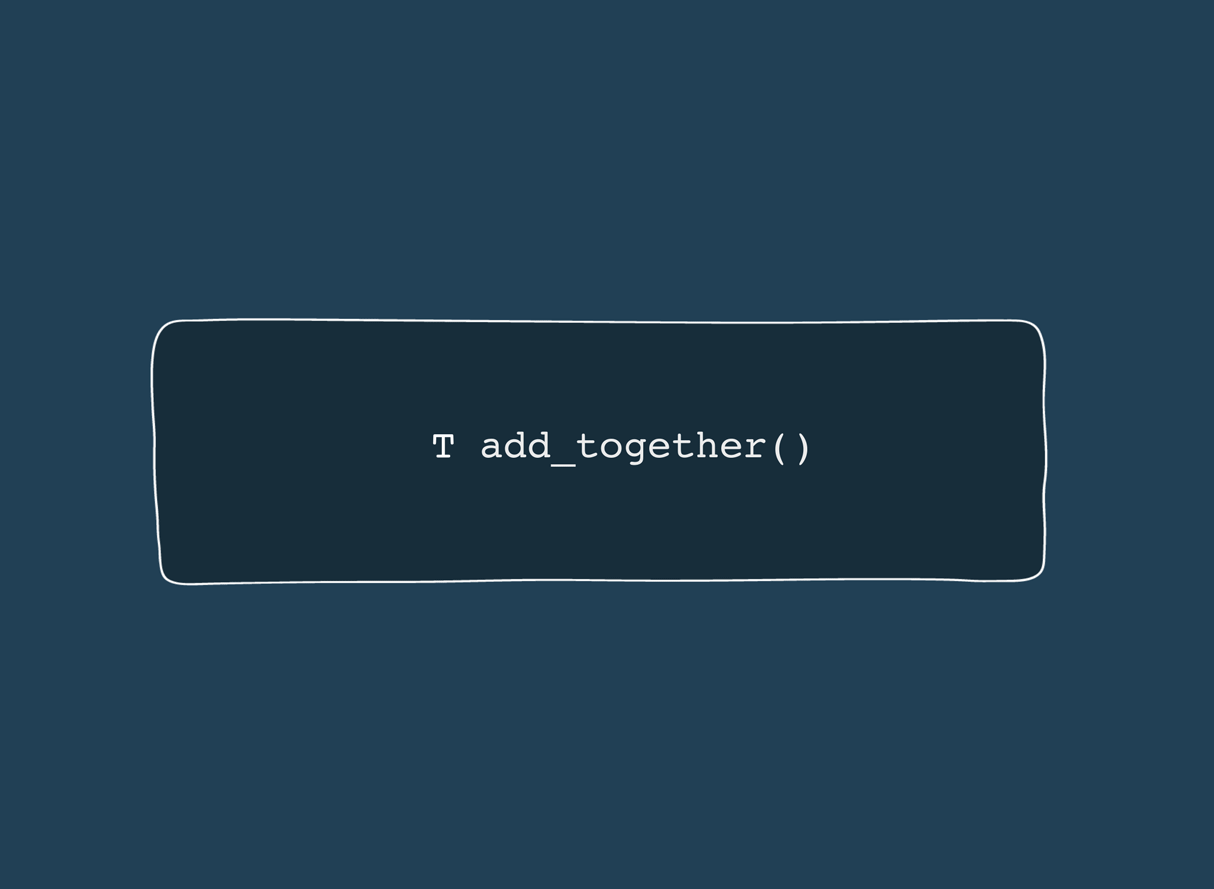
void print\_input\_twice (int num) { std::cout << num\*2; }

void print\_input\_twice (char c) { std::cout << c+c; }

1. **Local Scope**

Note that the using keyword, also has scope like declaring variables. If we use using inside a function, we would also be using the namespace inside that function only.

1. **Function Templates \*\***



If the function body is basically the same for different data types, we shouldn’t use function overload, but function template instead.

A template is a C++ tool that allows programmers to add data types as parameters.

*(In fact, std::string and std::vector are both template-based types)*.

Unlike regular functions, templates are entirely created in the header files.

Templates let us choose the type implementation right when you call the function. The type we choose may apply to the return type, a parameter type, or both.

template <typename T>

void print\_cat\_ears(T item) {

// Function body

}

Then, we can call the function for any type of parameter passed, as long as it works with the function body.

**Note:** Using templates will slow down the program’s compile time, but speed up the execution time.