Inline function

Md. Sharif Hossen
Assistant Professor, Dept. of ICT, Comilla University,

Why inline functions are used?

When the **program** executes the function call instruction, the **CPU stores** the memory address of the instruction following the function call, copies the arguments of the function on the stack and finally transfers control to the specified function. The **CPU** then executes the function code, stores the function return value in a predefined memory location/register and returns control to the calling function.

This can become **overhead** if the execution time of function is less than the switching time from the caller function to called function (callee). For functions that are large and/or perform complex tasks, the **overhead of the function call is usually insignificant** compared to the amount of time the function takes to run.

Md. Sharif Hossen, Assistant Professor, Dept. of ICT, Comilla University, Bangladesh

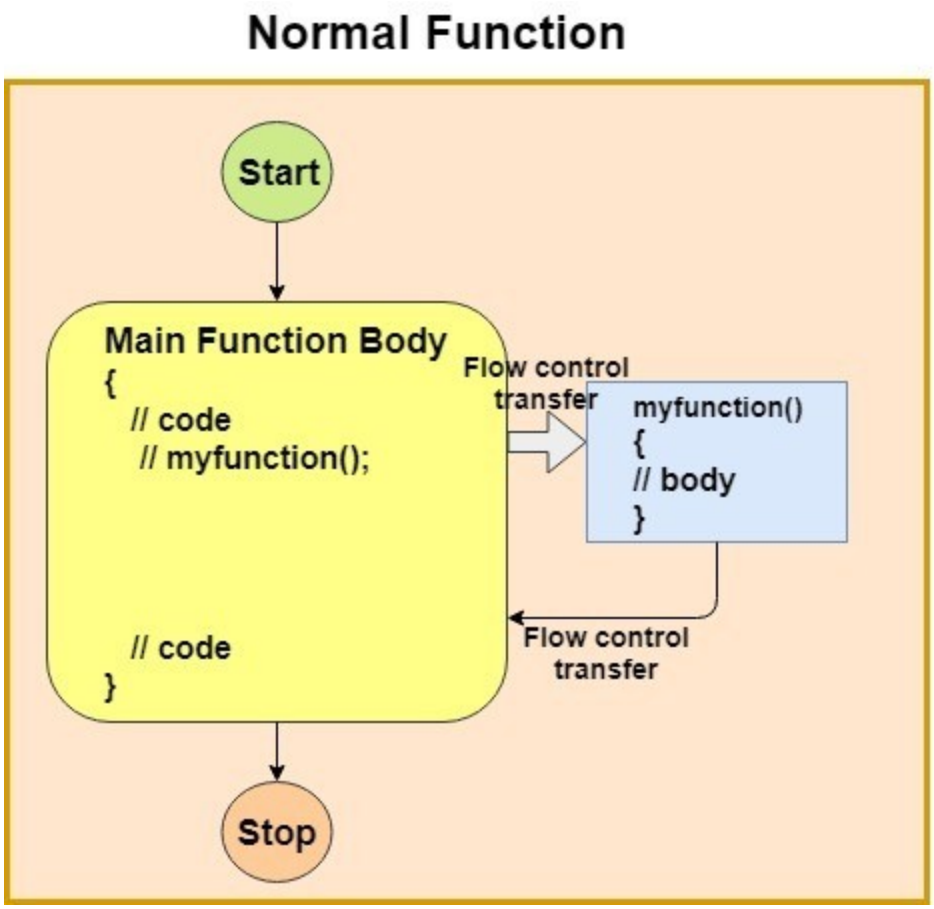
Why inline functions are used?

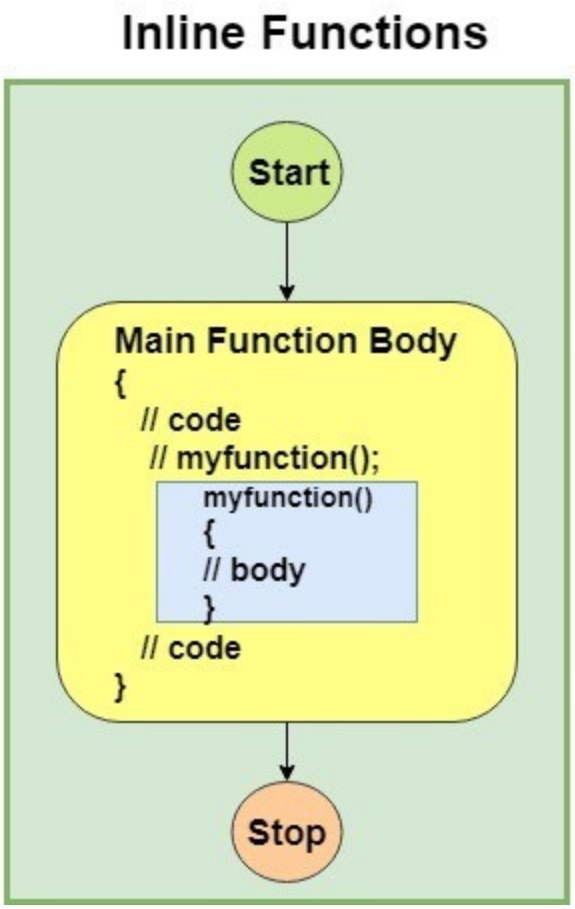
However, for small, commonly-used functions, the **time** needed to make the function call is often a lot **more than** the time needed to actually execute the function's code. This **overhead occurs for small functions** because execution time of small function is less than the switching time. C++ provides an **inline functions** to reduce the function call overhead.

What is the purpose of inline function?

Inline function is a function that is expanded in line when it is called. When the inline function is called whole code of the inline function gets inserted or substituted at the point of inline function call. This substitution is performed by the C++ compiler at compile time. Inline function may increase **efficiency if it is small**.

Normal and inline function





Normal and inline function

In a typical scenario , whenever a **function is called**, the program flow control is shifted to that function till it completes the execution and then returns back to where the function was called. This **context switching mechanism** many times causes an additional overhead leading to inefficiency. This overhead issue can be **resolved by inlining** the function (making the function inline).

Inline functions are actual functions, which are **copied everywhere during compilation**, like preprocessor macro, so the overhead of function calling is reduced. If a function is inline, the compiler places a copy of the code of that function at each point where the function is called at compile time.

Inlining is not possible

Remember, inlining is only a request to the compiler, not a command. Compiler can ignore the request for inlining. Compiler may not perform inlining in such circumstances like:

1) If a function contains a loop. (for, while, do-while)
2) If a function contains static variables.
3) If a function is recursive.
4) If a function return type is other than void, and the return statement doesn't exist in function body.
5) If a function contains switch or goto statement.

Syntax of inline function

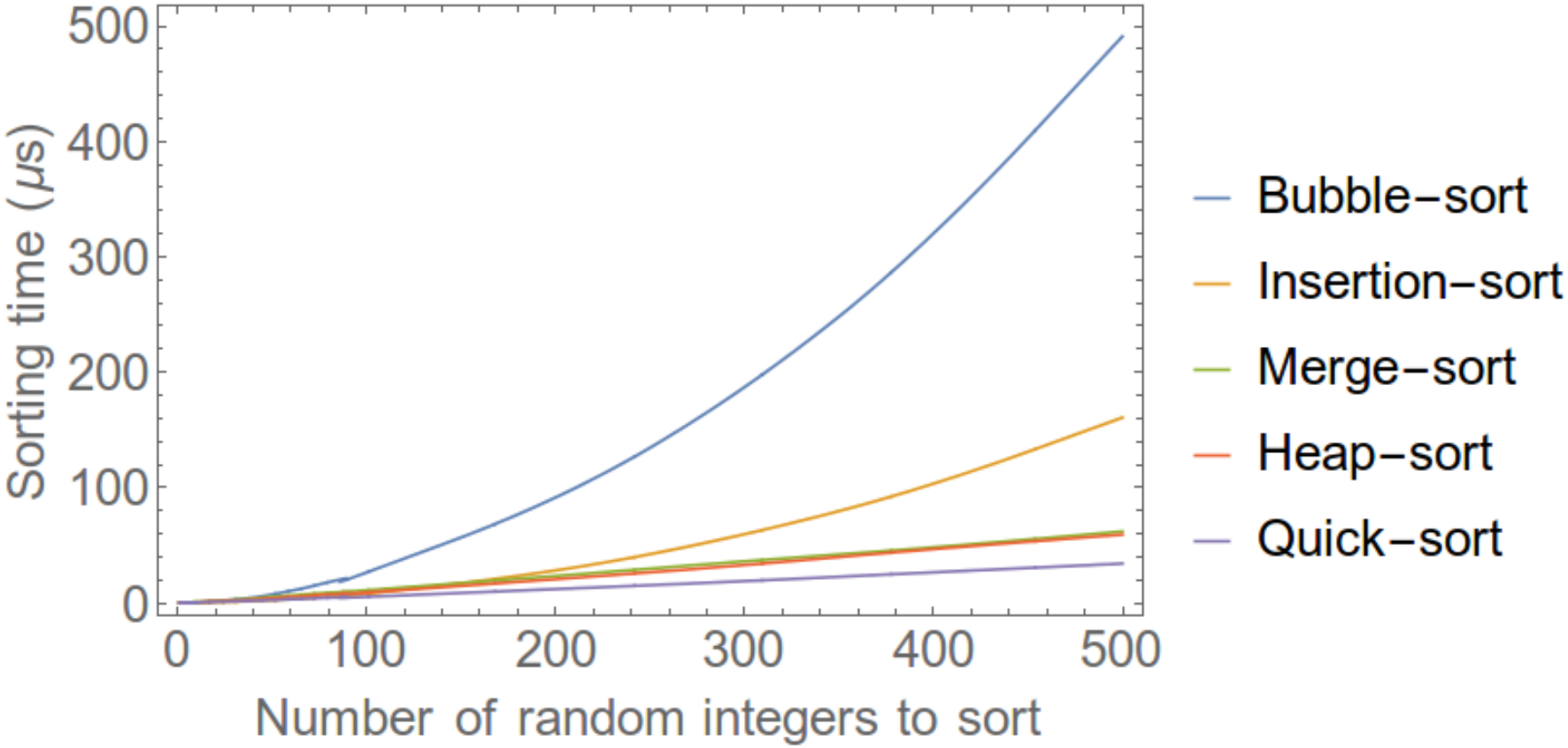
The syntax for defining the function inline is:

```
inline return-type function-name(parameters)
{
   // function code
}
```

Example

```
#include <iostream>
using namespace std;
inline int cube(int s)
{
    return s*s*s;
}
int main()
{
    cout << "The cube of 3 is: " << cube(3) << "\n";
    return 0;
} //Output: The cube of 3 is: 27
```

Sorting algorithms



Sorting algorithms