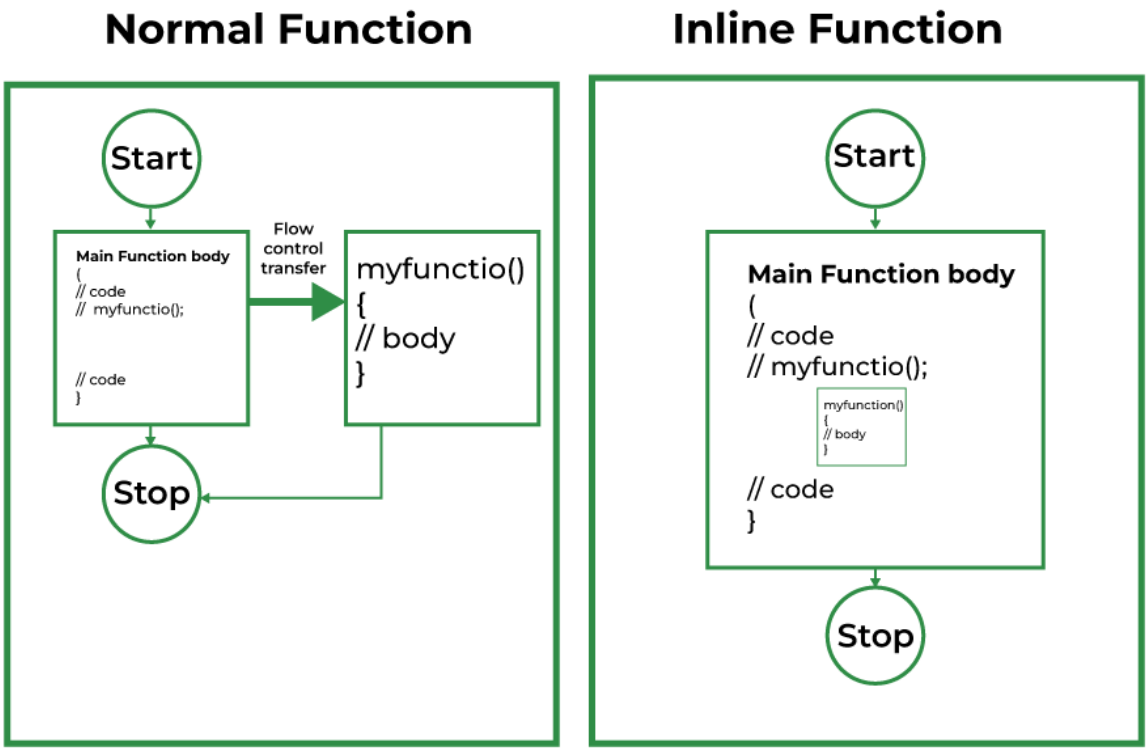
Inline Functions in C++

Difficulty Level : Medium • Last Updated : 05 Mar, 2023

C++ provides inline functions to reduce the function call overhead. An 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 the inline function call. This substitution is performed by the C++ compiler at compile time. An inline function may increase efficiency if it is small.

Syntax:

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



Remember, inlining is only a request to the compiler, not a command. The compiler can ignore the request for inlining.

The compiler may not perform inlining in such circumstances as:

1. If a function contains a loop. (*for, while and 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 a function body.
5. If a function contains a switch or goto statement.

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 the 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. 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 the execution time of a small function is less than the switching time.

Inline functions Advantages:

1. Function call overhead doesn't occur.
2. It also saves the overhead of push/pop variables on the stack when a function is called.
3. It also saves the overhead of a return call from a function.
4. When you inline a function, you may enable the compiler to perform context-specific optimization on the body of the function. Such optimizations are not possible for normal function calls. Other optimizations can be obtained by considering the flows of the calling context and the called context.
5. An inline function may be useful (if it is small) for embedded systems because inline can yield less code than the function called preamble and return.

Inline function Disadvantages:

1. The added variables from the inlined function consume additional registers, After the in-lining function if the variable number which is going to use the register increases then they may create overhead on register variable resource utilization. This means that when the inline function body is substituted at the point of the function call, the total number of variables used by the function also gets inserted. So the number of registers going to be used for the variables will also get increased. So if after function inlining variable numbers increase drastically then it would surely cause overhead on register utilization.
2. If you use too many inline functions then the size of the binary executable file will be large, because of the duplication of the same code.
3. Too much inlining can also reduce your instruction cache hit rate, thus reducing the speed of instruction fetch from that of cache memory to that of primary memory.

4. The inline function may increase compile time overhead if someone changes the code inside the inline function then all the calling location has to be recompiled because the compiler would be required to replace all the code once again to reflect the changes, otherwise it will continue with old functionality.
5. Inline functions may not be useful for many embedded systems. Because in embedded systems code size is more important than speed.
6. Inline functions might cause thrashing because inlining might increase the size of the binary executable file. Thrashing in memory causes the performance of the computer to degrade. The following program demonstrates the use of the inline function.

Example:

C++

```
#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
```

Inline function and classes

It is also possible to define the inline function inside the class. In fact, all the functions defined inside the class are implicitly inline. Thus, all the restrictions of inline functions are also applied here. If you need to explicitly declare an inline function in the class then just declare the function inside the class and define it outside the class using the inline keyword.

Syntax:

```
class S
{
public:
    inline int square(int s) // redundant use of inline
    {
        // this function is automatically inline
        // function body
```

```
    }
};
```

The above style is considered a bad programming style. The best programming style is to just write the prototype of the function inside the class and specify it as an inline in the function definition.

For Example:

```
class S
{
public:
    int square(int s); // declare the function
};

inline int S::square(int s) // use inline prefix
{
}
```

Example:

C++

```
// C++ Program to demonstrate inline functions and classes
#include <iostream>

using namespace std;

class operation {
    int a, b, add, sub, mul;
    float div;

public:
    void get();
    void sum();
    void difference();
    void product();
    void division();
};
inline void operation ::get()
{
    cout << "Enter first value:";
    cin >> a;
    cout << "Enter second value:";
    cin >> b;
}

inline void operation ::sum()
{
```

```
    add = a + b;
    cout << "Addition of two numbers: " << a + b << "\n";
}

inline void operation ::difference()
{
    sub = a - b;
    cout << "Difference of two numbers: " << a - b << "\n";
}

inline void operation ::product()
{
    mul = a * b;
    cout << "Product of two numbers: " << a * b << "\n";
}

inline void operation ::division()
{
    div = a / b;
    cout << "Division of two numbers: " << a / b << "\n";
}

int main()
{
    cout << "Program using inline function\n";
    operation s;
    s.get();
    s.sum();
    s.difference();
    s.product();
    s.division();
    return 0;
}
```

Output:

```
Enter first value: 45
Enter second value: 15
Addition of two numbers: 60
Difference of two numbers: 30
Product of two numbers: 675
Division of two numbers: 3
```

What is wrong with the macro?

Readers familiar with the C language know that the C language uses macro. The preprocessor replaces all macro calls directly within the macro code. It is recommended to always use the inline function instead of the macro. According to Dr. Bjarne Stroustrup, the creator of C++ macros are almost never necessary in C++ and they are error-prone. There

are some problems with the use of macros in C++. Macro cannot access private members of the class. Macros look like function calls but they are actually not.

Example:

C++

```
// C++ Program to demonstrate working of macro
#include <iostream>
using namespace std;
class S {
    int m;

public:
    // error
#define MAC(S::m)
};
```

Output:

```
Error: "::" may not appear in macro parameter list
 #define MAC(S::m)
```

C++ compiler checks the argument types of inline functions and necessary conversions are performed correctly. The preprocessor macro is not capable of doing this. One other thing is that the macros are managed by the preprocessor and inline functions are managed by the C++ compiler. Remember: It is true that all the functions defined inside the class are implicitly inline and the C++ compiler will perform inline calls of these functions, but the C++ compiler cannot perform inline if the function is virtual. The reason is called to a virtual function is resolved at runtime instead of compile-time. Virtual means waiting until runtime and inline means during compilation, if the compiler doesn't know which function will be called, how it can perform inlining? One other thing to remember is that it is only useful to make the function inline if the time spent during a function call is more compared to the function body execution time.

An example where the inline function has no effect at all:

```
inline void show()
{
    cout << "value of S = " << S << endl;
}
```

The above function relatively takes a long time to execute. In general, a function that performs an input-output (I/O) operation shouldn't be defined as inline because it spends a

considerable amount of time. Technically inlining of the show() function is of limited value because the amount of time the I/O statement will take far exceeds the overhead of a function call. Depending upon the compiler you are using the compiler may show you a warning if the function is not expanded inline.

Programming languages like Java & C# don't support inline functions. But in Java, the compiler can perform inlining when the small final method is called because final methods can't be overridden by subclasses, and the call to a final method is resolved at compile time.

In C# JIT compiler can also optimize code by inlining small function calls (like replacing the body of a small function when it is called in a loop). The last thing to keep in mind is that inline functions are a valuable feature of C++. Appropriate use of inline functions can provide performance enhancement but if inline functions are used arbitrarily then they can't provide better results. In other words, don't expect a better performance of the program. Don't make every function inline. It is better to keep inline functions as small as possible.

This article is contributed by **Meet Pravasi**. Please write comments if you find anything incorrect, or if you want to share more information about the topic discussed above.

338

Related Articles

1. Mathematical Functions in Python | Set 1 (Numeric Functions)
2. Mathematical Functions in Python | Set 2 (Logarithmic and Power Functions)
3. Difference between Inline and Macro in C++
4. Difference Between Inline and Normal Function in C++
5. Difference between virtual function and inline function in C++
6. C++ Inline Namespaces and Usage of the "using" Directive Inside Namespaces
7. Can Static Functions Be Virtual in C++?
8. Virtual Functions in Derived Classes in C++
9. Functions that cannot be overloaded in C++