

## Lecture 10

### Inline Functions in C++

Inline function is one of the important feature of C++. So, let's first understand why inline functions are used and what is the purpose of inline function?

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. 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. 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.

The syntax for defining the function inline is:

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

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.

## **Inline functions provide following advantages:**

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

## **Inline function disadvantages:**

- 1) The added variables from the inlined function consumes additional registers, After in-lining function if variables number which are going to use register increases than they may create overhead on register variable resource utilization. This means that when inline function body is substituted at the point of function call, total number of variables used by the function also gets inserted. So the number of register 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 an 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 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) 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 compiler would require 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 size of the binary executable file. Thrashing in memory causes performance of computer to degrade.

The following program demonstrates the use of use of inline function.

```
#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 inline function in the class then just declare the function inside the class and define it outside the class using inline keyword.

#### For example:

```

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

```

The above style is considered as a bad programming style. The best programming style is to just write the prototype of 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
{

}
```

The following program demonstrates this concept:

```
#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