Inline Functions in C++

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.

```
Basically, in case of normal functions, the program control
jumps to the place where it is
defined. But in case of inline functions, it can be underst
ood that the entire function
definition is brought to the place where it is called.
```

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
}
```

```
#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
```

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
```