Inline Function

Inline function is a C++ enhancement designed to speed up program execution. This is one of the features that separate C++ from its original C heritage. The primary distinction between inline function and normal functions is not in the way we code them, but in the way the compiler incorporates them into program.

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.

Inline mechanism increases execution performance in terms of speed. The overhead of repetitive function calls and returning values are removed. On the other hand, the program using inline functions needs more memory space since the inline functions are copied at every point where the function is invoked.

Some Important points about Inline Functions

1. We must keep inline functions small, small inline functions have better efficiency.
2. Inline functions do increase efficiency, but we should not make all the functions inline. Because if we make large functions inline, it may lead to code bloat, and might affect the speed too.
3. Hence, it is advised to define large functions outside the class definition using scope resolution :: operator, because if we define such functions inside class definition, then they become inline automatically.
4. Inline functions are kept in the Symbol Table by the compiler, and all the call for such functions is taken care at compile time.

Any change to an inline function could require all clients of the function to be recompiled because compiler would need to replace all the code once again otherwise it will continue with old functionality.

To inline a function, place the keyword inline before the function name and define the function before any calls are made to the function. The compiler can ignore the inline qualifier in case defined function is more than a line.

Following are some situations where inline function may not work:

1. The function should not be recursive.
2. Function should not contain static variables.
3. Function containing control structure such as switch, if, loop etc.
4. The function main( ) cannot work as inline.

Program 4.0 gives an example C++ program. This program shows how to define and use an inline function in your program.

Program 4.0: A program to illustrate inline functions.

```
#include<iostream.h>
#include<string.h>
void main( )
{
```

```
        clrscr( );
        int cube(int);
        int a, b, c=5;
        a=cube(3);
        b=cube(c);
        cout<< "The cube of 3 is: "<<a;
        cout<< "The cube of 5 is: "<<b;
        getch( );
}
inline int cube(int x)
{
        Return x*x*x;
}
```

Output:
The cube of 3 is: 27
The cube of 5 is: 125

Inline functions 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.