Inline Functions in C++

Function Overhead

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 functions code. This overhead occurs for small functions because execution time of small function is less than the switching time.

The use of inline functions

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.

1 Syntax for Inline Function

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

NOTE: Remember, inlining is only a request to the compiler, not a com-

mand. 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 doesnt exist in function body.
5. If a function contains switch or goto statement.

Disadvantages of Inline Functions

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.

Inline function and classes

- All the functions defined inside the class are implicitly inline.
- If you dont want to make the function inline,just declare the function inside the class and define it outside the class.

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

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

}
```

- 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 infront of the return type.

What is wrong with macro?

- Macro cannot access private members of class.

```
#include <iostream>
using namespace std;
class S
{
```

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

- C++ compiler checks the argument types of inline functions and necessary conversions are performed correctly. Preprocessor macro is not capable for doing this.
- One other thing is that the macros are managed by preprocessor and inline functions are managed by C++ compiler.