**Inline Functions**

All the member functions defined inside the class definition are by default declared as Inline. Let us have some background knowledge about these functions.

You must remember Preprocessors from C language. Inline functions in C++ do the same thing what Macros do in C. Preprocessors were not used in C++ because they had some drawbacks.

**Drawbacks of Macro**

In Macro, we define certain variable with its value at the beginning of the program, and everywhere inside the program where we use that variable, its replaced by its value on Compilation.

**1) Problem with spacing**

Let us see this problem using an example,

#define G (y) (y+1)

Here we have defined a Macro with name G(y), which is to be replaced by its value, that is (y+1) during compilation. But, what actually happens when we call G(y),

G(1) //Macro will replace it

the preprocessor will expand it like,

(y) (y+1) (1)

You must be thinking why this happened, this happened because of the spacing in Macro definition. Hence big functions with several expressions can never be used with macro, so Inline functions were introduced in C++.

**2) Complex Argument Problem**

In some cases such Macro expressions work fine for certain arguments but when we use complex arguments problems start arising.

#define MAX(x,y) x>y?1:0

Now if we use the expression,

if(MAX(a&0x0f,0x0f)) // Complex Argument

Macro will Expand to,

if( a&0x0f > 0x0f ? 1:0)

Here precedence of operators will lead to problem, because precedence of & is lower than that of >, so the macro evaluation will surprise you. This problem can be solved though using parenthesis, but still for bigger expressions problem will arise.

**3) No way to access Private Members of Class**

With Macros, in C++ you can never access private variables, so you will have to make those members public, which will expose the implementation.

class Y

{

 int x;

 public :

 #define VAL(Y::x) // Its an Error

}