### Inline Functions in C++

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

}

### Inline Functions

Inline functions are actual functions, which are copied everywhere during compilation, like preprocessor macro, so the overhead of function calling is reduced. All the functions defined inside class definition are by default inline, but you can also make any non-class function inline by using keyword **inline** with them.

For an inline function, declaration and definition must be done together. For example,

inline void fun(int a)

{

return a++;

}

#### 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 adviced 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.

#### Access Functions

We have already studied this in topic Accessing Private Data variables inside class. We use access functions, which are inline to do so.

class Auto

{

int i;

public:

int getdata()

{

return i;

}

void setdata(int x)

{

i=x;

}

};

Here getdata() and setdata() are inline functions, and are made to access the private data members of the class Auto. getdata(), in this case is called **Accessor** function and setdata() is a **Mutator** function.

There can be overlaoded Accessor and Mutator functions too. We will study overloading functions in next topic.

### Limitations of Inline Functions

1. Large Inline functions cause Cache misses and affect performance negatively.
2. Compilation overhead of copying the function body everywhere in the code on compilation, which is negligible for small programs, but it makes a difference in large code bases.
3. Also, if we require address of the function in program, compiler cannot perform inlining on such functions. Because for providing address to a function, compiler will have to allocate storage to it. But inline functions doesn't get storage, they are kept in Symbol table.

#### Forward References

All the inline functions are evaluated by the compiler, at the end of class declaration.

class ForwardReference

{

int i;

public:

int f() {return g()+10;} // call to undeclared function

int g() {return i;}

};

int main()

{

ForwardReference fr;

fr.f();

}

You must be thinking that this will lead to compile time error,but in this case it will work, because no inline function in a class is evaluated until the closing braces of class declaration.