INLINE FUNCTIONS

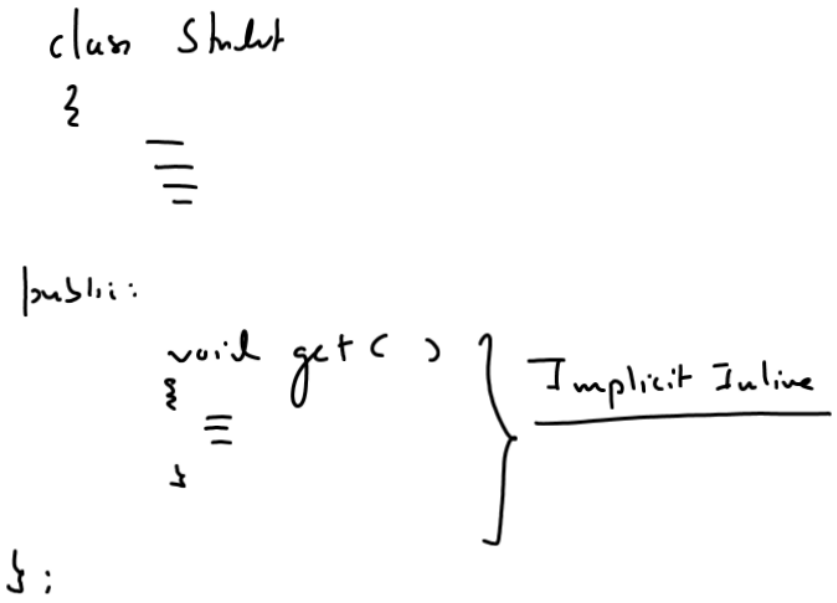
```
inline int square(int n)
{
    return n * n;
}

int main()
{
    int a = 10, b = 20, c;
    c = square(a);
    cout << "Sq of " << a << " is " << c << endl;
    c = square(b);
    cout << "Sq of " << b << " is " << c;
    return 0;
}
```

```
int square(int);

int main()
{
    ...
    c = sqr(a);
    ...
}

inline int square(int n)
{
    ...
}
```



Inline functions
==========
In C++, inline functions are those functions which are prefixed with the keyword inline in their definition. For ex:

```
inline int square(int n)
{
  return n*n;
}
```

The square function defined above is an inline function. When we declare a function as inline then the C++ compiler automatically replaces every call to that function with its definition. In other words we can say that an inline function gets its body pasted in place of its call.

By declaring a function as inline we can reduce compiler's overhead associated with function call like prototype checking, argument pasing and most importantly stack updations. Thus making a functio inline increases the execution speed and reduces the execution time. In short we can say making a function inline improves the efficiency of the code.

But we must remember 3 impt points before declaring a function as inline.

1. The body of an inline function must be short and small.

2. It should not contain any complecated statement like for, while, do-while etc.

3. The body of an inline function must appear before its call in the source code i.e. the compiler must become aware that function is inline before the call to the function.

If any of the above rules are broken then the compiler simply ignores the keyword inline and handles the function in a normal way.

So we can say that making a function inline doesn't compulsorily mean that compiler will accept our request.

Moreover w.r.t a class we can have 2 types of inline function

1. implicit inline
2. explicit inline

1. implicit inline: These are those member functions which are define within the class body and even if we have not used the keyword inline still the compiler will handle the function as inline function. Moreover all the 3 member functions which are automatically generated by the compiler for us (default constructor, default copy constructor and default destructor) are implicit inline function.

2. explicit inline: These are those functions which are defined outside the class prefixed with the keyword inline.

However the rules mentioned above are aplicable here also and if these rules are broken then the compiler will again ignore the keyword inline and handle the function in a normal way.

So, finally we can say that making a function inline is a request made by the programmer to the compiler which can either we accepted or rejected depending on the circumstances.

```
#include <iostream>
using namespace std;
class Emp
{
    int age;
    char name[20];
public:
    void get()
    {
        cout<<"Enter age and name:";
        cin>>age>>name;
    }
    void show();
};
inline void Emp::show()
{
    cout<<age<<","<<name<<endl;
}
int main()
{
    Emp E;
    E.get();
    E.show();
    return 0;
}
```

Implicit Inline

Explicit Inline