What is inline function

Functions can be instructed to compiler to make them inline so that compiler can replace those function definition wherever those are being called. Compiler replaces the definition of inline functions at compile time instead of referring function definition at runtime.

```
#include <iostream>
using namespace std;
inline void hello()
{   cout<<"hello";}
int main()
{
  hello(); //Call it like a normal function...
}
```

Why to use

- **Pros** :-
  1. It speeds up your program by avoiding function calling overhead.
  2. It save overhead of variables push/pop on the stack, when function calling happens.
  3. It save overhead of return call from a function.
  4. It increases locality of reference by utilizing instruction cache.
  5. By marking it as inline, you can put a function definition in a header file (i.e. it can be included in multiple compilation unit, without the linker complaining)

- **Cons** :-
  1. It increases the executable size due to code expansion.
  2. C++ inlining is resolved at compile time. Which means if you change the code of the inlined function, you would need to recompile all the code using it to make sure it will be updated
  3. When used in a header, it makes your header file larger with information which users don't care.
  4. As mentioned above it increases the executable size, which may cause thrashing in memory. More number of page fault bringing down your program performance.
  5. Sometimes not useful for example in embedded system where large executable size is not preferred at all due to memory constraints.

When to use -

Function can be made as inline as per programmer need. Some useful recommendation are mentioned below-

1. Use inline function when performance is needed.
2. Use inline function over macros.
3. Prefer to use inline keyword outside the class with the function definition to hide implementation details.

Friend class and function in C++

- **Friend Class** A friend class can access private and protected members of other class in which it is declared as friend. It is sometimes useful to allow a particular class to access private members of other class.

```
class Node
{
private:
  int key;
  Node *next;
  /* Other members of Node Class */

  friend class LinkedList; // Now class  LinkedList can
                           // access private members of Node
};
```

- **Friend Function** Like friend class, a friend function can be given special grant to access private and protected members. A friend function can be:
  a) A method of another class
  b) A global function

```
class Node
{
private:
  int key;
  Node *next;

  /* Other members of Node Class */
  friend int LinkedList::search(); // Only search() of
linkedList
                              // can access internal members
};
```

Importance

- **1)** Friends should be used only for limited purpose. too many functions or external classes are declared as friends of a class with protected or private data, it lessens the value of encapsulation of separate classes in object-oriented programming.
- **2)** Friendship is not mutual. If a class A is friend of B, then B doesn’t become friend of A automatically.
- **3)** Friendship is not inherited

Example of **friend Class**

```
#include <iostream.h>
class A {
private:
   int a;
public:
   A() { a=0; }
   friend class B;    // Friend Class
};
```

```
class B {
private:
   int b;
public:
   void showA(A& x) {
      // Since B is friend of A, it can access
      // private members of A
      std::cout << "A::a=" << x.a;
   }
};
```

```
int main() {
  A a;
  B b;
  b.showA(a);
  return 0;
}
```

Output:

```
A::a=0
```

Example of **friend function**

```
#include <iostream.h>

class B;

class A
{
public:
   void showB(B& );
};

void A::showB(B &x)
{
   // Since show() is friend of B, it can
   // access private members of B
   std::cout << "B::b = " << x.b;
}
```

```
class B
{
private:
   int b;
public:
   B()  {  b = 0; }
   friend void A::showB(B& x); // Friend
function
};
```

```
int main()
{
   A a;
   B x;
   a.showB(x);
   return 0;
}
```

Output:

```
B::b = 0
```

Example of global friend

```
#include <iostream>

class A
{
   int a;
public:
   A() {a = 0;}
   friend void showA(A&); // global friend function
};

void showA(A& x) {
   // Since showA() is a friend, it can access
   // private members of A
   std::cout << "A::a=" << x.a;
}

int main()
{
   A a;
   showA(a);
   return 0;
}
```

- Output:

```
A::a = 0
```