Inline function in C++

One of the key features of C++ is the inline function. Therefore, let's first examine the utilization of inline functions and their intended application. If make a function is inline, then the compiler replaces the function calling location with the definition of the inline function at compile time.

The compiler may reject the inlining request. The compiler may not implement inlining in situations like these:

1. If a function contains a loop. (for, while, do-while)
2. if a function has static variables.
3. Whether a function recurses.
4. If the return statement is absent from the function body and the return type of the function is not void.
5. Whether a function uses a goto or switch statement.

Syntax for an inline function:

```
inline return_type function_name(parameters)
{
        // function code?
}
```

EXAMPLE-1

```
#include <iostream>

using namespace std;

inline int add(int a, int b)

{

   return(a+b);

}

int main()
```

```
{
   cout<<"Addition of 'a' and 'b' is:"<<add(2,3);
   return 0;
}
```

EXAPMLE 2

```
#include<iostream>
using namespace std;
inline int add(int a, int b)
{
    return(a+b);
}
int main()
{
    cout<<"Addition of 'a' and 'b' is:"<<(2+3);
   return 0;
}
```

ENCAPSULATION

Encapsulation in C++ is defined as the wrapping up of data and information in a single unit. In Object Oriented Programming, Encapsulation is defined as binding together the data and the functions that manipulate them.
Consider a real-life example of encapsulation, in a company, there are different sections like the accounts section, finance section, sales section, etc. Now,

- The finance section handles all the financial transactions and keeps records of all the data related to finance.
- Similarly, the sales section handles all the sales-related activities and keeps records of all the sales.

Now there may arise a situation when for some reason an official from the finance section needs all the data about sales in a particular month.

In this case, he is not allowed to directly access the data of the sales section. He will first have to contact some other officer in the sales section and then request him to give the particular data.

This is what **Encapsulation** is. Here the data of the sales section and the employees that can manipulate them are wrapped under a single name "sales section".

Two Important property of Encapsulation

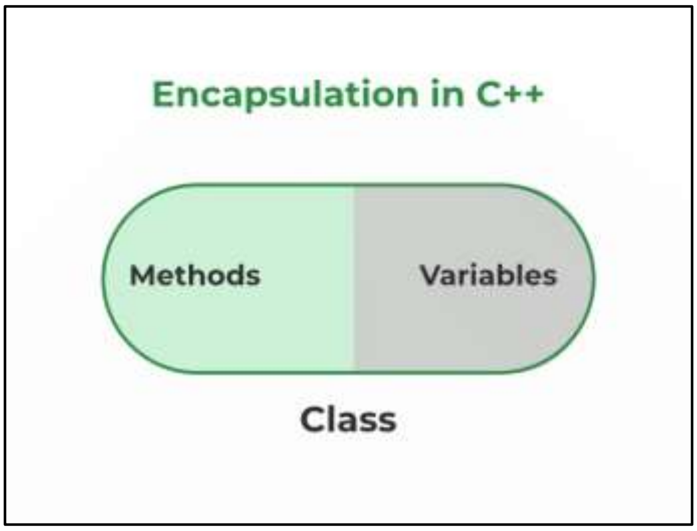
1. **Data Protection:** Encapsulation protects the internal state of an object by keeping its data members private. Access to and modification of these data members is restricted to the class's public methods, ensuring controlled and secure data manipulation.

2. **Information Hiding:** Encapsulation hides the internal implementation details of a class from external code. Only the public interface of the class is accessible, providing abstraction and simplifying the usage of the class while allowing the internal implementation to be modified without impacting external code.

Features of Encapsulation

Below are the features of encapsulation:

- We can not access any function from the class directly. We need an object to access that function that is using the member variables of that class.
- The function which we are making inside the class must use only member variables, only then it is called encapsulation.
- If we don't make a function inside the class which is using the member variable of the class then we don't call it encapsulation.
- Encapsulation improves readability, maintainability, and security by grouping data and methods together.
- It helps to control the modification of our data members.



EXAMPLE 1

```
#include <iostream>
using namespace std;


class Employee {
  private:
    int salary;


  public:
    void setSalary(int s) {
      salary = s;
    }
    int getSalary() {
      return salary;
    }
};
```

```
int main() {
  Employee myObj;
  myObj.setSalary(50000);
  cout << myObj.getSalary();
  return 0;
}
/*
The meaning of Encapsulation, is to make sure that "sensitive" data is hidden from users.
To achieve this, you must declare class variables/attributes as private (cannot be accessed from outside the class).
If you want others to read or modify the value of a private member, you can provide public get and set methods.*/
```

Encapsulation also leads to data abstraction. Using encapsulation also hides the data, as in the above example, the data of the sections like sales, finance, or accounts are hidden from any other section.

Example 2

```
#include <iostream>
#include <string>

using namespace std;

class Person {
private:
	string name;
	int age;
```

```
public:
          Person(string name, int age)
         {
          this->name = name;
          this->age = age;
          }
          void setName(string name)
         {
          this->name = name;
          }
          string getName()
           {
          return name;
          }
          void setAge(int age)
         {
          this->age = age;
          }
          int getAge()
        {
          return age;
          }
};

int main()
 {
```

```
Person person("DISHA", 30);
cout << "Name: " << person.getName() << endl;
cout << "Age: " << person.getAge() << endl;
person.setName("DISHA");
person.setAge(32);
cout << "Name: " << person.getName() << endl;
cout << "Age: " << person.getAge() << endl;
}
```

EXAMPLE 3

```
// C++ program to demonstrate Encapsulation
#include <iostream>
using namespace std;
class Encapsulation
{
private:
                int x;
public:
        void set(int a)
         {
             x = a;
          }
       int get()
      {
        return x;
      }
};
int main()
{
        Encapsulation e;
        e.set(5);
        cout << e.get();
}
```