
FACE

Encapsulation

- Binding together the data and the functions that manipulates them
- Also leads to data abstraction or hiding
- Can be implemented using class and access modifiers
- For example

```
class Car
{
      int mileage;
      int max_speed;
      public:
            void display();
}
```

FACE

Abstraction

- Abstraction - hiding irrelevant details from the user.

- Access specifiers are the main pillar of implementing abstraction.

FACE

Access Specifiers

- All members of a class can be accessed:
- Within the class - No restriction.
- From outside the class - ?

FACE

Access Specifiers

- Defines how a member's variables and member's functions of a class can be accessed from outside the class
- Used to set boundaries for availability of members of class
- Are followed by a colon
- Types:
    i. Public
    ii. Private
    iii. Protected

FACE

1. Public

- All the class members declared under public will be available to everyone
- Public members can be accessed by other classes too
- Can be accessed from anywhere in the program using the direct member access operator (.) with the object of that class

FACE

```
1  #include<iostream>
2  using namespace std;
3  class Circle
4  {
5      public:
6      double rad;
7      public:
8      double area()
9      {
10         return 3.14*rad*rad;
11     }
12 };
13 int main()
14 {
15     Circle obj;
16     obj.rad=1;
17     cout<<"Area: "<<obj.area();
18     return 0;
19 }
20
21
22
```

OUTPUT

```
Area: 3.14
```

2. Private

- Any object or function outside the class cannot access the private members

- Can be accessed only by the functions inside the class and friend functions.

- By default all the members of a class would be private.

FACE

```
1  #include<iostream>
2  using namespace std;
3  class Circle
4  {
5      public:
6      double rad;
7      public:
8      double area()
9      {
10         rad=1;
11         return 3.14*rad*rad;
12     }
13 };
14 int main()
15 {
16     Circle obj;
17     cout<<"Area: "<<obj.area();
18     return 0;
19 }
20
21
22
```

OUTPUT

```
Area: 3.14
```

3. Protected

- A protected member variable or function is very similar to a private member

- But they can be accessed by any subclass (derived class) of that class.

FACE

```
1  #include <iostream>
2  using namespace std;
3  class Example1
4  {
5        protected:
6              int a;
7  };
8  class Example2:public Example1
9  {
10       public:
11             void init_a(int a)
12             {
13                   this->a=a;
14             }
15             void print_a()
16             {
17                   cout<<"a: "<<a<<endl;
18             }
19 };
20
21
22
```

```
1   int main()
2   {
3         Example2 Ex;
4         Ex.init_a(100);
5         Ex.print_a();
6         return 0;
7   }
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
```

OUTPUT

```
a: 100
```

```
#include<iostream>
using namespace std;
class Sample
{
    int x;
};

int main()
{
    Sample obj;
    obj.x = 1;
    cout << obj.x;
    return 0;
}
```

By default members of a class are Private, so output will be error.

```
#include<iostream>
using namespace std;
struct Sample
{
    int x;
};

int main()
{
    Sample obj;
    obj.x = 1;
    cout << obj.x;
    return 0;
}
```

By default members of struct are public, output will be 1.

Friend Function

- One of the important concepts of OOP is data hiding, i.e., a nonmember function cannot access an object's private or protected data.

- But, sometimes this restriction may force programmer to write long and complex codes. So, there is mechanism built in C++ programming to access private or protected data from non-member functions.

- This is done using a friend function or/and a friend class.

FACE

Friend Function

- For accessing the data, the **declaration of a friend function should be made inside the body of the class** (can be anywhere inside class either in private or public section) starting with **keyword friend.**

```
class  class_name
{
    ... .. ...
    friend  return_type  function_name(arguments);
    ... .. ...
}
```

FACE

Friend Function

- Now, you can define the friend function as a normal function to access the data of the class. No friend keyword is used in the definition.

```
class  class_name
{
     ... .. ...
     friend   return_type  function_name(arguments);
     ... .. ...
}
return_type   functionName(arguments)
{
     ... .. ...
     ... .. ...
}
```

FACE

```
1  #include<iostream>
2  using namespace std;
3  class Distance
4  {
5        private:
6           int meter;
7        public:
8           Distance():meter(0)
9           {   }
10          friend int addFive(Distance);
11 };
12 int addFive(Distance d)
13 {
14       d.meter += 5;
15       return d.meter;
16 }
17 int main()
18 {
19       Distance D;
20       cout << "Distance:" << addFive(D);
21       return 0;
22 }
```

comment/pseudo code/output

Friend Function declaration

Friend Function definition, accessing private data member of the class

```
1  #include<iostream>
2  using namespace std;
3  class Distance
4  {
5        private:
6           int meter;
7        public:
8           Distance():meter(0)
9           {   }
10          friend int addFive(Distance);
11 };
12 int addFive(Distance d)
13 {
14       d.meter += 5;
15       return d.meter;
16 }
17 int main()
18 {
19       Distance D;
20       cout << "Distance:" << addFive(D);
21       return 0;
22 }
```

comment/pseudo code/output

This example is just to understand friend function

Friend Function

The actual purpose is :

To operate on objects of two different classes.

Operating on two objects of different classes without using the friend function is possible but the program will be long, complex and hard to understand.

FACE

```
1  #include<iostream>
2  using namespace std;
3  class B;
4  class A
5  {
6        private:
7           int numA;
8        public:
9           A(): numA(10) { }
10          friend int add(A, B);
11 };
12 class B
13 {
14       private:
15          int numb;
16       public:
17          B() : numb(20) { }
18          friend int add(A, B);
19 };
20
21
22
```

comment/pseudo code/output

A forward declaration of a class B should be made.

This is because class B is referenced within the class A using code:

```
friend int add(A, B);
```

Classes A and B have declared add() as a friend function.

```
1  #include<iostream>
2  using namespace std;
3  class B;
4  class A
5  {
6        private:
7           int numA;
8        public:
9           A(): numA(10) { }
10          friend int add(A, B);
11 };
12 class B
13 {
14       private:
15          int numb;
16       public:
17          B() : numb(20) { }
18          friend int add(A, B);
19 };
20
21
22
```

```
23 int add(A a, B b)
24 {
25       return (a.numA + b.numb);
26 }
27
28 int main()
29 {
30        A a;
31        B b;
32        cout << "Sum: " << add(a,b);
33        return 0;
34 }
35
36
37
38
39
40
41
```

OUTPUT

```
Sum: 30
```

```
1  class B;
2  class A
3  {
4        friend class B;
5        …….
6  }
7
8  class B
9  {
10       ……..
11 }
12
13
14
15
16
17
18
19
20
21
22
```

comment/pseudo code/output

Similarly, like a friend function, a class also be made a friend of another class using keyword friend.

When a class is made a friend class, all the member functions of that class becomes friend functions

```
1  class B;
2  class A
3  {
4        friend class B;
5        …….
6  }
7
8  class B
9  {
10       ……..
11 }
12
13
14
15
16
17
18
19
20
21
22
```

comment/pseudo code/output

In this program, all member functions of class B will be friend functions of class A.

Thus, any member function of class B can access the private and protected data of class A.

But, member functions of class A cannot access the data of class B

Question 1

The keywords private and public used in C++ are known as …….

A) keyword labels

B) visibility labels

C) declaration labels

D) display labels

Question 2

How access specifiers in Class helps in Abstraction?

A) They does not helps in any way

B) They allows us to show only required things to outer world

C) They help in keeping things together

D) Abstraction concept is not used in classes

Question 3

Predict the output of following C++ program

```
#include<iostream>
using namespace std;
class Face {
    int x;
};
int main()
{
  Face f;
  cout << f.x;
  return 0;
}
```

Question 3

A) 1

B) 0

C) Compiler Error

D) Runtime Error

Question 4

In a class, encapsulating an object of another class is called

A) Composition

B) Inheritance

C) Encapsulation

D) None

Question 5

Hiding the complexity is known as

A) Abstraction

B) Encapsulation

C) Data hiding

D) Composition

Question 6

Which of the following statement is correct with respect to the use of friend keyword inside a class?

A) A private data member can be declared as a friend.

B) A class may be declared as a friend.

C) An object may be declared as a friend.

D) We can use friend keyword as a class name.

Question 7

What does a class in C++ holds?

A) data

B) functions

C) both data & functions

D) arrays

FACE

Question 8

Which among the following best defines abstraction?

A) Hiding the implementation

B) Showing the important data

C) Hiding the important data

D) Hiding the implementation and showing only the features

Question 9

Encapsulation and abstraction differ as:

A) Binding and Hiding respectively

B) Hiding and Binding respectively

C) Can be used any way

D) Hiding and hiding respectively

Question 10

In terms of stream and files_____

A) Abstraction is called a stream and device is called a file

B) Abstraction is called a file and device is called a stream

C) Abstraction can be called both file and stream

D) Abstraction can’t be defined in terms of files and stream

THANK YOU

FACE