1. Introduction

In C++, **friend functions** provide a way to allow **non-member functions** to access the **private and protected members** of a class.
Normally, private members of a class cannot be accessed outside the class. However, by using the **friend keyword**, a function or another class can be given special permission.

Friend functions are an important feature of **Object-Oriented Programming (OOP)** and are mainly used when **two or more classes need to share data closely**.

---

2. Need for Friend Function

The main reason for using friend functions is to **overcome data hiding limitations** in certain situations.

Why friend functions are required:

- To access **private data** of a class
- To allow **external functions** to work with class data
- To simplify **operator overloading**
- To enable **data sharing between multiple classes**

---

3. Definition of Friend Function

A **friend function** is:

- Not a member of a class
- Declared using the keyword `friend` inside the class
- Allowed to access **private and protected members**
- Called like a **normal function**, not using objects

---

4. Syntax of Friend Function

```
class ClassName {
  private:
    int x;
  public:
    friend void show(ClassName obj);
};
```

The function definition is written outside the class:

```
void show(ClassName obj) {
  cout << obj.x;
}
```

5. Characteristics of Friend Function

- It is **not a class member**
- It does not use **this pointer**
- It can access **private and protected data**
- It is declared inside the class but defined outside
- It can be a **global function** or member of another class

6. Example of Friend Function

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

class Sample {
private:
   int num;
public:
   Sample(int n) {
      num = n;
   }
   friend void display(Sample s);
};

void display(Sample s) {
   cout << "Number: " << s.num;
}

int main() {
   Sample obj(10);
   display(obj);
   return 0;
}
```

Output

```
Number: 10
```

7. Friend Function and Encapsulation

Encapsulation means **wrapping data and functions together** and restricting direct access to data.

Friend functions:

- Break strict encapsulation
- Are used only when necessary
- Must be carefully implemented

They should be used **sparingly** to maintain data security.

8. Friend Function vs Member Function

Feature	Friend Function	Member Function
Belongs to class	No	Yes
Access private data	Yes	Yes
Uses object to call	No	Yes
Uses this pointer	No	Yes
Declared using	friend keyword	normal syntax

9. Friend Function and Multiple Classes

A friend function can be declared in **more than one class**, allowing it to access private data of multiple classes.

Example

```
class A {
   int x;
public:
   A(int a) { x = a; }
   friend void add(A, B);
};

class B {
   int y;
public:
   B(int b) { y = b; }
   friend void add(A, B);
};

void add(A obj1, B obj2) {
   cout << obj1.x + obj2.y;
}
```

10. Friend Class

A **friend class** is a class whose **all member functions** can access the private and protected members of another class.

Syntax

```
class B;

class A {
   friend class B;
};
```

---

11. Example of Friend Class

```
class A {
private:
   int x;
public:
   A() { x = 10; }
   friend class B;
};

class B {
public:
   void show(A obj) {
      cout << obj.x;
   }
};
```

---

12. Advantages of Friend Function

- Allows access to private data when required
- Simplifies **operator overloading**
- Useful for closely related classes
- Improves flexibility of program design

---

13. Disadvantages of Friend Function

- Breaks data hiding
- Reduces security
- Increases dependency between classes
- Makes code harder to maintain if overused

---

14. Friend Function and Operator Overloading

Friend functions are commonly used in **operator overloading**, especially when the left operand is not an object.

Example

```
class Sample {
   int x;
public:
   Sample(int a) { x = a; }
   friend Sample operator +(Sample, Sample);
};

Sample operator +(Sample a, Sample b) {
   return Sample(a.x + b.x);
}
```

---

15. Rules for Friend Function

1. Declared using `friend` keyword
2. Not affected by access specifiers
3. Cannot be inherited
4. Called like normal function
5. Can be declared in any section of class

---

16. Common Mistakes

- Excessive use of friend functions
- Assuming friend functions are class members
- Using friend functions when not required
- Breaking encapsulation unnecessarily

---

17. When to Use Friend Function

- When two classes share **common data**
- For **operator overloading**
- For performance optimization in some cases
- When tight coupling is acceptable

---

18. Best Practices

- Use friend functions **only when necessary**
- Prefer member functions when possible
- Keep friend functions minimal
- Document friend usage clearly

---

19. Applications of Friend Function

- Mathematical operations
- Operator overloading
- System-level programming
- Closely related class operations

---

20. Conclusion

Friend functions are a **powerful feature** of C++ that allow controlled access to private and protected members of a class.

- They improve flexibility
- Enable data sharing
- Support operator overloading

However, they should be used **carefully** to avoid breaking the principles of **encapsulation and data hiding**.