Friend function:- Friend fun is function that is declared as a friend of a class not as a member of a class instead of that it can access private & protected member of a class.

Syntax friend returntype Funname (class & ref);

Ex

```
#include <iostream>
using namespace std;
class Ankit;
class Ankush{
private:
int money=10;
friend void rohit (Ankit, Ankush);
};
Class Ankit{
int money=20; //Private member
friend void rohit (Ankit, Ankush);
};
main(){
Ankit obj1;
Cout << obj1.money;
}
```

error (declared at line 16) is inaccessible

error Not accessible

```
main(){
Ankush obj1
Cout << obj1-money;
}
```

error Inaccessible

friend will access private void rohit (Ankit

```
class Ankit;
class Ankush{
private:
int money=10;
friend void rohit (Ankit Ankush);
};
Class Ankit {
private:
int money=20;
friend void rohit (Ankit /Ankush);
};
```

// Rohit will access ∵ friend

```
Void rohit (Ankit r1, Ankur r2) {
  Cout << "Sum " << r1.money + r2.money ;  // r1 r2 → Pefrom
  }
main() {
Ankur obj1; Ankit obj2; // Make object for Ankit & Ankur
rohit (obj2, obj1); //rohit take obj of both)
  }
```

first Ankit's object After that Ankur object.

Sum = 30

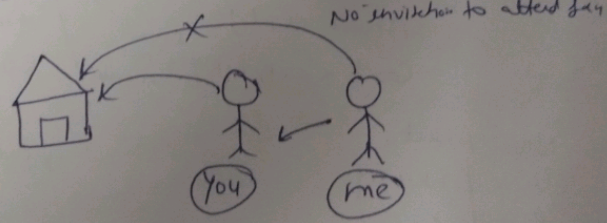
Rohit access the Both class's Private member b/c he is friend of both.

Another is friend class :-

Friend class is a class that granted accessibility of Private and protected member of another class.

Syntax :-

```
class class-name(A)
{ private:
  public:
  friend class . class-name ; (B)
  };
```




```
#include <iostream>
using namespace std;
 class A {
    int a=10, b=20;  // Private member
  };
 main() {
    A obj;
  Cout << obj.a;
 }
```

// Error - Inaccessible b/c Private

Operator Overloading :-

To assign more than one operations on an same Operators known as operator overloading.

To achieve operator overloading we have to write a special function known as operator( ) fun.

Syntax

```
return type operator op (arg list)
{
   body;
}
```

keyword → operator, op → operator, (arg list) → arg. Pass

Uni / bin.

You can write operator ( ) in 2 ways —

1. class function (write code in class)
2. friend function (simply declare it in class)

Following is the list of operators that can't be overloaded.

1) .
2) : :
3) ? :
4) size of ( )

In fun overloading we can overload.

Two types of operator overloading in c++ is —

1) Unary operator overloading
2) Binary

Two types of overloading in c++.

1) function overloading
2) Operator overloading

#Include ...

Type Conversion

Explicit type conversion :- Manually Value Convert.

In Programming suggest when type Cast → Use Explicit type conversion.

Control ~~it~~ the hand of Programmer. (Understandability of other Programmer for)

Program more effective / Understandable.

No changes in Result.

```
// Implicit type Cast
double a2 = a;
Cout << a2 << endl;
int b2 = b;
Cout << b2 << endl;
```

```
10
3
```

```
// Explicit typeCasting.
double a3 = (double) a;
Cout << a3 << endl;
int b3 = (int) b;
Cout << b3 << endl;
}
```

```
10
3
```

Operator

```
#include <iostream>
#include <typeinfo>
using namespace std;
main() {
  int a=10;
  double b=3.14; // Implicit type casting
  char c='A';
  cout << typeid(a).name() << endl;   // object which tells which type of value we are using.
  cout << typeid(b).name() << endl;
  cout << typeid(c).name() << endl;
  }
```

O/P

i
d
c

Type casting → Implicit type casting
→ Explicit type casting

1) Implicit type casting :-

```
#include <iostream>
#include <typeinfo>
using namespace std;
main() {
  int a=10;
  double b=3.14; // Implicit type casting
  double a2=a;   // No need to define data type.
  cout << a2 << endl;
  int b2=b;
  cout << b2 << endl;
  }
```

a conv. into double data type a2

O/P - 10
3.14

NOTE: Double (long) convert into int (short)

Difference b/w Type Casting and Type Conversion:

① Type Casting:- A data type is converted into another data type by the Programmer using the casting operator during the Program design.

In type casting the destination data type may be smaller than the source data type. When converting the data type to another data type, that's why it is also called narrowing Conversion.

Syntax:-

destination_datatype = (target_datatype)) variable;

(): is a casting operator.

target_datatype is a data type in which we want to convert the source data type.

Type Casting ex:

```
int a=10;          //a=10;
float b = (float) a; // b= 10.0.00
```

↳ type casting

same value's float version.

If type not define → data Loss.

```
#include <iostream>
#include <typeinfo> //Library.
using namespace std;
main() {
  int a=10;
  double b=3.14;
  char c='A';
cout<<typeid(a).name()<<endl;
cout<<typeid(b).name()<<endl;
cout<<typeid(c).name()<<endl;
}
```

operator → fun.

To tell the type of the variable

type id Operator
(Pass the variable)

O/P ie
d
c

Difference

Function Overloading :-

Whenever a program contains more than one function with same name different types of Parameter called function overloading.

Syntax :-

```
class class-name
{
public:
void add()
{
}
void add(int a)
{
}
};
```

fun

fun

Parameter

```
add();
add(10);
```

Through Parameter we know

fun overloading / Compile time polymorphism

Program :-

```
#include <iostream>
using namespace std;
class A
{ int num1=20, num2=10
public:
void fun()
{ int sum = num1+num2;
cout << "Addition " << sum << endl; }
void fun(int a, int b)
{ int sub = a-b;
cout << "Substraction " << sub << endl; }
};
int main() {
A obj;
obj.fun();
obj.fun(100,50);
return 0;
}
```

fun

More than one obj can create here.

```
demo ob(10,-20); t(100,-300);
ob.show(); //
-ob;   //-t;        the value -ve
                     -ve → +ve
ob.show(); // t.show();
getch();
}
```

O/P
A=10 B=-20
A=100 B=300
A=-100 B=300
A=-10 B=20

A=10 B=-20
A=-10 B=20

Binary operator overloading: A operator which contain two operand is called binary operator overloading.

write a prog to add 2 No using class fun & friend function.)

```
#include<iostream.h>
#include<conio.h>
class demo  //class
{
int a,b; //variable (Private)
public:
demo(int x, int y) //Parameterize Constructor
{
  a=x;
  b=y;
}
void show()  //fun
 {
  cout<<"A"<<a<<"  "<<"B"<<b<<endl; //value print
 }
demo operator+(demo obj)
{
```

demo → default constructor
demo test;

→ Arg Par
x store in A
b's value store in B.

class name ← operator keyword, operator overload

```
};
Void main()
{
  clrscr()  demo
  demo ob(-10,20); // value pass for x & y.
  obj.show();
  getch();
  -ob;        // operator for -ve.
  ob.show();
}
```

a=-10 b=20
a=10 b=-20

a=10 b=-20
a=-10 b=20

More than one operations on a same operator

Using friend fun, unary operator overloading.

```
#include<iostream.h>
Cout<<
)
Friend void operator -(demo & obj);
};
Void operator -(demo &obj)
{
  obj.a=-obj.a;
  obj.b=-obj.b;
)
void main()
  {
   clrscr();
```

We define the identical function outside of the class using the scope resolution operator (::),

```
double Dice::getVolume(void)
  {
  return L*B*H;
  }
```

NOTE:- We must use the class name exactly before the :: operator.

The dot (.) operator will be used to perform a member function on an object and will only manipulate data relevant to that object.

```
Dice myDice; // Generate an object
myDice.getVolume(); // Call the object's member function
```

```
#include <iostream>
using namespace std;
class Dice {
 public:
 double L; // dice's length
 double B; // dice's breadth
 double H; // dice's height

 double getVolume(void);   ← Member function declaration
 void setL(double length);
 void setB(double breadth);
 void setH(double height);
 };
```

object number 1 destroyed

```
double Dice::getVolume(void)
 {
 return L*B*H;
 }

Void Dice::SetL(double length)
 {
  L = Length;
 }
 Void Dice::SetB(double breadth)
  {
   B = Breadth;
  }
 Void Dice::SetH(double Height)
  {  H = Height;
  }

int main()
  {
 Dice Dice1;        // Declare Dice1 of type Dice
 Dice Dice2;        // Declare Dice2 of type Dice
 double volume = 0.0  // Here the vol. of Dice is stored.

 Dice1.SetL(6.0);
 Dice2.SetB(7.0);   // Dice1 Specification
 Dice3.SetH(5.0);

                    Volume of Dice1
 volume = Dice1.getVolume();
 Cout << "Volume of Dice1: " << volume << endl;
  return 0;         O/P → Volume of Dice1 : 210
 }
```

Member function definitions

Main function.

```
}
~test()
{
cout<<"\n\nDestructor Msg: Object number "<<count<<"
destroyed.. ";
count--;
}
};
int main()
{
cout<<"Inside the main block..";
cout<<"\n\ncreating first object T1..";
test T1;
{ //Block1
cout<<"\n\nInside Block 1..";
cout<<"\n\ncreating two more objects T2 and T3..";
test T2, T3;
cout<<"\n\nLeaving Block 1..";
}
cout<<"\n\nBack inside the main block..";
return 0;
}
```

OIP

```
Inside the main block
creating first object T1..
Constructor Msg: object number 1 created..
Inside Block 1..
Creating two more objects T2 and T3..
Constructor Msg: object number 2 created..
Constructor Msg: object number 3 created..
Leaving Block 1..
Destructor Msg: Object number 3 destroyed.
Destructor Msg: Object number 2 destroyed.
Back inside the main block..
Destructor Msg: Object number 1 destroyed
```