Const Member function

```
const int x = 5;
x = 10; → error
```

```
const Addition a(2,3);
a.add();
```

```
class Addition
{
  public:
    int x, y;
    Addition(int a, int b)
    {
      x = a;
      y = b;
    }
    int add() const
    {
      return x+y;
    }
```

Array of Objects

```
Addition a, b, c;

Addition a[100];
```

```
for(int i = 0; i<100; i++)
{
   cout<<a[i].x<<a[i].y;
}
```

Addition a[5]; ⟶ Default constructor needed to make array unintialized object.

Addition a[5] = { Addition(3), Addition(4,5), Addition(6)};

1) X = 3, Y = 0
2) X = 4, Y = 5
3) X = 6, Y = 0
4) X = 0, Y = 0
5) X = 0, Y = 0

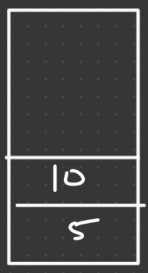
↳ we can intialize array object using multiple constructors. and remaining object will call default constructor.

Abstract Data Type (ADT)

```
class stack
{
    int s[100];
  public:
    void push(int x)
    {
      =
    }
    int pop()
    {
    } =
    int isEmpty()
    {
    } =
    int TOS()
    {
    } =
};
```

ADT

```
#include<stack>
int main()
{
  stack a;
  a.push(5);
  a.push(10);
  a.pop()
  if(a.isEmpty())
  {
  }
  return 0;
}
```



Operator Overloading

2 + 2 = int + int

2.5 + 3 = float + int

3 + 2.8 = int + float

2.5 + 2.7 = float + float

Time + Time → Operator Overloading

```
class Time
{
    int hour, min;
    ==
    ==
};

int main()
{
    Time t1, t2, t3;
    t1 = t2 + t3;   ⟶ will it work ?
}
```

```
Time operator + (Time t)
{
    Time temp;
    int m = min + t.min;
    temp.hour = hour + t.hour + m/60;
    temp.min = (min + t.min) % 60;
    return temp;
}
```

t2.operator+(t3);

T1 = t2 + t3; ✓

t1 = t2 + t3 + t4; ✓

t2 = 2 + t1; → ?

* The overloaded operator must have atleast one operand that is user-defined type. This prevents you from overloading operators for standard types.

You can't use an operator in a manner that violates the syntax rules for the original operator.

For ex:- You can't overload the % operator, so that it can be used with single operand.

%x , x% ✗ x%y ✓

t1 = t2 + t3 ; // t1 = t2.operator+(t3); ✓

t1 = t2 + 2; // t1 = t2.operator+(2); ✓

Problem ⟶ t1 = 2 + t2; // t1 = 2.operator+(t2); ✗

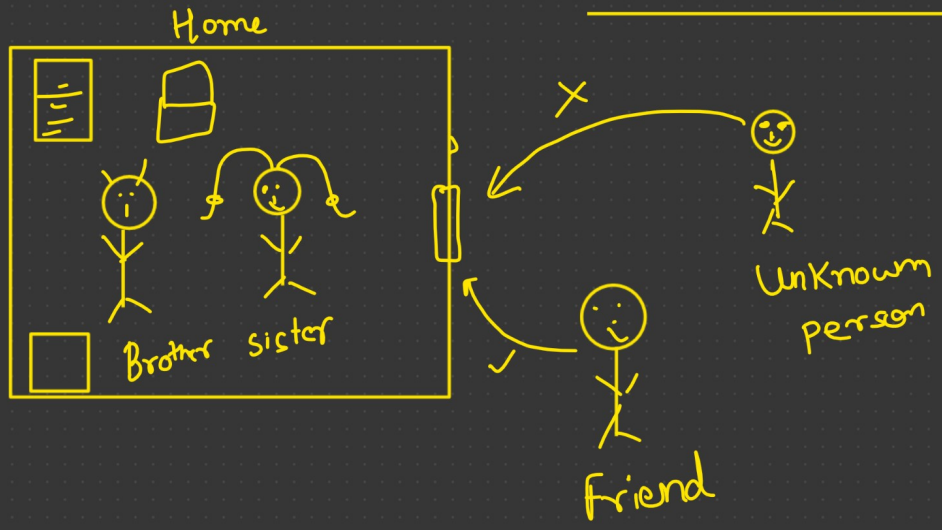
↑ error

But we know that ⇒ t1 + 2 = 2 + t1 → Both are same

What is its Solution ?

↳ friend function

Friend Function




```
Class ABC
{
    int x, y;          <- private members
  public :
    Void print()
    {
      cout<<x<<" "<<y;
    }
  ✓ friend ABC add(ABC a, ABC b);   <- declaration
};
```

```
friend ABC add( ABC a, ABC b)      <- defination
{
    ABC temp;
    temp.x = a.x + b.x;
    temp.y = a.y + b.y;
    return temp;
}
```

friend function

→ Ouside the class

```
int main ()
{
    ABC a;
    a.x = 5; //error
```

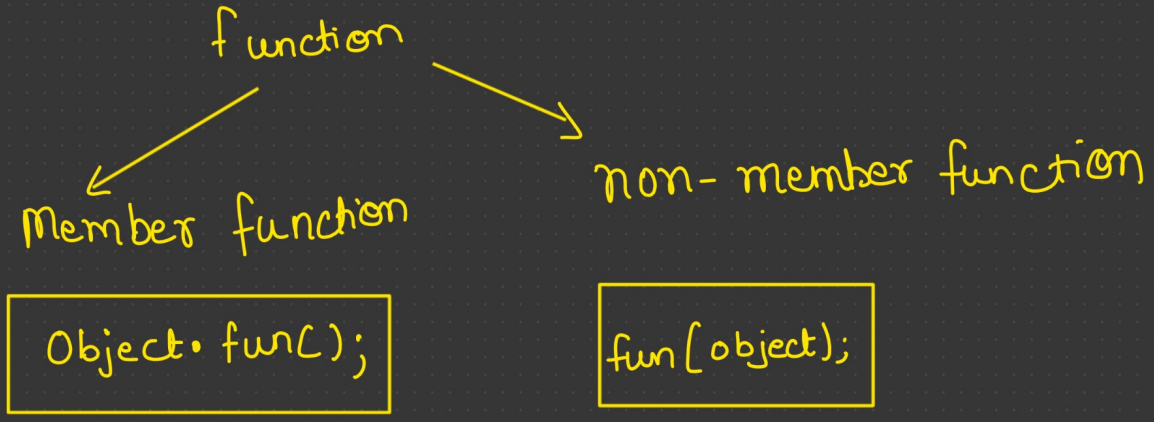
↑ Because x is private member of class ABC

```
    a.print(); //valid
}
```

↑ Unknown function

```
int main()
{
    ABC a,b;
    a.print();
    add(a,b);
}
```

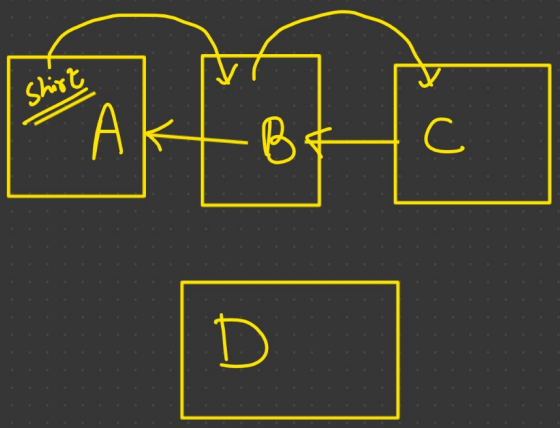
print() is member function of ABC class



friend function :- It is a non-member function.

* No. of Parameters required in non-member function is always 1 more than the no. of parameters required for member function.

Obj. MF(3) = NMF(4) ✓



Now, Let's see How to solve 2 + t1 ⇒ int + Time

```
class Time
{
   int hour, min;

 public:

   friend Time operator+( int a, Time b);
};

 Time operator+ (int a, Time b)
 {
   Time temp;

      temp.hour = a + b.hour;
      temp.min  = b.min;

    return temp;
  }
```

```
int main()
{
  Time t1(2,30), t2;

  t2 = 5+t1; ⇒ operator+(5,t1);

  t2.print()

   return 0;
}
```

Time::

- We don't use friend keyword in the definition
- You don't need to use Time:: qualifier.

Overloading << operator

int x = 5;

Cout << x ; → will it work ? → Yes

cout is a object of ostream class

Cout.operator<<(int);

↑

It is defined in Ostream Class

Time t1;

Cout << t1;

→ will it work ? → No.

Cout.operator<<(Time);

↑

There is no such type of function defined in ostream class having Time as its parameters.

Overloading

- using member fn
- using friend fn.

```
class Time
{
    public :
      return ?  operator << ( Ostream & )      // parameter ?
        {
            cout << " Hour = " << hour << " min = " << min << endl;
        }

};

 int main()
 {
   Time t1;
    t1 << cout; // t1.operator<<(Ostream)

   return 0;
 }
```

```
Cout<<x;
t1<<cout;
```

I don't like this syntax.

I want to write this:-

```
cout << t1;
```

(Ostream object).operator<<(Time);

↑ Is this possible ? → Not possible.

We can't add new fn in Ostream class.

So, It is "not" possible using "member function".

Now let's look at using friend function:-

Class Time
{

};

class Ostream
{

};

2+ t1 → (int, Time)

cout << t1

friend return type operator<<(Ostream , Time)

```
friend ostream& operator<<(ostream &os, Time T)
{
    os << "Hour = "<< hour<<" min = "<< min <<endl;
    return os;
}
```

Why reference is needed?

Ans:- Because we are not allowed to create an object of ostream class but we can make the reference of ostream.

It is needed to support

```
cout<< t1 <<t2 <<t3;
```

```
ostream os = cout; → error
ostream &os = cout; → Allowed
```