For eg `c = a + b;`

but if one is userdefined (a) & other is built in type variable then automatic conversion is not possible then.

Three types of situation might arise in the data conversion.

1. basic type to class type
2. class type to basic type
3. one class to another class

① Basic to class type → ① we use constructor to convert the basic into class type.

```
class distance
{
int feet;
float inches;
Public: distance(float mtr)
        {
        float f = 3.28 * mtr;
        feet = int(f);
        inches = 12 * (f - feet);
        }
void show()
    {
    cout << "distance is" << endl;
    cout << feet << "-" << inches;
    } };
void main()
    {
    float meter;
    cin >> meter;
    distance d1(meter);
    d1.show();
    getch();
    }
```

1M = 3.28 feet
1F = 12 inches

.3048

float to classtype

```
class time
{
int hrs;
int mins;
Public:
time(int t)
{
    hrs = t/60;
    mins = t % 60;
}
};

void main()
{
time T;
int duration = 80;
T = duration;
}
```

⇓

time T(duration);

int to classtype

(11) Class to basic type → (1) By constructor, we can not convert

(11) From class to basic type, we can define a overloaded casting operator in C++.

i.e

```
operator basic-typename( )
{
    body;
}
```

(1) must be class member
(11) must not specify a return type
(111) must not have any arguments.

foreg →

```
class distance
{
int feet;
float inches;
Public: distance(int f, float i)
        {
          feet = f, inches = i;
        }
    void show()
    {
    cout << feet << " - " << inches;
    }
    operator float()
    {
    float ft = inches/12;
          ft = ft + feeet;
          return (ft/3.28);
    }
};
void main()
{
distance d1(3,3.36);  → class type
float m = d1;  → class to basic type
cout << d1.show();
cout << m;
getch()
}
```

O/P ⇒ 1 meter

(3) One class to another class (i) using constructor

(ii) ~~using operator type~~ using overloaded casting operator

```
class student1
{
int roll, marks;
Public:
Student1(int a, int b)
{
roll=a;
marks=b;
}
void print()
{
cout<< roll;
cout<<marks;
}
int getroll()
 {
  return roll;
 }
float getmarks()
  {
   return marks;
  };
  };
```

```
class Student2
{
 int roll;
 float per;
 Public:
Student2 (Student1 S2)
 {
 roll = S2.getroll();
 per = S2.getmarks()*100;
        100
 }
 void print()
 {
  cout<< Roll<< per;
 }
 };

Void main()
{
Student1 S1(25, 57);
Student2 S2 = S1;
 S2.print();
}
```

Overloading of memory management