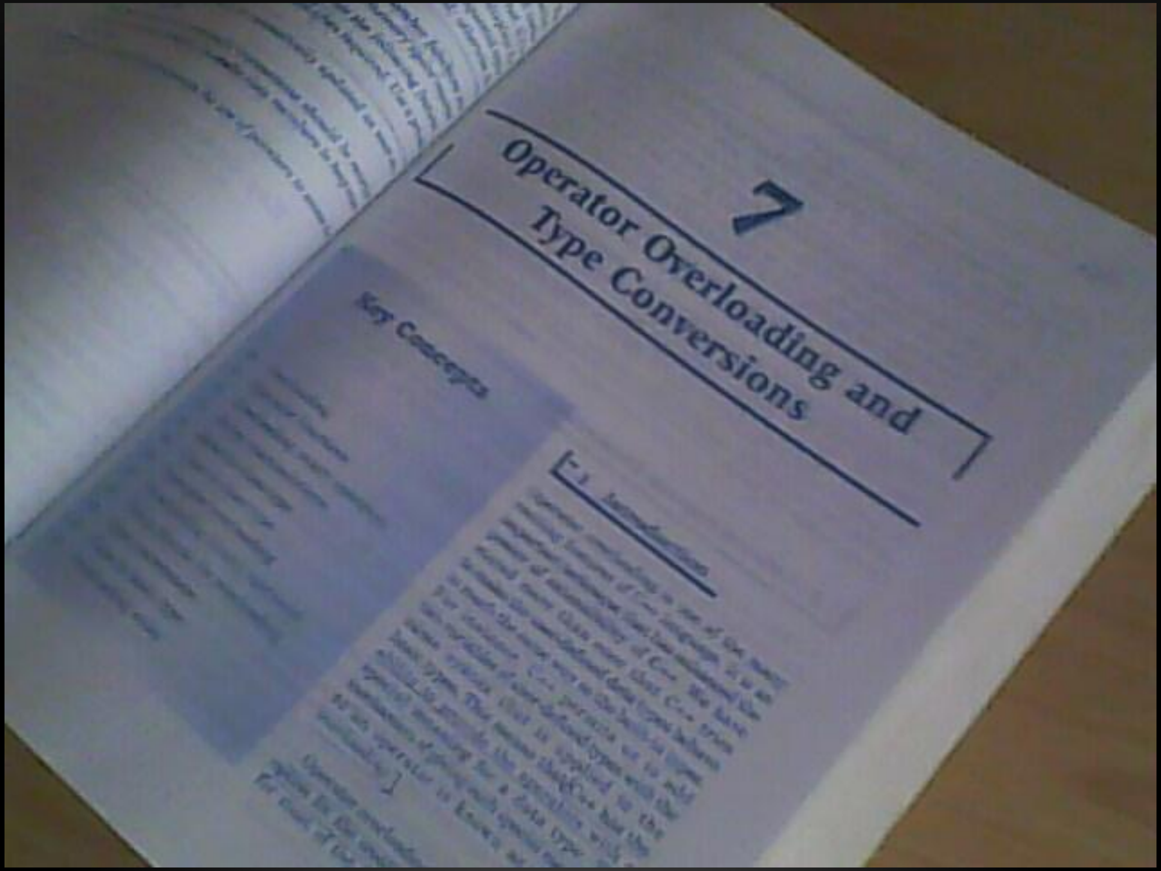
OUR TOPIC IS OPERATOR OVERLOADING AND TYPE CONVERSIONS



7

Operator Overloading and Type Conversions

Key Concepts

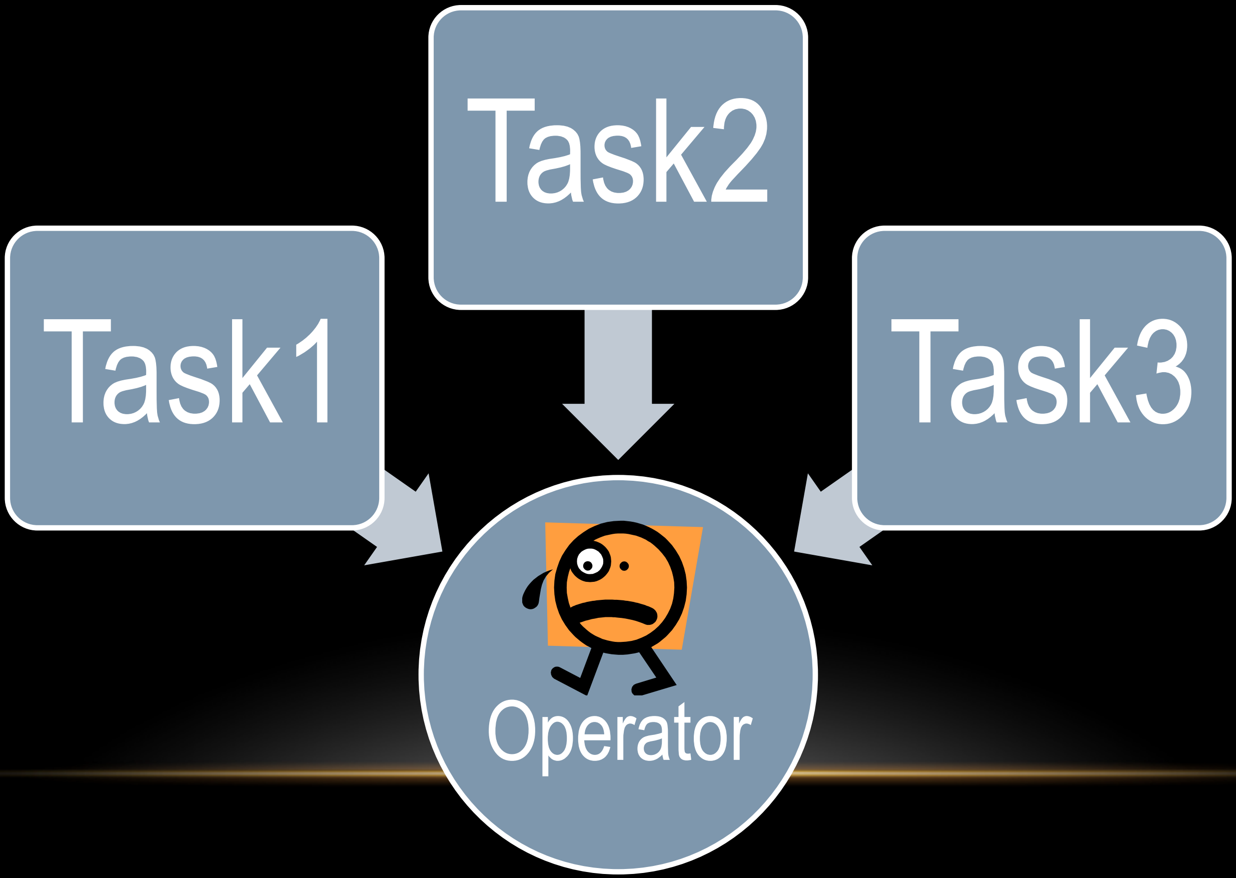
[illegible]

7.1 Introduction

[illegible]

BY HARDEEP SINGH

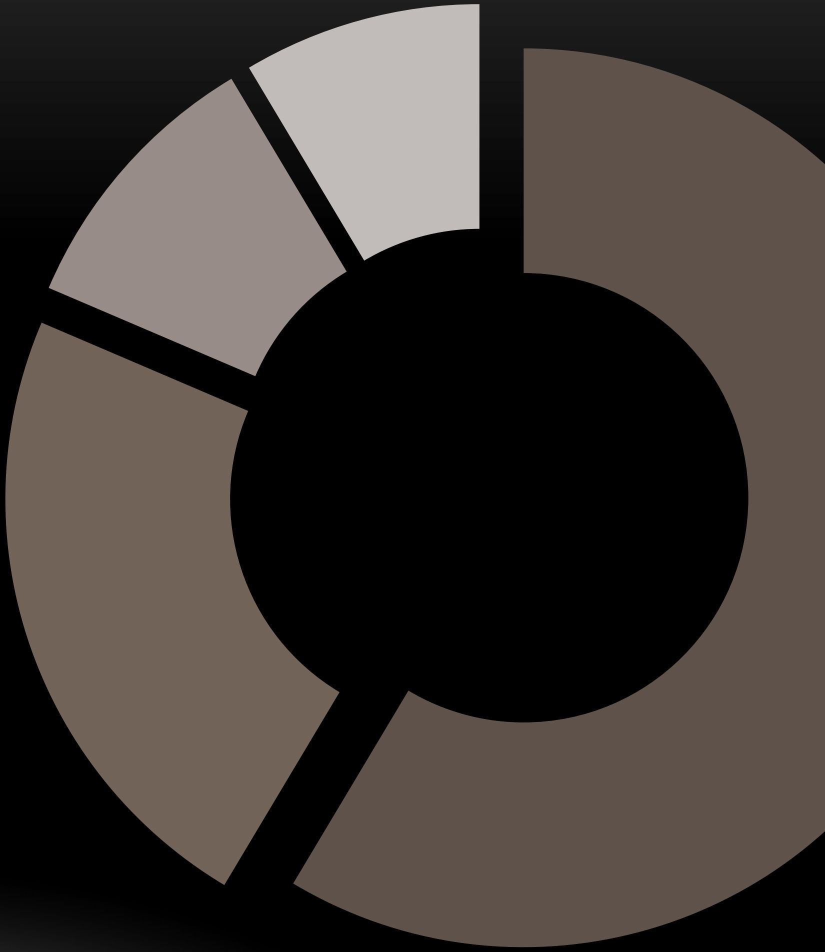
ACTUALLY WHAT IS OPERATOR OVERLOADING



BY HARDEEP SINGH

KEY CONCEPTS

- Introduction
- Defining operator overloading
- Operator functions
- Overloading unary operator
- Overloading binary operators
- Using friends for overloading
- Overloading rules
- Type conversions
- Basic to class type
- Class to basic type
- Class to class type



INTRODUCTION

- Operator overloading is the one of the many exciting features of C++ language. It is an important technique that has enhanced the power of extensibility of C++. C++ tries to make the user defined data types behave in much the same way as the built-in data type . For instance , C++ permits us to add two variables of the user defined data types with same syntax that is applied to basic data types. This means that C++ has the ability to provide the operator with a special meanings to an operator is known as **operator overloading.**

DEFINING THE OPERATOR OVERLOADING

- Operator function - Defines the new task which is going to assign to the operator.

```
Return type  classname :: operator op(arglist)
{
                    function body              //task defined
}
```

FEW EXAMPLES OF OPERATOR FUNCTION DEFINITION

```
Void space::operator-()
{
        x = -x;
        y = -y;
        z = -z;
}
```

```
Complex ::operator+(complex c)
{
    complex temp;
temp.x = x + c.x ;
temp.y = y + c.y ;
}
```

OVERLOADING UNARY OPERATOR

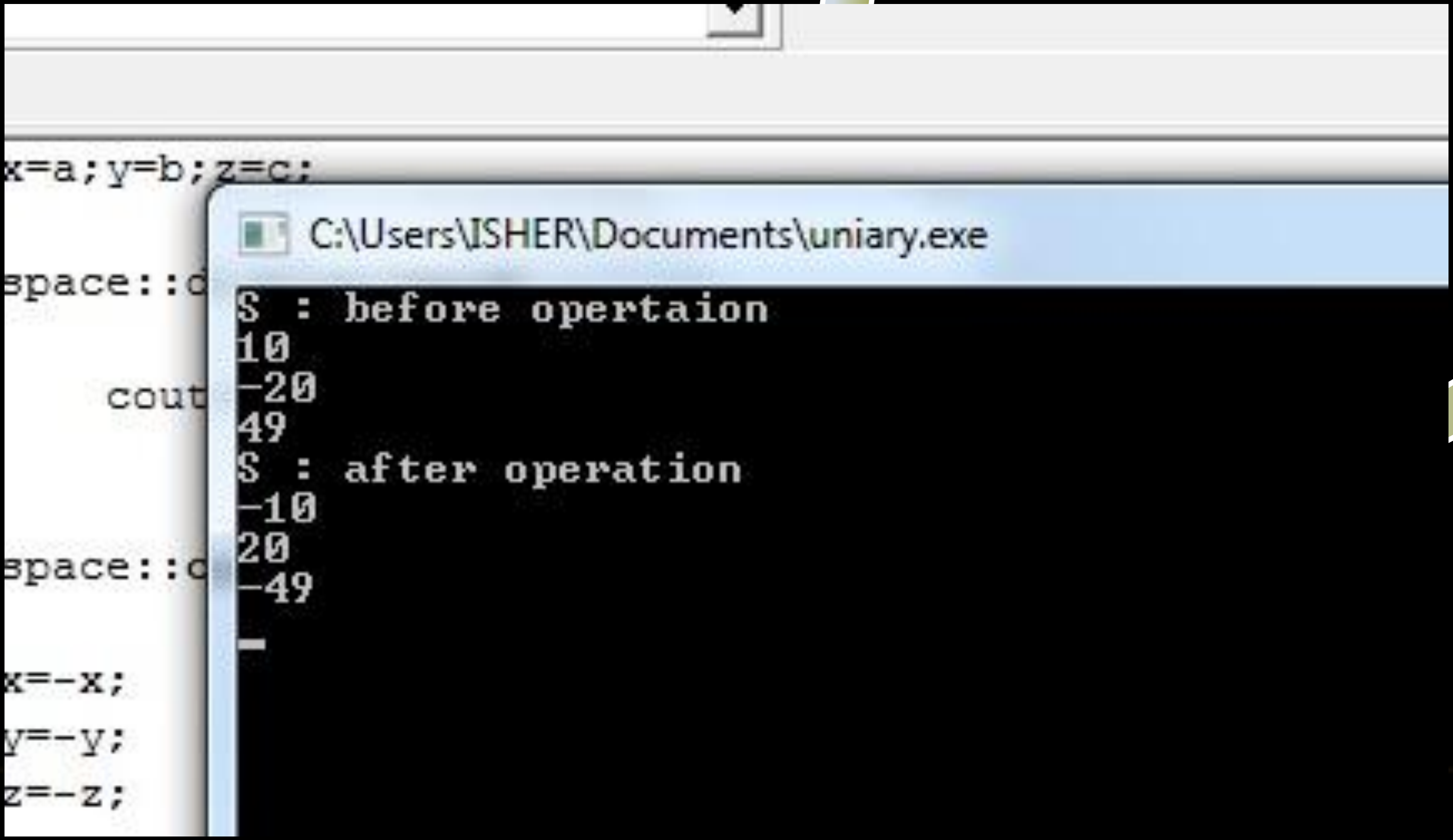
```
#include<iostream.h>
#include<conio.h>
class space{
                    int x,y,z;;
     public:
         void getdata(int,int,int);
         void display();
         void operator-();                    //overload unary minus
    };
void space::getdata(int a,int b,int c)
{
      x=a;    y=b;     z=c;
}
```

CONTINUED…….

```
void space::display(void)
{
        cout<<x<<“\n”;
        cout<<y<<“\n”;
        cout<<z<<“\n”;
}
void space::operator-()
{
             x = -x;
             y = -y;
             z = -z;
}
```

```
int main()
{
space S;
S.getdata(10,-20,30);
cout<<“S : ”;    S.display();
-S;     //activates operator-() function
cout<<“S : “;
S.display();
getch();
return 0;
}
```

OUTPUT FOR THE PROGRAM



```
x=a;y=b;z=c;

space::c

     cout

space::c

x=-x;
y=-y;
z=-z;
```

C:\Users\ISHER\Documents\uniary.exe

```
S : before opertaion
10
-20
49
S : after operation
-10
20
-49

_
```

BY HARDEEP SINGH

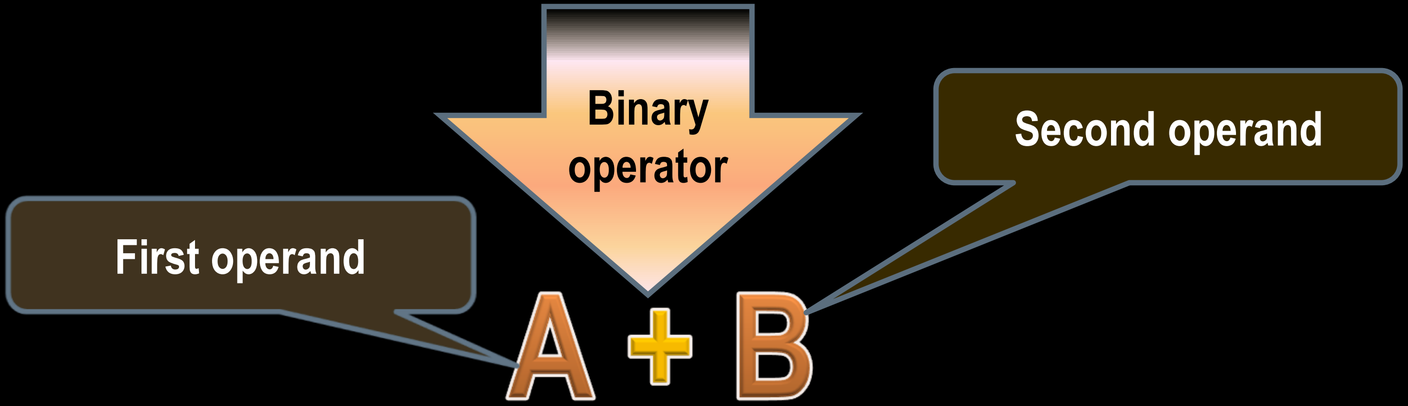
OVERLOADING BINARY OPERATORS

As we have overload an unary operator, same mechanism can be used to overload a binary operator. To add two no. using a friend function a statement like

```
      c=sum(a.b);                  //functional notation
```

can be replaced with help of the operator overloading of +operator by using the expression

```
c=a+b;                             //arithmetic notation
```



PROGRAM FOR OVERLOADING BINARY OPERATOR

```
#include<iostream.h>
class complex
{
  float x;                                  //real part
  float y;                                  //imaginary part
  public:
  copmlex( ){ }
  complex(float real,float imag)
  {x=real;y=imag;}
    complex operator+(complex);
    void display();
};
```

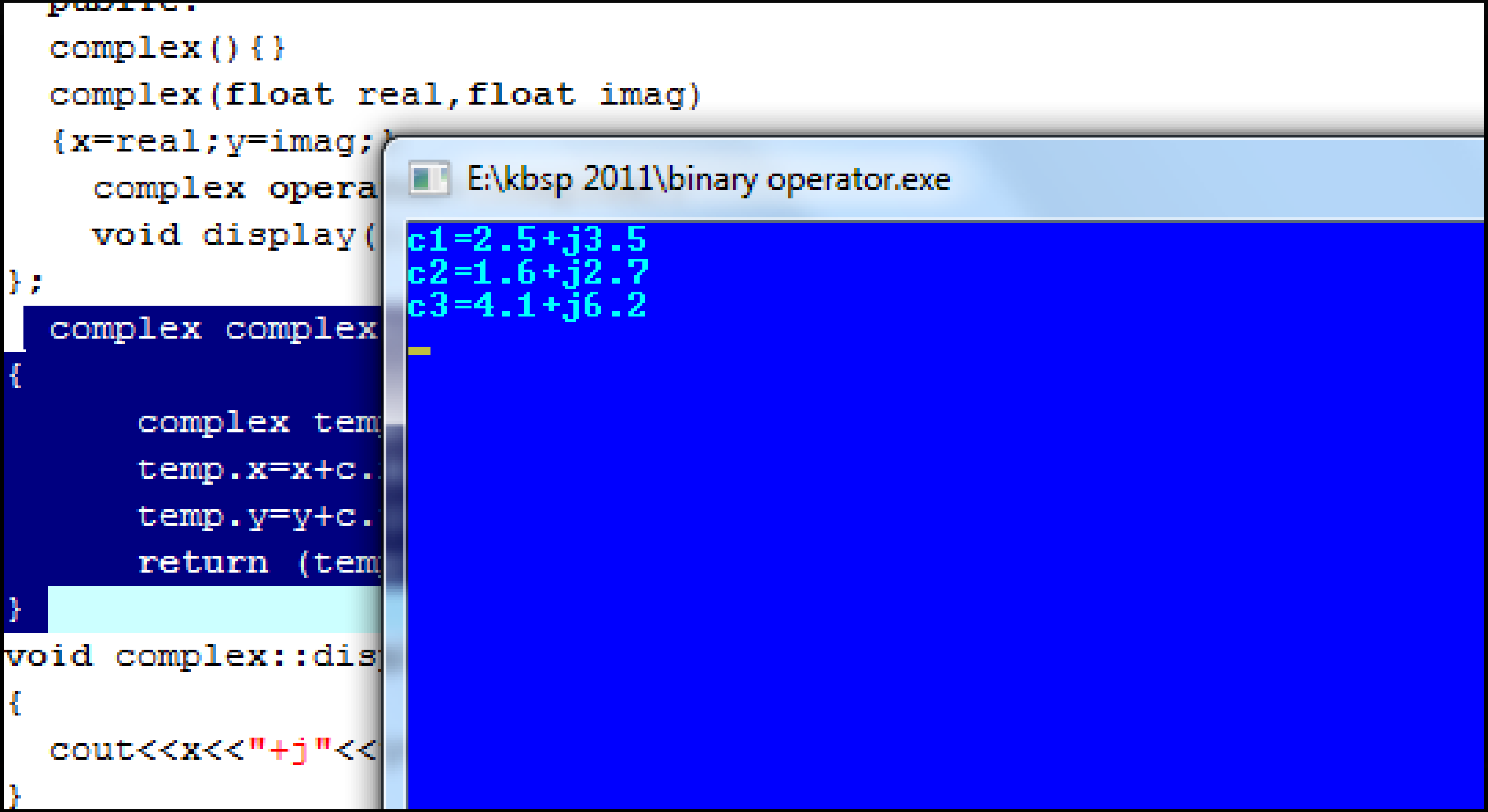
CONTINUED………..

```
 complex complex::operator+(complex c)
{
      complex temp;                    //temporary
      temp.x=x+c.x;
      temp.y=y+c.y;
      return (temp);
}
void complex::display()
{
  cout<<x<<"+j"<<y<<endl;
}
```

CONTINUED………..

```
int main()
{
     complex c1,c2,c3;
    c1=complex(2.5,3.5);      //invokes the constructer 1
    c2=complex(1.6,2.7);  //invokes the constructor 2
    c3=c1+c2;
    cout<<"c1=";c1.display();
   cout<<"c2=";c2.display();
   cout<<"c3=";c3.display();
   getch();
   return 0;
}
```

THE RESULT WILL BE LIKE THIS



```
  complex(){}
  complex(float real,float imag)
  {x=real;y=imag;}
    complex opera
    void display(
};
 complex complex
{
        complex tem
        temp.x=x+c.
        temp.y=y+c.
        return (tem
}
void complex::dis
{
  cout<<x<<"+j"<<
}
```

E:\kbsp 2011\binary operator.exe

```
c1=2.5+j3.5
c2=1.6+j2.7
c3=4.1+j6.2
```

RULES FOR OPERATOR OVERLOADING

1. Only existing operator can be overloaded. New operators can not be created.
2. The overloaded operator must have at least one operand that is of user defined data type.
3. We can't change the basic meaning of an operator. That is to say, we can't redefine the plus(+) operator to subtract one value from other.
4. Overloaded operators follow the syntax rules of the original operators. They can't be overridden.
5. There are some operators that can't be overloaded.
6. We can't use friend functions to overload certain operators. How-ever , member functions can be used to overload them.
7. Unary operators overloaded by means of member function , take no explicit arguments and return no explicit values, but, those overloaded by means of the friend function, take one reference argument (the object of the relevant class).
8. Binary operators overloaded through a member function, take one explicit argument and those which are overloaded through a friend function take two explicit arguments.

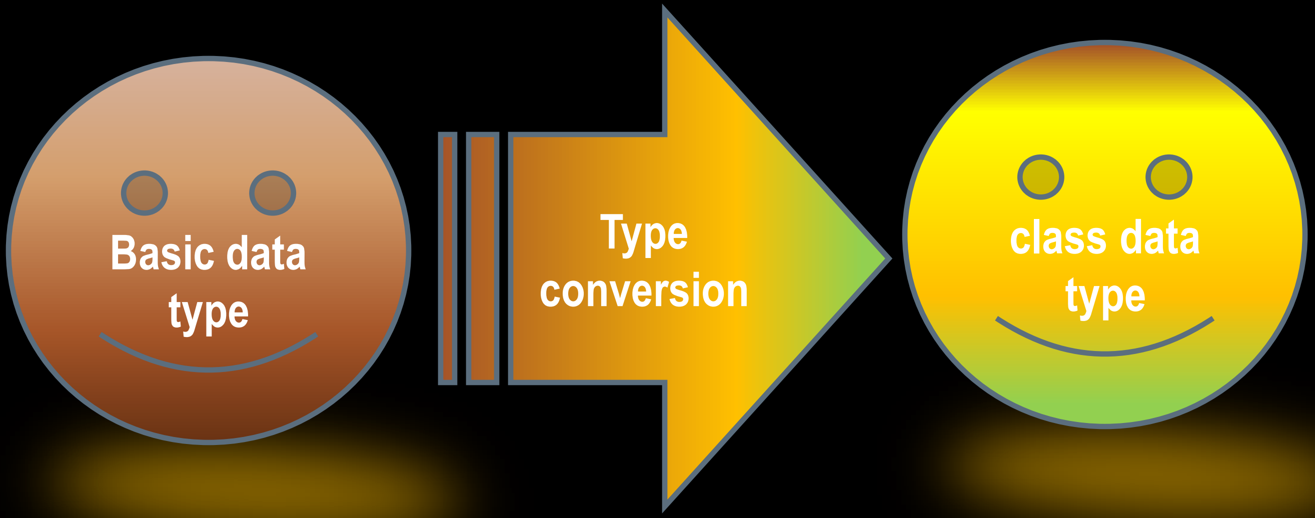
CONTINUED…………

9 When using binary operators overloaded through a member function, the left hand operand must be an object of the relevant class.

10 Binary arithmetic operators such as +,-,* and / must explicitly return a value. They must not attempt to change their own arguments.

UNDER TYPE CONVERSIONS

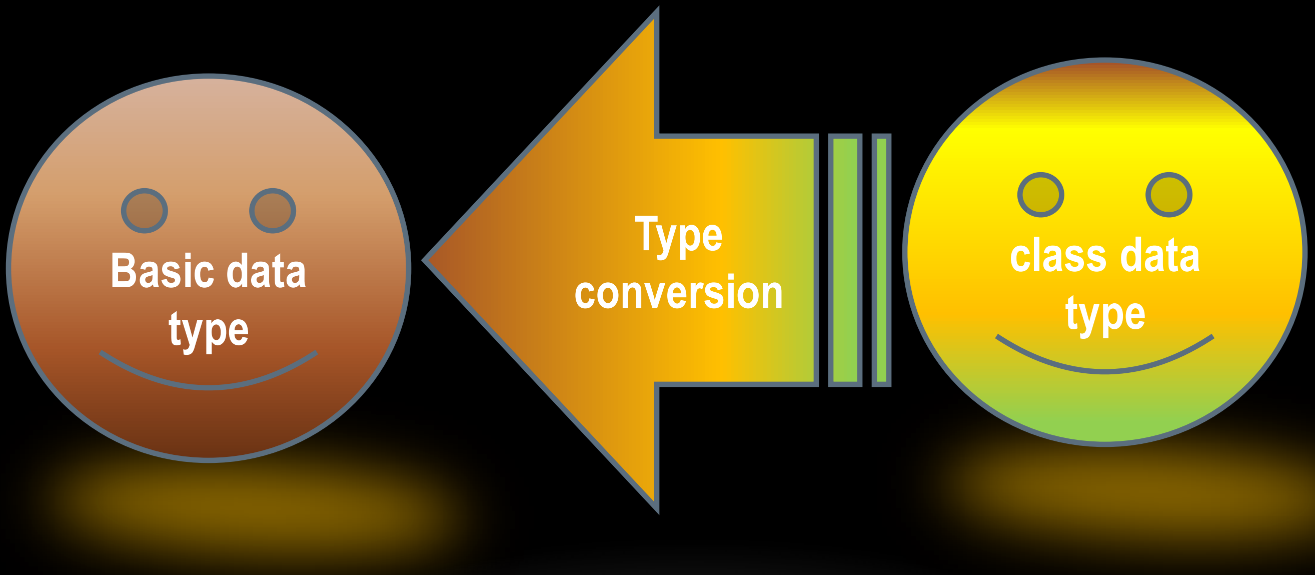
Basic to class type



Predefined data types to User defined data types

UNDER TYPE CONVERSIONS

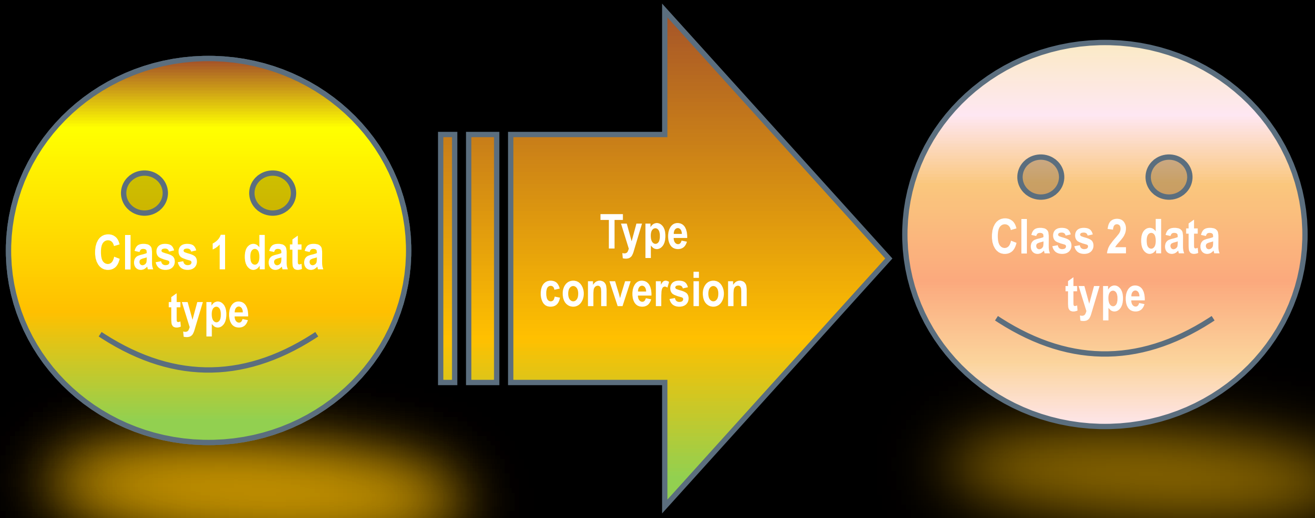
Class to basic type



Predefined data types from User defined data types

UNDER TYPE CONVERSIONS

Class to class type



User defined data types to User defined data types

Thanks a lot for listening
Now please clear your doubts , if any