EX.NO : 4(A)	C++ CLASS OBJECT CONVERSION
DATE :	

PROGRAM STATEMENT :

To write a CPP Program for Class conversion that can be achieved by conversion function which is done by the use of operator overloading

ALGORITHM:

1. Start the program.
2. Define a class Class_type_one with a private float variable a to store a float value. Provide a default constructor to initialize a to 0.0, a parameterized constructor to initialize a with a specific float value, a get method to return a, and a display method to print a.
3. Define another class Class_type_two with a private float variable b to store the converted value. Overload the assignment operator (=) to convert an object of Class_type_one to Class_type_two by assigning a's value to b. Include a display method to print b.
4. In the main function, read a float input from the user and create an object obj1 of Class_type_one initialized with this input.
5. Create an object obj2 of Class_type_two, and use the overloaded assignment operator to assign obj1 to obj2.
6. Call the display method for both obj1 and obj2 to display the values stored in Class_type_one and Class_type_two, respectively.
7. End the program.

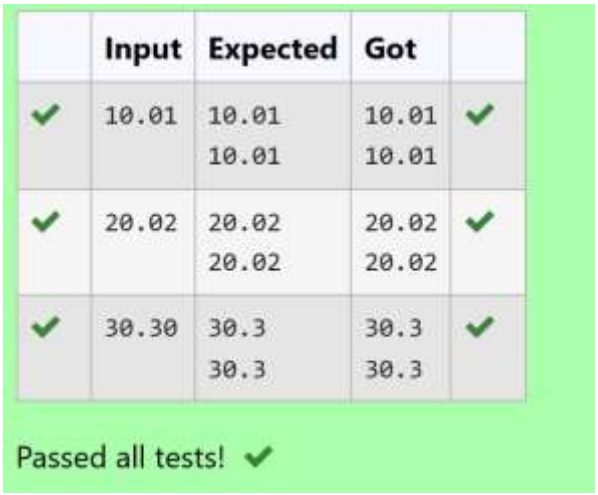
PROGRAM:

```
#include
<bits/stdc++.h>
#include    <string.h>
using namespace std;
//type   to   which   it   will   be
converted      //      Class      to
storetheoriginalfloat value
class Class_type_one
{ float a; // Private membervariable to storethe float
  value

  public:
    // Constructorto initialize thevalue of'a'
    Class_type_one()
```

```
    { a= 0.0; // Initialize withdefault value
    0.0 }

    // Parameterizedconstructorto initializewithaspecific value
    Class_type_one(float value)
    { a=value; // Initialize
    withtheprovidedvalue }

    // Function to returnthestored float value
    float get()
    {   return
      a;
    }

    // Function to displaythe stored value
    void display()
    {   cout   <<a<<
      endl;
    }
};

//classto be converted
// Class to receivetheconverted value
class Class_type_two
{ float b; // Privatemembervariable to storetheconverted
  value

  public:
    // Overloading the assignment operatorto convert Class_type_one to
    Class_type_two void operator=(Class_type_one a_object)
    { b= a_object.get(); // Assignthe value fromClass_type_one'sobject
    to 'b' }

    // Function to displaythe stored value
    void display()
    {   cout   <<b   <<
      endl;
    }
};
```

```
int main()
{ float input; // Variable to storeuser input
  //cout << "Enterafloat value: "; // Prompt
  theuser cin >> input;// Read user input

  Class_type_one obj1(input); // Create an object of Class_type_one and initialize with input value
  Class_type_two obj2; // Create an object of Class_type_two

  obj2 = obj1; // Use theoverloaded '=' operatorto convert obj1 to obj2
  obj1.display(); // Displaythevalue in obj1 (Class_type_one)
  obj2.display(); // Displaythevalue in obj2 (Class_type_two)


  return 0; // Endtheprogram
}
```

OUTPUT :



	Input	Expected	Got	
✔	10.01	10.01 10.01	10.01 10.01	✔
✔	20.02	20.02 20.02	20.02 20.02	✔
✔	30.30	30.3 30.3	30.3 30.3	✔

Passed all tests! ✔

RESULT :

Thus, the C++ program for Class conversion that can be achieved by conversion function, which is done by the use of operator overloading, is created successfully.

EX.NO : 4(B)	C++ COMPOSITION VS. INHERITANCE
DATE :	

PROGRAM STATEMENT :

To write a CPP program to demonstrate on the object composition (use character data).

ALGORITHM:

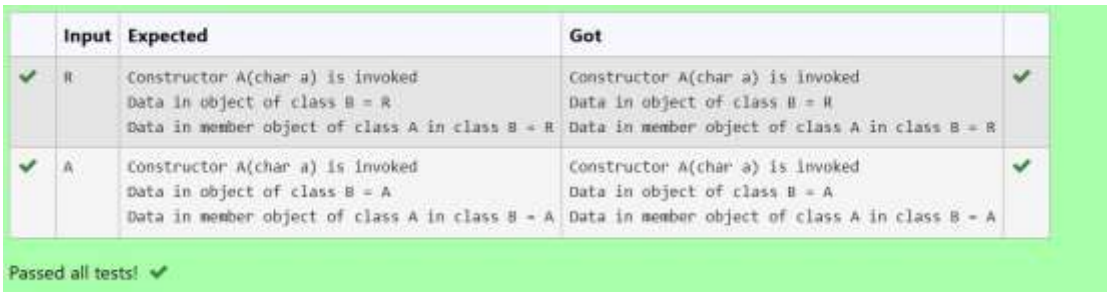
1. Start the program.
2. Define a class A that has a character variable and a constructor to initialize it, along with a method to display its value.
3. Define a class B that contains an object of class A and uses an initializer list to initialize it through B's constructor. Include a display method to show the value of A's object.
4. In the main function, read a character input, create an object of class B with this input, and call the displa'y method.
5. End the program.

PROGRAM:

```
#include          <iostream>
#include <string>
usingnamespacestd;

// Simple class class A { char a; // Member
variable to storethestring

public:
  // Constructorthat initializes thestring 'a'
  A(char str)
  { a = str;
    cout << "Constructor A(char a) is invoked" <<
  endl; }


  //Method to displaythestringstored in 'a'
  void display()
  { cout << "Data in memberobject of class A in class B = " << a
  << endl; }
```

```
};
    class B { AobjA; // Object of class A as amember of class B


public:
  // Constructor that initializes the object 'objA' of class A
  B(charstr):objA(str) // Usinginitializer list to initialize objA
  { cout << "Data in object ofclass B = " << str <<
  endl; }
    void display()

  { objA.display();
  }
};
    int main()

{         char
  input;
  //cout << "Enterastring: ";
  cin >> input;

  B objB(input); // Create an object of class B with the input string
  objB.display(); // Displaydata in objBand its composed object objA


  return 0;
}
```

OUTPUT :



	Input	Expected	Got	
✔	R	Constructor A(char a) is invoked Data in object of class B = R Data in member object of class A in class B = R	Constructor A(char a) is invoked Data in object of class B = R Data in member object of class A in class B = R	✔
✔	A	Constructor A(char a) is invoked Data in object of class B = A Data in member object of class A in class B = A	Constructor A(char a) is invoked Data in object of class B = A Data in member object of class A in class B = A	✔

Passed all tests! ✔

RESULT :

Thus, the C++ program to demonstrate on the object composition is created successfully.

EX.NO : 4(C)	C++ VIRTUAL FUNCTIONS – THIS POINTER
DATE :	

PROGRAM STATEMENT :

To write a CPP program to override the print() function in the base class with the print() function in the child class using the concept of virtual functions.

ALGORITHM:

1. Start the program.
2. Define a base class with a virtual print method that outputs "Base class".
3. Define a derived class that inherits from base and overrides the print method to read and display a string.
4. In the main function, create a pointer of type base and an object of type derived.
5. Assign the address of the derived object to the base pointer and call the print method using the pointer.
6. End the program.

PROGRAM:

```
#include<iostream
> using namespace
std; class base
{ public:
  virtualvoidprint()
  {        cout<<"Base
    class";
  } };
class derived:public base
{ public:
  string str;
  voidprint(
  )
  { cin>>str;
    cout<<str<<end
    l;;
  } }; int
main()
```

```
{ base*bptr;
   derived d;
   bptr=&d;
   bptr->print();
}
```

OUTPUT :

	Test	Input	Expected	Got	
✔	1	VirtualOne	VirtualOne	VirtualOne	✔
✔	2	VirtualTwo	VirtualTwo	VirtualTwo	✔
✔	3	VirtualThree	VirtualThree	VirtualThree	✔

RESULT :

Thus, the C++ program to override the print() function in the base class with the print() function in the child class using the concept of virtual functions is created successfully.

EX.NO : 4(D)	C++ VIRTUAL DESTRUCTORS – DYNAMIC BINDING
DATE :	

PROGRAM STATEMENT :

To write a CPP program to show how the override works using virtual functions and how it works without the virtual concept.

ALGORITHM:

1. Start the program.
2. Define a base class with two methods: disp (virtual) and disp1 (non-virtual), each reading a string input and printing a message.
3. Define a derived class that inherits from base and overrides both disp and disp1 to read and print a different message.
4. In the main function, create a pointer of type base and an object of type derived.
5. Assign the address of the derived object to the base pointer.
6. Use the pointer to call the disp method (virtual, bound at runtime) and disp1 method (non-virtual, bound at compile time).
7. End the program.

PROGRAM:

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

classbase{ public :
   stringstr;
   virtualvoiddisp()
   { cin>>str;
      cout<<str<<" Base Class Not overridden"<<endl;
   }
   void disp1()
   { cin>>str;
      cout<<str<<"         Base         Class         Not
   overridden"<<endl; }
};
```

```
class  derived: public base{
  public:
  string str2;
  void  disp()
  { cin>>str2;
     cout<<str2<<"DerivedClassOverridding"<<endl; }
  void disp1()
  { cin>>str2;
     cout<<str2<<" Derived Class Overridding"<<endl; }
};
int main()
{
   //Base      ClassPointer
   base *bptr;
   //Derived     Class     Object
   derived d;
   //AssignBase Class Pointerwith Derivedclass Object
   bptr = &d;

   //Virtual function, binded at runtime
   bptr->disp();
   //Non-virtual  function,  binded  at  compiletime  bptr-
   >disp1();
   return 0;
}
```

OUTPUT :

	Test	Input	Expected	Got	
✔	1	one one	one Derived Class Overridding one Base Class Not overridden	one Derived Class Overridding one Base Class Not overridden	✔
✔	2	two two	two Derived Class Overridding two Base Class Not overridden	two Derived Class Overridding two Base Class Not overridden	✔

Passed all tests! ✔

RESULT :

Thus, the C++ program to show how the override works using virtual functions and how it works without the virtual concept is created successfully.