```
};
int main( )
{
   Derived2 d2;
   return 0;
}
```

Output:

```
I am Base constructor
I am Derived1 constructor
I am Derived2 constructor
I am Derived2 destructor
I am Derived1 destructor
I am Base destructor
```

9.4. TYPE CONVERSIONS

Consider a simple program as shown below.

Program 9.18

```
/* 9.18.cpp */
#include <iostream>
using namespace std;
int main( )
{
   int a;
   float b;
   b = 5.55;
   a = b;
   cout << a << endl;
   return 0;
}
```

Output:

```
5
```

As we can observe, in 'main ( )' function, variable 'a' is of type **int** and variable 'b' is of type **float**. Both variables are of built-in data types. When assignment operation is carried on, i.e., when the value of 'b' on the right hand side of assignment operator (=) is copied to the variable 'a' on the left hand side of the assignment operator (=), then automatic type conversion takes place. That is, floating point type is converted to integer

type automatically. Hence the output of the above program 9.18 is '5'. The fractional part is truncated.

From this program, it can be proved that, when assigning a value from one variable to another, involving built-in data types, the type conversion is automatic.

But, what if one variable is user-defined type (i.e. class object) and the other is a built-in data type? As class objects are user-defined type, the compiler does not support automatic type conversion for such data types. Therefore, for such operations, the conversion routines have to be written by the user.

✓ Conversion of one data type to another involving user-defined data types can be done using ***constructors*** and ***type-conversion functions.*** Note that, for type conversion, constructor is ***placed in the target*** and type-conversion function is placed in the ***source.*** The previous sentence is well understood by reading further.

✓There are three cases that arise while converting data of one type to another. They are,

- Conversion from basic type to class type
- Conversion from class type to basic type
- Conversion from one class type to another class type

Let us discuss each of them in detail.

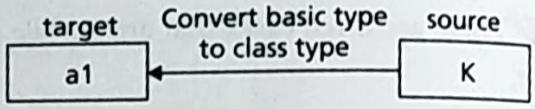
Case 1: Conversion from basic type to class type

In this type of conversion, ***source*** is the ***basic type***(or built-in type) and ***target*** is the ***class type***. Constructors are used for this type of data type conversion. Constructors are placed in the target. Only one argument should be passed to the constructor. The argument passed to the constructor will be a data of some built-in data type which has to be converted to the class type. When the constructor receives a single argument, then implicit type conversion takes place in which the type received by the constructor is converted to a type of the object of a class, in which the constructor is defined. Let us look at an example to understand the concept well.

Program 9.19

```
/* 9.19.cpp */
#include <iostream>
using namespace std;
class A
{
```

```
    private :
       int a, b;
    public :
       A( )
       {
       }
       A( int x ) // Constructor performing conversion
       {
          a = x + 5;
          b = x + 6;
       }
       void display( )
       {
          cout << "a = " << a << endl;
          cout << "b = " << b << endl;
       }
};
int main( )
{
   A a1;// 'a1' is of class type
   int k = 8;//'k' is of basic type
   a1 = k;  //'k' is source, 'a1' is target, converting from basic
            //type to class type
   a1.display ( );
   return 0;
}
```



Output:

```
a = 13
b = 14
```

In the above program, in 'main ()' function, notice the statement,

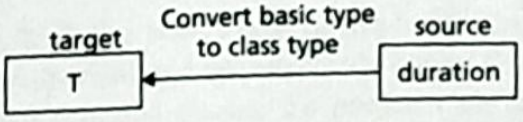
```
a1 = k;
```

Here, the right hand side operand of assignment operator (=) is of 'int' type i.e., a basic data type and the left hand side operand of assignment operator(=) is of 'class A' type, i.e, a user-defined data type. Here, the conversion is taking place from a 'int' data type to a 'class A' type, i.e., from built-in data type to class type while performing an assignment operation. Therefore, the source is 'k' and target is 'a1'. Hence the constructor is placed in the 'class A'(target). So when compiler encounters such a statement, a suitable constructor is called by passing the argument of basic data type to it. Hence, as shown in the program 9.19 above, the constructor converts a 'int' type of data to a 'class A' type data. Note that, only one argument has to be passed to the constructor, which is of basic data type.

Let us try another example.

Program 9.20

```
/*9.20.cpp*/
#include <iostream>
using namespace std;
class Time
{
  private :
     int hours;
     int minutes;
  public :
     Time( )
     {
     }
     Time( int x )
     {
        hours = x / 60;
        minutes = x % 60;
     }
     void display( )
     {
        cout << "Hours = " << hours << endl;
        cout << "Minutes = " << minutes << endl;
     }
};
int main( )
{
   Time T; //'T' is of class type
   int duration = 65; //'duration' is of basic type
   T = duration; // 'T' is target, 'duration' is source, converting
                 // from basic type to   class type
   T.display( );
   return 0;
}
```



Output:

```
Hours = 1
Minutes = 5
```

Case 2: Conversion from class type to basic type

In this type of conversion, the ***source*** is the ***class type*** and ***target*** is the ***basic type***. Special functions known as the type-conversion functions are used for this type of data

type conversions. Constructors are not used. Type-conversion function must be placed in the source. The syntax for operator function is as shown below.

```
operator type( )
{
   //Function body
}
```

where,

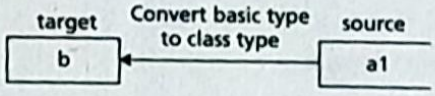
`operator` -> keyword to create a type-conversion function

`type` -> target type i.e, type to which the data type has to be converted.

Let us look into an example program.

Program 9.21

```
/* 9.21.cpp */
#include <iostream>
using namespace std;
class A
{
   private :
      float a, b;
   public :
      A( )
      {
         a = 6.5;
         b = 7;
      }
      operator int ( )
      {
         float x = ( a + b );
         return x;
      }
};
int main( )
{
   A a1; //'a1' is of class type and is the source
   int b = a1; //'b' is of basic type and is the target, converting
               //from class type to basic type
   cout << "b = " << b << endl;
   return 0;
}
```



Output :

```
b = 13
```

In the above program, in 'main( )' function, notice the statement,

```
int b = a1;
```

the right hand side operand (a1) of the assignment operator(=) is of user-defined type, i.e., 'class A' type, and the left hand side operator (b) of the assignment operator(=) is of built-in data type i.e, 'int' type. Hence, source is 'a1' and target is 'b'. Therefore, the type-conversion function is placed in the source (i.e class A). When compiler encounters the statement,

```
int b = a1;
```

it calls a type-conversion function to do the type conversion from class type to built-in type.

The type-conversion function has to satisfy the following conditions.

- It must have no arguments
- It must be a class member function
- It must not specify a return type but should return a value

Case 3: Conversion from class type to class type

In this type of conversion, both the ***source*** and ***target*** is of user-defined type. In other words, the right hand side operand and left hand side operand of the assignment operator(=) are objects of two different classes. ***To perform the type conversion, either of the constructor or type-conversion function can be used.*** **But it has to be kept in mind that, the constructor should be passed with single argument and must be placed in the target, while the operator functions must be placed in the source.** The type conversion using both the techniques is shown below.

Program 9.22

```
/* 9.22.cpp */
/* conversion using constructors */
class B
{
   private :
      int x, y;
   public :
      B( )
      {
```

```
            x = 6;
            y = 7;
        }
        int get_x( )
        {
            return x;
        }
        int get_y( )
        {
            return y;
        }
};
class A
{
    private :
            int a, b;
    public :
        A( )
        {
        }
        A( B b1 )
        {
            a = b1.get_x( );
            b = b1.get_y( );
        }
        void display( )
        {
            cout << "a = " << a << endl;
            cout << "b = " << b << endl;
        }
};
int main( )
{
    A a1; //target
    B b1;  //source
    a1 = b1; //conversion using constructors
    a1.display( );
    return 0;
}
```

Output:

```
a = 6
b = 7
```

Program 9.23

```
/* 9.23.cpp */
/* conversion using type-conversion function */
#include <iostream>
using namespace std;
class A
{
  private :
     int a, b;
  public :
     A( )
     {
     }
     void display( )
     {
        cout << "a = " << a << endl;
        cout << "b = " << b << endl;
     }
     int & get_a( )
     {
        return a;
     }
     int & get_b( )
     {
        return b;
     }
};
class B
{
  Private :
        int x, y;
  Public :
        B( )
        {
           x = 6;
           y = 7;
        }
        operator A( ) // type-conversion function
        {
           A temp;
           temp.get_a( ) = x;
           temp.get_b( ) = x + y;
           return temp;
        }
```

```
};
int main( )
{
   A a1;
   B b1;
   a1 = b1; // conversion using type-conversion function
   a1.display( );
   return 0;
}
```

Output:

```
a = 6
b = 13
```

9.5. THE 'explicit' KEYWORD

We have learnt earlier in this chapter, that, when a constructor receives a single argument, then, an implicit(automatic) type conversion takes place, in which, the type of the argument passed to the constructor is converted from its basic type to the class type in which the constructor is defined. If we wish we can suppress such a behavior. It means, if we do not want such an automatic conversion to take place, then, we should define the class constructors to be "explicit constructors". The **'explicit'** keyword is used to declare the class constructors to be "explicit constructors". Once the constructor is made explicit, the automatic type conversion does not takes place. Let us look into a simple program.

Program 9.24

```
/* 9.24.cpp */
#include <iostream>
using namespace std;
class A
{
   private :
      int a, b;
   public :
      explicit A( int x ) // Explicit Constructor
      {
         a = x + 5;
         b = x + 6;
      }
      void display( )
      {
         cout << "a = " << a << endl;
```