Data Conversion

Contents

What is Data Conversion

As we know that when constants and variables of different types are mixed in a expression, C++ applies automatic type conversion to the operands as per rules. Similarly, an assignment operation also causes the auto conversion. *The type of data to the right of an expression of an `assignment operator` is automalically converted to the type of the variable on the left.* For e.g.

```
int m;
float x = 3.14159;
m = x;
```

What happens when they are user-defined data types?

Since, User-defined data types are designed by the programmer, the compiler doesn't support automatic type conversion for such data types. Therefore, the data coversion concepts helps us, if such operations are required.

Three types of situations might occur: - Conversion from basic type to class type - Conversion from class type to basic type - Conversion from one class type to other class type

1. Basic type to Class type :

The coversion from basic type to class type is easy to accomplish. If we can recall the use of the constructors was ilustrated in the number of examples to initialize objects. For e.g.

Let us consider the conversion of int to a class type:

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

```
main(){
  time T;                 // object T is created
  int countdown = 100;
  T = countdown;         // int to class type
}
```

After this conversion, the `hrs` member of T will contain a value of 1 and the `mins` member a value of 40, denoting 1 Hour and 40 minutes.

2. Class type to Basic Data type :

The constructor function doesn't support this conversion. So, C++ allows us to define an `overloaded casting operator` that count be used to convert a class type to basic type. The general form of an overloading casting operator function , usually referred to as a `conversion function`.

Syntax: `cpp operator typename() { ..... (function statements) }` This function converts the class type to data type *typename* (here _typename_can be int, char, doulbe).

For example,

```
meter :: operator float()
{
  float Length_m;
  Length_m = length * 100.0; // centimeter to meter
  return(Length_m);
}
```

`The above can be invoked by :`

```
float length = float(M1);
```

or

```
float length = M1;
```

where M1 is the object of `class meter`.

The Casting operator function should satisfy the following conditions: - It must be a class member. - It must not specify a return type. - It must not have any arguments.

Since it is a member function, it is invoked by the objects and therefore, the values used for the conversion inside the function belongs to the pbject that invoked the function. This means that the function doesnt need an argument.

3. One Class to Another Class :

For Example :

```
objX = objY;    // objects of different classes
```

The **class Y** type data is converted to the **class X** type data and the conerted value is assigned to the **objX**. Since the conversion takes place from **class Y** to **class X**. Y is known as the *source class* and the X is known as the *destination class*.

Types of Conversions:

Conversion Required	Conversion takes place	
	Source class	Destination class
Basic –> Class	N/A	Constructor
Class –> Basic	Casting Operator	N/A
Class –> Class	Casting Operator	Constructor

Example Programs:

```
#include<iostream.h>
Class alpha{
  int commonA;
public:
  alpha(){}
  alpha(int x)
  {
    commonA = X;
  }
  int getvalue(){

    cout<<" Counstructor Called "<<endl;
    return commonA;
  }
};
class beta
{
  int commonB;
public:
  beta(){}
  beta(int x)
  {
      commonB = x;
  }
```

```
  beta(alpha temp)
  {
      commonB = temp.getvalue();
  }
};

int main()
{
    alpha obj(10);
    beta objb = obja;
}
```