TYPE CONVERSION

Er. Rudra Nepal

TYPE CONVERSION

- Type conversion is converting one type of data to another type.
- Compiler automatically converts basic to another basic data type (for eg int to float, float to int etc.) by applying type conversion rule provided by the compiler.
- The type of data to the right of the assignment operator (=) is automatically converted to the type of variable on the left.

For example:
The statements:
int m;
float x=3.14159
m=x;
Value stored in m is 3 because, here the fractional part is truncated.
Compiler does not support automatic type conversion for user-defined data type. Therefore, we must design conversion routines for the type conversion of user-defined data type.

TYPE CONVERSION

There are three possible type conversions are:

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

Conversion from basic to class type (basic to user defined type)

The constructor is used for the type

conversion takes the single argument which type is to be converted.

Example1:

```
#include<iostream>
using namespace std;
class time
{
int hours;
int minutes;
public:
time(int t)
{
hours=t/60;
minutes=t%60;
}
void display()
{
cout<<"Hours="<<hours<<endl;
cout<<"Minutes="<<minutes<<endl;
}
};
int main()
{
int duration=65;
time t1=duration;
t1.display();
return 0;
}
```

After conversion, the hours members of t1 contains the value 1and minutes member contains

the value 5,denoting 1 hours and 5 minutes.

ample:

xample 2:

Program to convert meter to feet and inches using (Basic to class type conversion)

```
#include <iostream>
using namespace std;
class dist
{
private:
int feet;
float inches ;
public:
dist()
{
}
dist(float m)
{
float f=3.28083*m ;
feet=int(f) ;
inches=12*(f-feet) ;
}
void display()
{
cout<<feet<<"feet"<<inches<<"inches"<<endl;
}
};
```

```
int main()
{
float meter=12.5;
dist d1;
d1=meter;
d1.display();
return 0;
}
```

QUESTION

Make a class called memory with member data to represent bytes, kilobytes, and megabytes. In your program, you should be able to use statements like m1=1087665; where m1 is an object of class memory and 1087665 is an integer representing some arbitrary number of bytes. Your program should display memory in a standard format like 1 megabyte 38 kilobytes 177 bytes.

Example:

```
#include<iostream>
using namespace std;
class memory
{
int mb;
int kb;
int byte;

public:
memory()
{
}
memory(long int m)
{
int rem;
mb=m/(1024*1024);
rem=m%(1024*1024);
kb=rem/1024;
byte=rem%1024;
}
void display()
{
cout<<mb<<"megabytes"<<endl;
cout<<kb<<"kilobytes"<<endl;
cout<<byte<<"bytes"<<endl;
}
};
int main()
{
memory m1;
long int m=1087665;
m1=m;
m1.display();
return 0;
}
```

CLASS TO BASIC TYPE

C++ allows us to define an overloaded casting operator that could be used to convert a class data type to a basic type. The general form of an overloaded casting operator function, usually referred to as a conversion function is:

```
operator typename()
{
..... .
//function statemets
..... ...
}
```

The function converts a class type to typename. For example,the operator int() converts an class object to type int,operator float() converts the class type object to float and so on.

The casting operator function should satisfy the following conditions.

It must be a class member.

It must specify return type.

It must not have any arguments.

Example:

```
#include<iostream>
using namespace std;
class item
{
private:
float price;
int quantity;
public:
item(float p,int q)
{
price=p;
quantity=q;
}
void display()
{
cout<<"Price of items="<<price<<endl;
cout<<"Quantity of items="<<quantity<<endl;
}
operator float()
{
return (price*quantity);
}
};
int main()
{
item i1(255.5,10);
float total;
i1.display();
total=i1;
cout<<"Total amount="<<total<<endl;
return 0;
}
```

Example 2:

Program to convert feet and inches into meter (class to basic type conversion)

```
#include <iostream>
using namespace std;
class dist
{
int feet;
float inches;
public:
dist(int f, float i)
{
feet=f;
inches=i;
}
void display()
{
cout<<feet<< "feet"<<inches<<"inches"<<endl;
}
operator float()
{
float f=inches/12 ;
f=f+feet ;
return(f/3.28083);
}
};
int main( )
{
dist d1(5,3.6) ;
float m=d1;
cout<<"Distance in feet and inches:"<<endl;
d1.display();
cout<<"Distance in meter="<<m<<endl;
return 0;
}
```

ONE CLASS TYPE TO ANOTHER CLASS

We can convert one class type data to another class type.

Example:

objX=objY; //objects of different classes

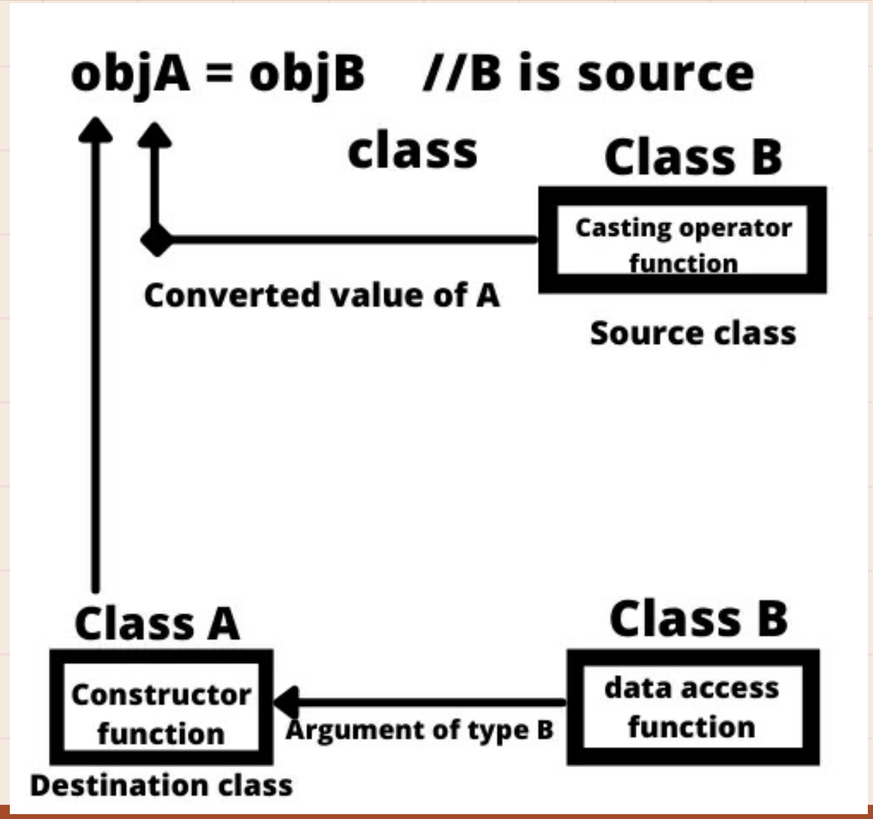
ObjX is an object of class X and objY is an object of class Y. The class Y type data is converted to the class X type data and the converted value is assigned to the objX. Since the conversion takes place from class Y to class X,Y is known as the source class and X is known as the destination class.

Such conversions between objects of different classes can be carried out by either a constructor or a conversion function.

Note:

Conversion form a class to any other type (or any other class) should make use of casting operator function in the source class.

On the other hand to perform the conversion from any other type /class type constructor should be used in destination class.


objA = objB //B is source
class
Class B
Casting operator function
Converted value of A
Source class
Class A
Class B
Constructor function
Argument of type B
data access function
Destination class

Conversion Routine in source class (conversion function in source class)
Conversion from polar to rectangle (using conversion routine in polar)

```
#include<iostream>
#include<math.h>
using namespace std;
class Rectangle
{
private:
float xco;
float yco;
public:
Rectangle()
{
xco=0.0;
yco=0.0;
}
Rectangle(float x,float y)
{
xco=x;
yco=y;
}
void display()
{
cout<<"("<<xco<<","<<yco<<")"<<endl;
}
};
class Polar
{
private:
float radius;
float angle;
public:
Polar()
{
radius=0.0;
angle=0.0;
}
Polar(float r,float a)
{
radius=r;
angle=a;
}
void display()
{
cout<<"("<<radius<<","<<angle<<")"<<endl;
}
operator Rectangle()
{
float x=radius*cos(angle);
float y=radius*sin(angle);
return Rectangle(x,y);
}
};
int main()
{
Polar p(10.0,0.758539);
Rectangle r;
r=p;
cout<<"Polar coordinates=";
p.display();
cout<<"Rectangular coordinates=";
r.display();
return 0;
}
```

Conversion from Rectangle to Polar (using conversion routine in rectangle)

```
#include<iostream>
#include<math.h>
using namespace std;
class Polar
{
private:
float radius;
float angle;
public:
Polar()
{
radius=0.0;
angle=0.0;
}
Polar(float r,float a)
{
radius=r;
angle=a;
}
void display()
{
cout<<"("<<radius<<","<<angle<<")"<<endl;
}
};
class Rectangle
{
private:
float xco;
float yco;
public:
Rectangle()
{
xco=0.0;
yco=0.0;
}
Rectangle(float x,float y)
{
xco=x;
yco=y;
}
void display()
{
cout<<"("<<xco<<","<<yco<<")"<<endl;
}
operator Polar()
{
float a=atan(yco/xco);
float r=sqrt(xco*xco+yco*yco);
return Polar(r,a);
}
};
Cont main()
{
Rectangle r(7.07107,7.07107);
Polar p;
p=r;
cout<<"Rectangular coordinates=";
r.display();
cout<<"Polar coordinates=";
p.display();
return 0;
}
```