**EXPERIMENT NO. 07**

|  |  |
| --- | --- |
| **DATE OF PERFORMANCE:** | **GRADE:** |
| **DATE OF ASSESSMENT:** | **SIGNATURE OF LECTURER/ TTA:** |

**AIM: To Study Operator Overloading and Type Conversion in C++.**

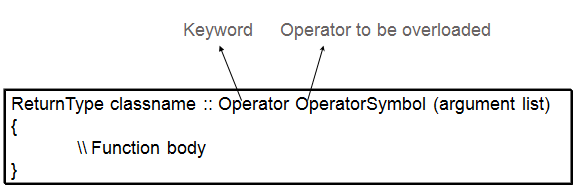
**OPERATOR OVERLOADING:**

**Operator overloading is an important concept in C++. It is a type of polymorphism in which an operator is overloaded to give user defined meaning to it. Overloaded operator is used to perform operation on user-defined data type. For example '+' operator can be overloaded to perform addition on various data types, like for Integer, String(concatenation) etc.**

**Almost any operator can be overloaded in C++. However there are few operators which cannot be overloaded.Operators that are not overloaded are follows**

* **scope operator – ::**
* **sizeof**
* **member selector - .**
* **member pointer selector - \***
* **ternary operator - ?:**

**OPERATOR OVERLOADING SYNTAX:**

****

**IMPLEMENTING OPERATOR OVERLOADING:**

**Operator overloading can be done by implementing a function which can be :**

1. **Member Function**
2. **Non-Member Function**
3. **Friend Function**

**Operator overloading function can be a member function if the Left operand is an Object of that class, but if the Left operand is different, then Operator overloading function must be a non-member function.**

**Operator overloading function can be made friend function if it needs access to the private and protected members of class.**

### OPERATOR OVERLOADING EXAMPLES:

**Almost all the operators can be overloaded in infinite different ways. Following are some examples to learn more about operator overloading. All the examples are closely connected.**

**OVERLOADING UNARY OPERATOR IN C++:**

**The unary operators operate on a single operand and following are the examples of Unary operators:**

* **The increment (++) and decrement (--) operators.**
* **The unary minus (-) operator.**
* **The logical not (!) operator.**

**Following example explain how minus (-) operator can be overloaded for prefix as well as postfix usage.**

**#include <iostream.h>**

**class Distance**

**{**

**private:**

**int feet; // 0 to infinite**

**int inches; // 0 to 12**

**public:**

**// required constructors**

**Distance(){**

**feet = 0;**

**inches = 0;**

**}**

**Distance(int f, int i){**

**feet = f;**

**inches = i;**

**}**

**// method to display distance**

**void displayDistance()**

**{**

**cout << "F: " << feet << " I:" << inches <<endl;**

**}**

**// overloaded minus (-) operator**

**Distance operator- ()**

**{**

**feet = -feet;**

**inches = -inches;**

**return Distance(feet, inches);**

**}**

**};**

**int main()**

**{**

**Distance D1(11, 10), D2(-5, 11);**

**-D1; // apply negation**

**D1.displayDistance(); // display D1**

**-D2; // apply negation**

**D2.displayDistance(); // display D2**

**return 0;**

**}**

**When the above code is compiled and executed, it produces the following result:**

**F: -11 I:-10**

**F: 5 I:-11**

**OVERLOADING BINARY ARITHMETIC OPERATOR:**

**Arithmetic operator is most commonly used operator in C++. Almost all arithmetic operators can be overloaded to perform arithmetic operation on user-defined data type. In the below example we have overridden the + operator, to add to Time(hh:mm:ss) objects.**

**Example: overloading '+' Operator to add two time object**

**#include<iostream.h>**

**#include<conio.h>**

**class time**

**{**

**inth,m,s;**

**public:**

**time()**

**{**

**h=0, m=0; s=0;**

**}**

**voidgetTime();**

**void show()**

**{**

**cout<< h<< ":"<< m<< ":"<< s;**

**}**

**time operator+(time); //overloading '+' operator**

**};**

**time time::operator+(time t1) //operator function**

**{**

**time t;**

**inta,b;**

**a=s+t1.s;**

**t.s=a%60;**

**b=(a/60)+m+t1.m;**

**t.m=b%60;**

**t.h=(b/60)+h+t1.h;**

**t.h=t.h%12;**

**return t;**

**}**

**void time::getTime()**

**{**

**cout<<"\n Enter the hour(0-11) ";**

**cin>>h;**

**cout<<"\n Enter the minute(0-59) ";**

**cin>>m;**

**cout<<"\n Enter the second(0-59) ";**

**cin>>s;**

**}**

**void main()**

**{**

**clrscr();**

**time t1,t2,t3;**

**cout<<"\n Enter the first time ";**

**t1.getTime();**

**cout<<"\n Enter the second time ";**

**t2.getTime();**

**t3=t1+t2; //adding of two time object using '+' operator**

**cout<<"\n First time ";**

**t1.show();**

**cout<<"\n Second time ";**

**t2.show();**

**cout<<"\n Sum of times ";**

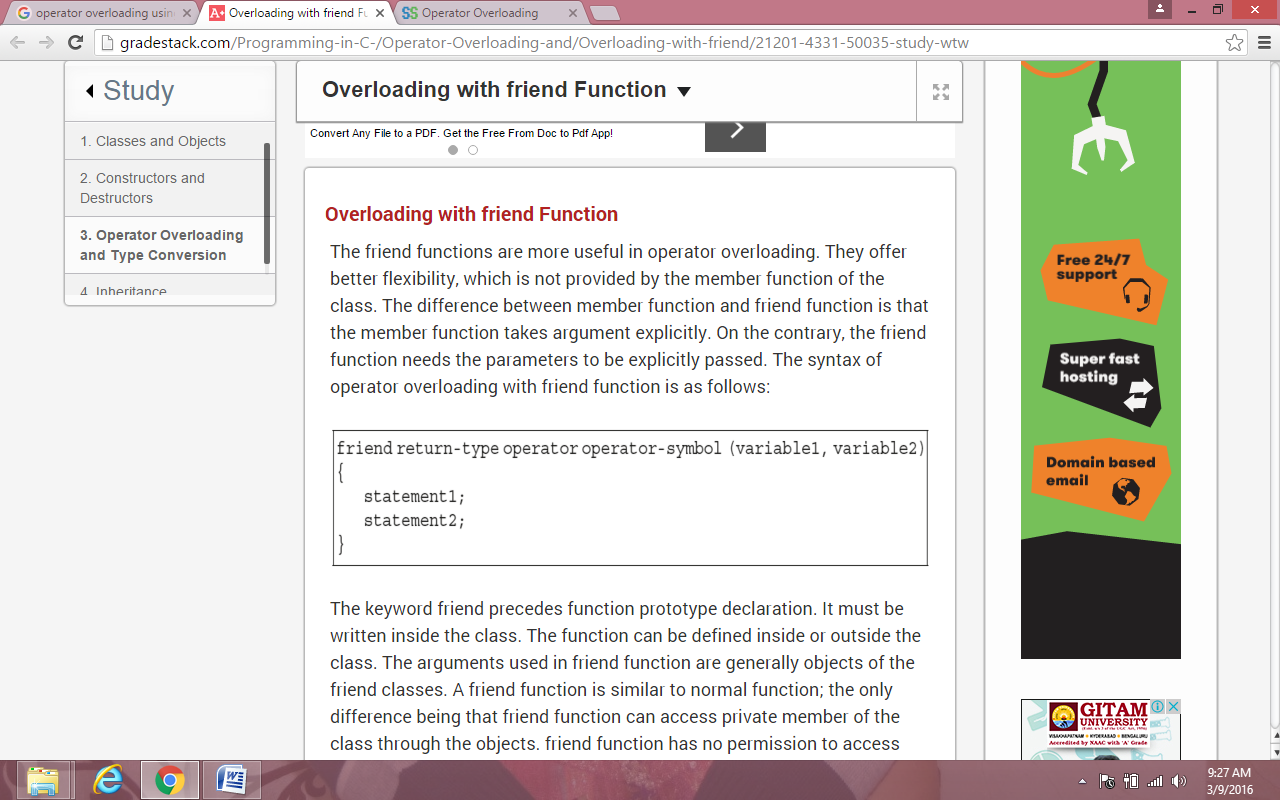
**t3.show();**

**getch();**

**}**

**OPERATOR OVERLOADING USING FRIEND FUNCTIONS:**

**The friend functions are mole useful in operator over loading. They offer better flexibility,which is not provided by the member function of the class. The difference between member function and blend function is that the member function takes argument explicitly. On the contrary, the friend function needs the par ameters to be explicitly passed. The syntax of operator overloading with friend function is as follows:**

****

**TYPE CONVERSION:**

**We learnt that when constants and variables of various data types are clubbed in a single expression, automatic type conversion takes place. This is so for basic data types. The compiler is unknown about the user-defined data type and about their conversion to other data types. The programmer should write the routines that convert basic data type to user-defined data type or vice versa.**

**There are three possibilities of data conversion as given below:**

**a. Conversion from basic data type to user-defined data type (class type)**

**b. Conversion from class type to basic data type**

**c. Conversion from one class type to another class type**

**CONVERSION FROM BASIC TO USER DEFINED TYPE(CLASS):**

**The most famous example, employed to explain conversion between basic and user defined data type is that of meters to distance class; where meter is a float type basic data type and distance is a user defined class. Some programming logic is implemented in order to convert the float to user defined class. You will be amazed to know that we will be able to do something like**

**#include <iostream.h>**

**const float MeterToFloat=3.280833;**

**class Distance {**

**int feets;**

**float inches;**

**public:**

**Distance() //Distance Constructor**

**{**

**feets=0;**

**inches=0.0;**

**}**

**Distance(float numofmeters) //Single Parameter constructor**

**{**

**float feetsinfloat= MeterToFloat \* numofmeters;**

**feets=int(feetsinfloat);**

**inches=12\*(feetsinfloat-feets);**

**}**

**void displaydist() // Method to display converted values**

**{**

**cout<<"Converted Value is: "<<feets<<"\' feets and "<<inches<<'\"'<<" inches.";**

**}**

**};**

**int main()**

**{**

**cout <<"Float to distance conversion.\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";**

**float meters;**

**cout<<"Enter values in meter:";**

**cin >>meters;**

**Distance distance = meters; //Performs type conversion**

**distance.displaydist();**

**}**

**CONVERSION FROM USER DEFINED DATA TYPE TO BASIC DATA TYPE:**

**A whole new concept is involved in conversion from user defined to basic data type, which is known as the overloading casting operator.**

**OVERLOADED CASTING OPERATOR:  
  
Overloaded casting operator actually overloads the built in casting operator. For example, you can achieve different functionality from a float casting operator. Syntax of overloaded casting operator is simple.**

**Code:**

**operator type()**

**{**

**.**

**.**

**.**

**}**

**There are three conditions that need to be satisfied for an overloaded casting operator.**

1. **Overloaded casting operator does not have any return type.**
2. **It cannot take any parameters or in other words no arguments can be passed to an overloaded casting operator.**
3. **Finally, it has to be defined inside a class definition. The class definition will be the user defined data type that we want to convert into a basic type whose casting operator has been overloaded.**

**Example:**

**#include <iostream>**

**const float MeterToFloat=3.280833;**

**// Meter to feet**

**class Distance {**

**int feet;**

**float inches;**

**public:**

**Distance() // Default Constructor {**

**feet=0;**

**inches=0.0;**

**}**

**Distance(int ft, float in) //two arguements constructor {**

**feet=ft;**

**inches=in;**

**}**

**operator float() //overloaded casting operator {**

**float feetinfractions=inches/12;**

**feetinfractions+=float(feet);**

**return (feetinfractions/MeterToFloat);**

**}**

**};**

**int main() {**

**int feet;**

**float inches;**

**cout <<"Enter distance in Feet and Inches.";**

**cout<<"\nFeet:";**

**cin>>feet;**

**cout<<"Inches:";**

**cin>>inches;**

**Distance dist(feet, inches);**

**float meters=dist;**

**// This will call overloaded casting operator**

**cout<<"Converted Distance in Meters is: "<< meters;**

**}**

**Conversion From One Class Type to another Class Type:**

**Now the third method is conversion from class type to another class type. When an object of one class is assigned to object of another class, it is necessary to give clear-cut instructions to the compiler about how to make conversion between these two user-defined data types. The method must be instructed to the compiler. There are two ways to convert object data type from one class to another. One is to define a conversion operator function in source class or a one-argument constructor in a destination class.**

**Consider the following example:**

**X=A;**

**Here, Xis an object of class XYZ and A is an object of class ABC. The class ABC data type is converted to class XYZ. The conversion happens from class ABC to XYZ. The ABC is a source class and XYZ is a destination class.**

**We know the operator function operator data-types. Here, data type may be built-in data type or user-defined data type. In the above declaration, the data type indicates target type of object. Here, the conversion takes place from class ABC (source class) to class XYZ (destination class).**

**PROGRAM 1: Program of Unary Operator Overloading.**

**#include<iostream.h>**

**#include<conio.h>**

**class n**

**{**

**public:**

**int a1,b1;**

**void getdata(inta,int b)**

**{**

**a1=a;**

**b1=b;**

**}**

**void show()**

**{**

**cout<<"a1 : "<<a1<<"\n";**

**cout<<"b1 : "<<b1<<"\n";**

**}**

**void operator - ();**

**};**

**void n::operator - ()**

**{**

**a1=-a1;**

**b1=-b1;**

**}**

**void main()**

**{**

**clrscr();**

**n n1;**

**n1.getdata(10,20);**

**n1.show();**

**- n1;**

**n1.show();**

**}**

**OUTPUT:**

**PROGRAM 2: Program of Binary Operator Overloading.**

**#include<iostream.h>**

**#include<conio.h>**

**class a**

**{**

**public:**

**inti,j;**

**void set()**

**{**

**cout<<"Enter any two values:"<<"\n";**

**cin>>i>>j;**

**}**

**void get()**

**{**

**cout<<i<<"i"<<"+"<<j<<"j"<<"\n";**

**}**

**a operator + (a b1);**

**}**

**aa :: operator + (a k1)**

**{**

**a b;**

**b.i=k1.i+i;**

**b.j=k1.j+j;**

**return b;**

**}**

**void main()**

**{**

**clrscr();**

**a a1,b1,c1;**

**a1.set();**

**b1.set();**

**c1=a1+b1;**

**a1.get();**

**b1.get();**

**c1.get();**

**getch();**

**}**

**OUTPUT:**

**PROGRAM 3: Program of Basic To Class Type Conversion.**

**#include<iostream.h>**

**#include<conio.h>**

**class time**

**{**

**public:**

**int h,m;**

**time(int d)**

**{**

**h=d/60;**

**m=d%60;**

**}**

**};**

**void main()**

**{**

**int dur=65;**

**time t1=dur;**

**cout<<"h"<<t1.h<<"m"<<"\n"<<t1.m<<"\n";**

**getch();**

**}**

**OUTPUT:**

**PROGRAM 4: Program of Class To Class Type Conversion.**

**#include<iostream.h>**

**#include<conio.h>**

**class inv2;**

**class inv1**

**{**

**public:**

**int code,item,price;**

**inv1(int c,int i,int p)**

**{**

**code=c;**

**item=i;**

**price=p;**

**}**

**void putdata()**

**{**

**cout<<"code:"<<code<<"\n";**

**cout<<"item:"<<item<<"\n";**

**cout<<"price:"<<price<<"\n";**

**}**

**int getcode(){return code;}**

**int getitem() {return item;}**

**int getprice() {return price;}**

**};**

**class inv2**

**{**

**public:**

**int code,t\_price;**

**inv2(inv1 i1)**

**{**

**code=i1.getcode();**

**t\_price=i1.getitem()\*i1.getprice();**

**}**

**void putdata()**

**{**

**cout<<"code:"<<code<<"\n";**

**// cout<<"item:"<<item<<"\n";**

**cout<<"price:"<<t\_price<<"\n";**

**}**

**};**

**void main()**

**{**

**inv1 i1(3,5,30);**

**i1.putdata();**

**inv2 i2=i1;**

**getch();**

**}**

**OUTPUT:**