Scope Resolution Operator object as Function Argument

Lecture 5

CLASS WORK

The Scope Resolution Operator

- The concept of classes adds **new scope rules** to those of the C++ language.
- You remember that one point of classes is to provide an **encapsulation technique**
- It makes sense that all names declared within a class be treated within their own scope as distinct from external names, function names, and other class names.
- This creates the need for the **scope resolution operator.**

CLASS WORK

Tahir Nawaz
Assistant Professor
UOL Sargodha Campus

Syntax of Scope resolution operator

- Return data type [class name]:: function name(argument list)
- e.g
  - Void test::hello();

CLASS WORK

Scope resolution operator

CLASS WORK

```
#include <iostream>
using namespace std;
/* run this program using the console pauser or add your own getch, system("pause") or input loop */
class test{
        private:
        public:
                void hello(void);
};
                void test::hello(void)
                { std::cout<<"Hello"<<endl; }
int main(int argc, char *argv[]) {
test obj;
obj.hello();
system("PAUSE");
return EXIT_SUCCESS;
}
```

Tahir Nawaz
Assistant Professor
UOL Sargodha Campus

Scope Resolution Operator

```
#include <cstdlib>
#include <iostream>
using namespace std;
class temp
{ private:
        int t;
        public:
              temp(){};
              temp(int x): t(x)
              { }
                    void add(temp);
      void show()  {
          cout<<"Value in T variable"<<t<<endl;
       }      };
```

CLASS WORK

Tahir Nawaz
Assistant Professor
UOL Sargodha Campus

Scope Resolution Operator

```
void temp::add(temp num1)
      {
          t=num1.t*num1.t;
       }
int main(int argc, char *argv[])
{
    temp obj;
    obj.add(5);
    obj.show();
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

CLASS WORK

Tahir Nawaz
Assistant Professor
UOL Sargodha Campus

Passing object As Arguments

- Objects can also be passed as arguments to member functions. When an object is passed as an argument to a member function.
  - Only the name of the object is written in the argument list
  - The number of parameters and their types must be defined in the member function to which the object is to be passed.
  - the objects that are passed are treated local for the number function and are destroyed when the control returns to the calling function.

CLASS WORK

Tahir Nawaz
Assistant Professor
UOL Sargodha Campus

Passing object As Arguments

CLASS WORK

```
#include <iostream.h>
class test{
private:
Char name[15];
public:
Void get (void)
{ cout<<“Enter your name”<<endl;
Cin>>name;
}
Void display(test s)
{
Cout<<“Your entered name is”<<s.name<<endl;
}
};
```

Tahir Nawaz
Assistant Professor
UOL Sargodha Campus

Passing object As Arguments

CLASS WORK

```
int main(int argc, char *argv[])
{
test temp, obj;
temp.get();
obj.display(temp);
system("PAUSE");
    return EXIT_SUCCESS;
}
```

Tahir Nawaz
Assistant Professor
UOL Sargodha Campus

Passing and Returning object to Functions as Arguments

CLASS WORK

```
#include <iostream>
using namespace std;
class Distance {
private:
int feet;
float inches;
public:
Distance() : feet(0), inches(0.0){ }
Distance(int ft, float in) : feet(ft), inches(in)
{ }
void getdist() {
cout << “\nEnter feet: “;
cin >> feet;
cout << “Enter inches: “;
cin >> inches;
}
```

Tahir Nawaz
Assistant Professor
UOL Sargodha Campus

Passing and Returning object to Functions as Arguments

```
void showdist()
{ cout << feet << "\'-" << inches << '\"'; }
Distance add_dist(Distance);
};
Distance Distance::add_dist(Distance d2)
{
Distance temp;
temp.inches = inches + d2.inches;
if(temp.inches >= 12.0)
{
temp.inches -= 12.0;
temp.feet = 1;
}
temp.feet += feet + d2.feet;
return temp;
}
```

CLASS WORK

Tahir Nawaz
Assistant Professor
UOL Sargodha Campus

Passing and Returning object to Functions as Arguments

```
int main(int argc, char *argv[])
{
Distance dist1, dist3;
Distance dist2(11, 6.25);
dist1.getdist();
dist3 = dist1.add_dist(dist2);
cout << “\ndist1 = “; dist1.showdist();
cout << “\ndist2 = “; dist2.showdist();
cout << “\ndist3 = “; dist3.showdist();
cout << endl;
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

CLASS WORK

Four main member Functions in Classes

- There are four member functions that the C++ compiler “expects” to be in every class.
  - Default Constructor
  - Destructor
  - Copy Constructor
  - Assignment Operator

If programmer does not write these member functions now-a-days C++ compiler will automatically generates them.

CLASS WORK

Tahir Nawaz
Assistant Professor
UOL Sargodha Campus

Example

```
#include<iostream.h>
Class memberfunc{
private:
Int x;
public:
Void input(int a)
{x=a;}
void output()
{
cout<<x<<endl;
}};
```

CLASS WORK

Tahir Nawaz
Assistant Professor
UOL Sargodha Campus

```
int main(int argc, char *argv[])
{
memberfunc obj1;
obj1.input(5);
memberfunc obj2=obj1;
obj1.output();
obj2.output();
Obj2.input(90);
Obj1=obj2;
Obj1.output();
Obj2.ouput();
 system("PAUSE");
    return EXIT_SUCCESS
}
```

Quiz 2% marks

CLASS WORK

- Write a program to find the volume of a cylinder using class which have overloaded constructors and two function named input, display.
- **The formula to calculate the volume of a cylinder is**
  - [Radius * Radius * Height * Pi]

Tahir Nawaz
Assistant Professor
UOL Sargodha Campus

Fundamentals of Operator Overloading

- Operator overloading is not automatic we must write operator-overloading functions to perform the desired operations
- An operator is overloaded by writing a non-static member function definition as we normally would, except that the function name starts with the keyword *operator* followed by the symbol for the operator being overloaded.

CLASS WORK

Tahir Nawaz
Assistant Professor
UOL Sargodha Campus

Fundamentals of Operator Overloading

- For example, the function name operator + would be used to overload the addition operator (+) for use with objects of a particular class. When operators are overloaded as member functions, they must be non-static, because *they must be called on an object of the class* and operate on that object.

CLASS WORK

Tahir Nawaz
Assistant Professor
UOL Sargodha Campus

Fundamentals of Operator Overloading

- To use an operator on an object of a class, the operator must be overloaded for that class—with three exceptions:
- The assignment operator (=) may be used with *every* class to perform *memberwise assignment* of the class's data members—each data member is assigned from the assignment's "source" object (on the right) to the "target" object (on the left). *Memberwise assignment is dangerous for classes with pointer members*, so we'll explicitly overload the assignment operator for such classes.

CLASS WORK

Tahir Nawaz
Assistant Professor
UOL Sargodha Campus

Fundamentals of Operator Overloading

- The address (&) operator returns a pointer to the object; this operator also can be overloaded.
- The comma operator evaluates the expression to its left then the expression to its right, and returns the value of the latter expression. This operator also can be overloaded.

CLASS WORK

Tahir Nawaz
Assistant Professor
UOL Sargodha Campus

Restrictions on Operator Overloading

- C++ operators that can be overloaded

Operators that can be overloaded							
+	-	*	/	%	^	&	|
~	!	=	<	>	+=	-=	*=
/=	%=	^=	&=	|=	<<	>>	>>=
<<=	==	!=	<=	>=	&&	||	++
--	->*	,	->	[]	()	new	delete
new[]	delete[]						

- C++ Operators that cannot be overloaded

Operators that cannot be overloaded				
.	.*	::	?:	sizeof

Rules and Restrictions on Operator Overloading

- As we prepare to overload operator with our own classes, there are different rules and restrictions we should need to keep in mind before overloading.
- *The precedence of an operator cannot be changed by overloading.* However, parentheses can be used to *force* the order of evaluation of overloaded operators in an expression.

CLASS WORK

Rules and Restrictions on Operator Overloading

- *The associativity of an operator cannot be changed by overloading*—if an operator normally associates from left to right, then so do all of its overloaded versions.
- *You cannot change the "arity" of an operator* (that is, the number of operands an operator takes)—overloaded unary operators remain unary operators; overloaded binary operators remain binary operators. Operators &, *, + and
  - all have both unary and binary versions; these unary and binary versions can be separately overloaded.

CLASS WORK

Rules and Restrictions on Operator Overloading

CLASS WORK

- *We cannot create new operators; only existing operators can be overloaded*.
- The meaning of how an operator works on values of fundamental types *cannot* be changed by operator overloading.
  - For example, you cannot make the + operator subtract two integers. Operator overloading works only with *objects of user-defined types or with a mixture of an object of a user-defined type and an object of a fundamental type.*

Tahir Nawaz
Assistant Professor
UOL Sargodha Campus

Rules and Restrictions on Operator Overloading

- Related operators, like + and +=, must be overloaded separately.
- When overloading (), [], -> or any of the assignment operators, the operator overloading function *must* be declared as a class member. For all other over-loadable operators, the operator overloading functions can be member functions.

Tahir Nawaz
Assistant Professor
UOL Sargodha Campus

Types of Operator

- Unary operator
- Binary operator

CLASS WORK

Tahir Nawaz
Assistant Professor
UOL Sargodha Campus

Unary Operators

- Unary operators act on only one operand. (An operand is simply a variable acted on by an operator.)
  - Examples of unary operators are the increment and decrement operators ++ and --, and the unary minus, as in -33.
  - Operators attached to a single operand
    - (-a, +a, --a, a--, ++a, a++)

CLASS WORK

Tahir Nawaz
Assistant Professor
UOL Sargodha Campus

Unary Operator Overloading example 1

```
#include<iostream.h>
#include<conio.h>

class operatorincreament{
int i;
public:
operatorincreament (int a){
i=a;
}
void operator ++(){
i++ ;
}
void print(){
cout<< i<<endl;
}
};
```

CLASS WORK

Tahir Nawaz
Assistant Professor
UOL Sargodha Campus

(Cont) unary operator

```
 int main(int argc, char *argv[])
{
clrscr();
        operatorincreament obj(0);
        obj.print();
        for(int i=0;i<=5;i++){
        obj++;
        obj.print();  }
        system("PAUSE");
        return 0;
}
//-------------------------------------------------------------------
```


Tahir Nawaz
Assistant Professor
UOL Sargodha Campus

Example 2 Unary operator

```
#include<iostream.h>
#include<conio.h>
 class signedop{
    private:
        int a;
        int b;
    public:
        void getdata(int x, int y)  {
            a = x; b = y;
        }
        void show_data() {
            cout<< endl << "A = "<< a <<endl<<"B = " << b;
        }
        void operator -();
        };
       void signedop ::  operator -(){
            a = -a; b = -b;
        }
```

CLASS WORK

Tahir Nawaz
Assistant Professor
UOL Sargodha Campus

(Cont) unary operator

```
int main(int argc, char *argv[])
{
clrscr();
    signedop signedobj;
    int a,b;
    cout<<"Enter Ist Value"<<endl;
    cin>>a;
    cout<<"Enter 2nd Value"<<endl;
    cin>>b;
    signedobj.getdata(a,b);
    cout << endl << "Signed Object Before Operator Applied : ";
    signedobj.show_data();
    -signedobj;
    cout << endl << "Signed Obect Data After Operator Applied : ";
    signedobj.show_data();
    system("PAUSE");
    return 0;
}
```

CLASS WORK

Tahir Nawaz
Assistant Professor
UOL Sargodha Campus