Chapter 10

Namespace

The **namespace** keyword allows you to partition the global namespace by creating a declarative region. In essence, a **namespace** defines a scope. The general form of **namespace** is shown here:

```
namespace name
{
        // declarations
}
```

Anything defined within a **namespace** statement is within the scope of that namespace.

There is one difference between a class definition and a namespace definition: The namespace is concluded with a closing brace but no terminating semicolon.

Example:

```
namespace A
{
        int m;
        void display(int n)
        {
        cout<<n;
        }
}
```

using namespace A;
m=100; //OK
display(200); //OK
	using namespace A::m;
	m=100; //OK
	display(200); // NOT OK, display is not visible

In general, to access a member of a namespace from outside its namespace, precede the member's name with the name of the namespace followed by the scope resolution operator.

Here is a program that demonstrates the use of **CounterNameSpace**.

```
// Demonstrate a namespace.
#include <iostream.h>
namespace CounterNameSpace
{
        int upperbound;
        int lowerbound;
        class counter
        {
                int count;
                public:
                counter(int n)
                {
                if(n <= upperbound) count = n;
                else count = upperbound;
                }
```

```
                void reset(int n)
                {
                if(n <= upperbound) count = n;
                }
                int run()
                {
                if(count > lowerbound) return count--;
                else return lowerbound;
                }
        };
}
void main()
{
        CounterNameSpace::upperbound = 100;
        CounterNameSpace::lowerbound = 0;
        CounterNameSpace::counter ob1(10);
        int i;
        do
        {
        i = ob1.run();
        cout << i << " ";
        } while(i > CounterNameSpace::lowerbound);
        cout << endl;
        CounterNameSpace::counter ob2(20);
        do
        {
        i = ob2.run();
        cout << i << " ";
        } while(i > CounterNameSpace::lowerbound);
        cout << endl;
        ob2.reset(100);
        CounterNameSpace::lowerbound = 90;
        Do
         {
        i = ob2.run();
        cout << i << " ";
        } while(i > CounterNameSpace::lowerbound);
}
```

Notice that the declaration of a **counter** object and the references to **upperbound** and **lowerbound** are qualified by **CounterNameSpace**. However, once an object of type **counter** has been declared, it is not necessary to further qualify it or any of its members. Thus, **ob1.run()** can be called directly; the namespace has already been resolved.

using

As you can imagine, if your program includes frequent references to the members of a namespace, having to specify the namespace and the scope resolution operator each time you need to refer to one quickly becomes a tedious chore. The **using** statement was invented to alleviate this problem. The **using** statement has these two general forms:

using namespace name;

using name::member;

In the first form, name specifies the name of the namespace you want to access. All of the members defined within the specified namespace are brought into view (i.e., they become part of the current namespace) and may be used without qualification. In the second form, only a specific member of the namespace is made visible. For example, assuming **CounterNameSpace** as shown above, the following **using** statements and assignments are valid.

```
using CounterNameSpace::lowerbound; // only lowerbound is visible
lowerbound = 10; // OK because lowerbound is visible
using namespace CounterNameSpace; // all members are visible
upperbound = 100; // OK because all members are now visible
```

Unnamed Namespaces

There is a special type of namespace, called an unnamed namespace that allows you to create identifiers that are unique within a file. Unnamed namespaces are also called anonymous namespaces. They have this general form:

```
namespace
{
// declarations
}
```

Unnamed namespaces allow you to establish unique identifiers that are known only within the scope of a single file. That is, within the file that contains the unnamed namespace, the members of that namespace may be used directly, without qualification. But outside the file, the identifiers are unknown. Unnamed namespaces eliminate the need for certain uses of the **static** storage class modifier.

For example, consider the following two files that are part of the same program.

File One

```
static int k;
void f1() {
k = 99; // OK
}
```

File Two

```
extern int k;
void f2() {
k = 10; // error
}
```

Because k is defined in File One, it may be used in File One. In File Two, k is specified as extern, which means that its name and type are known but that k itself is not actually defined. When these two files are linked, the attempt to use k within File Two results in an error because there is no definition for k. By preceding k with static in File One, its scope is restricted to that file and it is not available to File Two. While the use of static global declarations is still allowed in C++, a better way to accomplish the same effect is to use an unnamed namespace. For example:

File One

```
namespace
{
int k;
}
void f1()
{
k = 99; // OK
}
```

File Two

```
extern int k;
void f2()
{
k = 10; // error
}
```

Here, **k** is also restricted to File One. The use of the unnamed namespace rather than **static** is recommended for new code.

Some Namespace Options

There may be more than one namespace declaration of the same name. This allows a namespace to be split over several files or even separated within the same file.

For example:

```
#include <iostream.h>
namespace NS
{
        int i;
}
namespace NS
{
        int j;
}
void main()
{
        NS::i = NS::j = 10;
        // refer to NS specifically
```

```
        cout << NS::i * NS::j << "\n";
        // use NS namespace
        using namespace NS;
        cout << i * j;
        return 0;
}
```

Output:

100
100

Here, NS is split into two pieces. However, the contents of each piece are still within the same namespace, that is, NS.

The std Namespace

Standard C++ defines its entire library in its own namespace called **std**. This is the reason that most of the programs in this book include the following statement:
using namespace std;
This causes the **std** namespace to be brought into the current namespace, which gives you direct access to the names of the functions and classes defined within the library without having to qualify each one with **std::**. Of course, you can explicitly qualify each name with **std::** if you like. For example, the following program does not bring the library into the global namespace.

// Use explicit std:: qualification.

```
#include <iostream>
void main()
{
        int val;
        std::cout << "Enter a number: ";
        std::cin >> val;
        std::cout << "This is your number: ";
        std::cout << std::hex << val;
}
```

Here, **cout**, **cin**, and the manipulator **hex** are explicitly qualified by their namespace. That is, to write to standard output, you must specify **std::cout**; to read from standard input, you must use **std::cin**; and the hex manipulator must be referred to as **std::hex**.

Creating Conversion Functions

In some situations, you will want to use an object of a class in an expression involving other types of data. Sometimes, overloaded operator functions can provide the means of doing this. However, in other cases, what you want is a simple type conversion from the class type to the target type. To handle these cases, C++ allows you to create custom conversion functions. A conversion function converts your class into a type compatible with that of the rest of the expression. The general format of a type conversion function is:

operator type( ) { return value; }

Here, type is the target type that you are converting your class to, and value is the value of the class after conversion. Conversion functions return data of type type, and no other return type specifier is allowed. Also, no parameters may be included. A conversion function must be a member of the class for which it is defined. Conversion functions are inherited and they may be virtual.

```
#include <iostream.h>
const int SIZE=100;
class stack
{
        int stck[SIZE];
        int tos;
        public:
        stack() { tos=0; }
        void push(int i);
        int pop(void);
        operator int() { return tos; } // conversion of stack to int
};
void stack::push(int i)
{
        if(tos==SIZE)
        {
        cout << "Stack is full.\n";
        return;
        }
        stck[tos] = i;
        tos++;
}
int stack::pop()
{
        if(tos==0)
         {
        cout << "Stack underflow.\n";
        return 0;
        }
```

```
        tos--;
        return stck[tos];
}
void main()
{
        stack stck;
        int i, j;
        for(i=0; i<20; i++) stck.push(i);
        j = stck; // convert to integer
        cout << j << " items on stack.\n";
        cout << SIZE - stck << " spaces open.\n";
}
```

Output:

```
20 items on stack.
80 spaces open.
```

As the program illustrates, when a stack object is used in an integer expression, such as j = stck, the conversion function is applied to the object. In this specific case, the conversion function returns the value 20. Also, when stck is subtracted from SIZE, the conversion function is also called.

Assignment 10

Short Type Questions

1. Define namespace.
2. Define anonymous namespace.
3. How the member of a namespace can be accessed?

Long Type Questions

1. Suppose there is a class X with a double type attribute. Write a c++ program to create three objects named as ob1, ob2 and ob3 of the above class and perform the following operation:
   Ob2 = 5.5 + ob 1;
   Ob 3 = ob1 + 6.7;
2. Write a complete program where you can perform the following conversion within main() :
   A ob 1 (5.5), ob2;
   int i=10;
   double d=ob 1;
   ob2=i; where A is a user defined class.
3. Discuss the use of std namespace.
4. Write a C++ program to convert a primitive or built in type value such as int or char to a user defined type value such as class A or class X.

BPUT 3rd Semester Examination 2012, BRANCH: CSE/ EEE

1. Answer the following questions:

a. Find the output from the following C++ program

```
#include<iostream.h>
int a=20;
int main()
{
        int a=5;
        {
                inta=10;
                cout<<a<<::a;
        }
cout<<::a<<a;
}
```

b. Find the output if any from the following C++ code

```
int a[]={1,5,9};
int *p=a;
cout<<*p+1<<*++p<<++*p<<*(p+1);
```

c. State the difference between a macro and and an inline function.

d. Differentiate between class and structure.

e. Define a destructor. Does it always require a destructor in a C++ program?

f. What is friend function? Mention its main disadvantage with respect to object oriented programming.

g. Differentiate method over loading and method overriding.

h. Mention the importance of this pointer by using an example.

i. What is object copying ?Why is it required to pass an object as reference parameter during object copying?

j. How can you use the concept of object oriented programming?

2. (a) Briefly explain the different concepts of object oriented programming.

(b) Write a C++ program to generate the following output:

```
                1
            1       1
        1       2       1
    1       3       3       1
1       4       6       4       1
1       5       10      10      5       1
```

3. (a) Write a C++ program using class to reverse a string.
   (b) Show the use of different types of constructors with a suitable C++ program.

4. (a) Define inheritance .Briefly explain the different types of inheritance associated with C++.
   (b) Write a C++ program using friend function to overload - operator so that it can subtract two complex numbers.

5. (a) Illustrate the use of virtual function in handling the ambiguity raised during inheritance.
   (b) Briefly explain the exception handling mechanism used in C++.

6. (a) Write a function template to find the binary equivalent of a decimal number.

   (b) Mention the different roles of classes during design of a program.

7. (a) Mention the class hierarchy associated with iostreams.

   (b) Write a program that creates a binary file "BPUT.text" and then writes records of five students in to it. (The attributes of students are name, roll number and total marks obtained).

8. Write short notes on any TWO of the following:

   (a) Recursion.
   (b) Generic classes.
   (c) Namespaces.

****

BPUT 3rd Semester Examination 2011, BRANCH: CSE/ EEE

1. Answer the following Question:

(a) Write the output of the following sets of C++ statements

```
void main()
{
unsigned i=1;
signed j=-1;
If(i<j)
cout<<"less";
else
if(i>j)
cout<<"greater";
else
if(i==j)
cout<<"equal";
}
```

(b) How many copies of a class static member are shared between objects of the class ?

(c) What are the benefits of using ADTs?

(d) What is the wrong with the following set of statements?

```
#include"iostream.h"
class  my_class {
int  I
public:
. . . .
. . . .
. . . .
};
int main()
{
my_class obj;
obj.I=15;
. . . .
. . . .
. . . .
}
```

(e) Find out the error in the following set of statements:

```
class  my_class {
double a,b,c;
double my_class();
```

};

(f) _____ provides facilities for organizing the names in a programe to avoid name clashes.

(g) Differentiate between friend and inheritance.

(h) _____ keywords supports dynamic method resolution.

(i) What is the implicit pointer that is passed as the first argument for non-static member functions?

(j) _____ is the most general exception handler that catches exception of any type

.

2. Write a C++ program using class and object with constructor to convert the Fahrenheit to Celsius and vice versa.

3. (a) Discuss the different forms of constructor with examples.
(b) State any conflict that may arise due to multiple inheritances .Justify your answer.

4. (a) Can we use compiler generated default assignment operator in case our class is using dynamic memory ? Justify your answer?
(b) Write short notes on try, throw and catch block in C++.

5. What is virtual function? When do we make a virtual function pure? What is the implication of making a pure virtual function? Explain with suitable examples.

6. Write a program using pure virtual function to find out the area of circle, tringle and square.

7. Write a program to read two double type numbers from keyboard and a function to calculate the division of these two numbers. A try block to throw an exception when wrong type of data is entered and another try Block to throw an exception if the condition "division "occurs". Appropriate catch block to handle the exception thrown.

8. (a) Differentiate between class template and template class with examples.
(b) Write short notes on STL.

****

BPUT 3rd Semester Examination 2010, BRANCH: CSE/ IT

1. Answer the following questions:

(a)What will be the output and why?

```
#include<iostream.h>
using namespace std;
 int main()
{        const int a=20;
          const int *ptr=&a;
          cout<<*ptr<<endl;
          (*ptr)++;
          cout<<a<<endl;
          return 0;
}
```

(b)State at least two differences between an inline function and a macro substitution.

(c)What will be the output and why?

```
#include<iostream.h>
using namespace std;
class X
        {  int i;
        public:
                static void f1()
                { i=20; i++;}
                Void f2()
                {cout<<"i="<<endl;}
        };
        int main()
        {
                X ob1 , ob2;
                ob1:f1();
                ob2:f2();
                return 0;
        }
```

(d)Explain the two primary differences between a pointer variable and a reference variable.

(e)What will be the output and why?

```
#include<iostream.h>
using namespace std;
class X
{
        public:
        int p;
```

```
};
class Y:protected X
{
        public:
        int q;
};
        int main()
        {
        Y yob;
        yob.p=10;
        yob .q=20;
        cout<<yob.p<<" "<<yob.q<<endl;
        return 0;
}
```

(f)What will be the output and why?

```
#include<iostream.h>
using namespace std;
class X
{
        public:
        void f1()
        {cout<<"x-Men"<<endl;
]
virtual void f2()=0;
};
class Y:public X
{
        public:
        void f1()
        {cout<<"Y-Men"<<endl;}
};
        int main()
        {
        X *xp;
        Y yob;
        xp=&yob;
        xp->f1();
        return 0;
}
```

(g)What will be the output and why?

```
#include<iostream.h>
using namespace std;
class X
{
        int i;
        public:
        static void f(int i){ this->i=I;}
};
        int main()
        {
        X xob;
        xob.f(20);
        return 0;
        }
```

(h)What will be the output and why?

```
        #include<iostream.h>
        using namespace std;
        class X
        {
                const int i;
                public:
                X(int p){i=p;}
                void f(){cout<<"i="<<endl;}
        };
                int main()
                {
                X xob(100);
                xob.f();
                return 0;
                }
```

(i)What will be the output and why?

```
#include<iostream.h>
using namespace std;
int main()
{
        cout<<"Inside main"<<endl;
        try
        {
        cout<<"Inside try block"<<endl;
        throw 22;
        cout<<"Exception thrown"<<endl;
```

```
        }
    catch(double d)
    {
        cout<<"caught an exception="<<d;
    }
    cout<<"End"<<endl;
    return 0;
    }
```

(j)What will be the output and why?

```
#include<iostream.h>
using namespace std;
template<class T1 void f(T1 tt1)
{ cout<<"tt1="<<tt1<<endl;}
template<class  T2> void f(T2 tt2)
{ cout<<"tt2="<<tt2<<endl;}
int main()
{
f(1000);
        f(101.7);
        return 0;
}
```

2. (a)Create a class called **student** which contains protected attributes such as stud_name , stud_branch. Provide an appropriate method to take user input to initialize these attributes and display the details regarding 50 students of a class.

   (b)Create a class called **Area** which contains a method called "find_area". Write down appropriate code to create objects named as **circle** and **rectangle** of the above class and implement **function overloading** to calculate area of a rectangle and area of a circle based upon user input.

3. (a)Write a C++ program to convert a primitive or built in type value such as int or char to a user defined type value such as class A or class X.

   (b)Write a C++ program where you can only be able to create a single instance of a class. Upon trying to create more than one number of instances will result in termination of a program.

4. (a)Create a class called point with two integer attributes such as x and y to represent its x-coordinate and y-coordinate. Provide constructor to initialize the attributes. Provide another method named as move () which will move the coordinates only in the direction of x-axis for 10 unit at a time. Also display the new and old values of the coordinates.

(b)With an appropriate example, explain the role of virtual base class in removing ambiguities in case of diamond inheritance which is a special case of multi path inheritance.

5. (a)Create an abstract class called **Figure** which contains a pure virtual function called find_area() and a protected attribute named as area. Create two new derived classes from the above class named as **Circle** and **Square** having double type attribute named as radius and side respectively. Implement **dynamic polymorphism** to find out area of a circle and a square, and show the result.

   (b)Write appropriate code to overload the pre increment and post increment operators in a same program using non member operator functions.

6. (a)Suppose there is a class called **X** with a double type attribute. Write a C++ program to create three objects named as ob 1, ob 2 and ob 3 of the above class and perform the following operations:
   ob2=5.5 + ob1
   ob3=ob1 + 6.7

   (b)Write a complete program to create a class named **Student** with protected attributes such as id and marks. The attributes should be initialized only through constructors. The class contains a public method named as show() to display the initialized attributes. Provide a mechanism to create an array of student objects. The array size should be given by the user at run-time.

7. (a)Provide at least one good reason to choose a **const reference** rather than a reference variable as a parameter to a **copy constructor**. Explain with a suitable example.

   (b)Write a complete program where you achieve the following conversion within main():

   A ob1(10), ob2(101.5);
   int i=ob1;
   double d=ob2;
   Where **A** is a user defined class .

8. (a)Write a complete program where you can restrict a user defined function to throw only int or char type exception out of it.

   (b)Write a complete program to declare and define a generic function that is capable of arranging any kind of elements in descending order.

****

BPUT 3rd Semester Examination 2010, BRANCH: EEE/ CIVIL

1. Answer the following questions:

(a) What will be the output and why?

```
#include<iostream.h>
using namespace std;

class X
        {
        int p;
        public:
                X(int a){p=a;}
                void f(){cout<<"p="<<p<<endl;}
        };
        int main()
        {
                X xob;
                xob.show();
                return 0;
        }
```

(b) State the differences between the following two declarations with examples:
Const int *p; and int *const p;

(c) What will be the output and why?

```
#include<iostream.h>
        using namespace std;
        class A
        { int i;
                public:
                static void f() {i=20;i++;}
        };
        int main()
        { A ob 1 , ob2;
                ob 1.f();
                ob2.f();
                return 0;
        }
```

(d) Private attributes can't be inherited. State a remedy for this problem so that attributes of a class behave like private attributes but can be inherited. Explain with an example.

(e) What will be the output and why?

```
        using namespace;
        struct X
        {int p;};
        struct Y:protected X
        {int q;};
        Int main()
        {
```

```
        Y yob;
        yob.p=11;
        yob.q=22;
        cout<<yob.p<<" "<<yob.q<<endl;
        return 0;
}
```

(f) What will be the output and why?

```
using namespace std;
class X
{
        int p;
        public:
                friend void f(int a);
};
void f(int a)
{
        P=a;
        cout<<"p="<<p<<endl;
}
int main()
{
        f(200)
        return 0;
}
```

(g) What will be the output and why?

```
using namespace std;
class X
{
        int a
        public:
                X(int p){a=p;}
};
class Y
{
        public:
                int b;
};
        public:
                D(int k){b=k;}
                void show(){cout<<"b="<<b<<endl;}
};
int main()
{
        D dob(100);
        dob.show();
        return 0;
```

```
}
```

(h) State at least two differences between new and malloc.

(i) What will be the output and why?

```
using namespace std;
class X
{
        int i;
        public:
                X(int a){i=a;}
                        X(X &copy)){i=copy.i;}
                        void f() {cout<<"i="<<endl;}
};
int main()
{
        X const xob1(10);
        X xob2=xob1;
        xob2.f()
        return 0;
}
```

(j) What will be the output and why?

```
using namespace std;
void f(int a) throw()
{
                if (a==1)throw 100;
                if (a==2)throw 20.5;
}
int main()
{
                cout<<"inside main"<<endl;
                        try
                        {
                                cout<<"inside try block"<<endl;
                                        F(2);
                        }
                        catch(int i){cout<<"i="<<i<<endl;}
                        catch(double d){cout<<"d="<<endl;}
return 0;
}
```

2. (a) Create a class called **Employee** which contains protected attributes such as emp_id, emp_salary and emp_da. emp_da is 20% of the emp_salary. Provide an appropriate method to take user input to initialize the attributes and display the details regarding 25 students of a class.

(b) Create a class called **Volume** which contains a method called "find_vol". write down appropriate code to create objects named as **sphere** and **cylinder** of the above class and implement **function overloading** to calculate volume of a sphere and cylinder based upon user input.

3. (a) Write a C++ program to convert a primitive or built in type value such as int or char to a user defined type value such as class A or class X.

   (b) Write a C++ program where you can only be able to create a single instance of a class. Upon trying to create more than one number of instances will result in termination of the program.

4. (a) With an appropriate example, explain how ambiguities can be resolved for public and protected attributes in case of multi path inheritance without using virtual base class.

   (b) Create an abstract class called **Shape** which contains a pure function called find_vol() and a protected attribute named as volume. Create two new derived classes from the above class named as **Cube** and **Sphere** having double type attribute named as side and radius respectively. Implement **dynamic polymorphism** to find out volume of a cube and a sphere. Also display the result.

5. (a) Write appropriate code to overload the bit-wise << operator to perform the following task:

   int y= x<<ob;

   where ob is an object of class **myclass** with a single int attribute and x is an int declared in main() and its value is to be supplied by the user.

   (b) Suppose there is a class called X with two double type attributes. Write a c++ program to create two objects named ob 1 and ob 2 of the above class and overload the binary == operator to perform the following operation within main():

   ```
   if(ob 1== ob 2)
   cout<<"Objects are same"<<endl;

   else
   cout<<"Objects are different"<<endl;
   ```

6. (a) Suppose there is a class X with a double type attribute. Write a c++ program to create three objects named as ob1, ob2 and ob3 of the above class and perform the following operation:

   ```
   Ob2 = 5.5 + ob 1;
   Ob 3 = ob1 + 6.7;
   ```

(b) Write an appropriate C++ code showing ambiguity resolving mechanism where a class attribute has same name as that of a local parameter of a member by using this pointer.

7. (a) Write a complete program to create a class called **Account** with protected attributes such as account number and balance. The attributes should be initialized through constructors. The class contains a public method named as show () to display the initialized attributes. Provide a mechanism to create an array of **Account** objects. The array size should be given by the user at run time.

(b) What is a **copy constructor**? Explain the role of a **copy constructor** while initializing a pointer attribute of a class for which the memory allocation takes place at the run time.

8. (a) Write a complete program where you can perform the following conversion within main() :

```
A ob 1 (5.5), ob2;
int i=10;
double d=ob 1;

ob2=i; where A is a user defined class.
```

(b) Write a complete program where an **exception** of type double can be **rethrown**.

****