CBCS Scheme

USN | | | | | | | | | | |

16MCA22

Second Semester MCA Degree Examination, June/July 2017

Object Oriented Programming Using C++

Time: 3 hrs. Max. Marks: 80

Note: Answer FIVE full questions, choosing one full question from each module.

Module-1

1 a. Explain the five basic data types of C++ along with their size in bits. **(04 Marks)**

b. What are const and volatile qualifiers? Give an example. **(04 Marks)**

c. Write a program to implement stack operations. Use constructor and destructor for the stack class. **(08 Marks)**

OR

2 a. Explain the four principles of object oriented programming. **(04 Marks)**

b. Write a program which overloads a function for calculating the perimeter of geometrical figures like square, rectangle and triangle. The formulae for perimeter of a square is 4 × side, for a rectangle is 2 × (length + breadth) and for a triangle is sum of its three sides respectively. **(06 Marks)**

c. Explain the concept of a static data member and a static member function with a suitable example. **(06 Marks)**

Module-2

3 a. Create a student class with three data members: student name, USN, percentage. Create an array of student objects, enter the details for the students and display the details. **(05 Marks)**

b. Explain the concept of pointer to an object with an example. **(05 Marks)**

c. What is a copy constructor? Give an example of initializing the elements of an integer array. **(06 Marks)**

OR

4 a. Write a program to find the difference of two numbers using default arguments. **(04 Marks)**

b. Explain the usage of reference parameters with an example of swapping two numbers. **(06 Marks)**

c. Explain the usage of dynamic allocation operators new and delete with an example. **(06 Marks)**

Module-3

5 a. Create a class called MATRIX using two dimensional array of integers. Implement the following operators by overloading: The operator = which checks the compatibility of two matrices to be subtracted. Overload the operator '–' for matrix subtraction as $m_3 = m_1 - m_2$ when $(m_1 = m_2)$. **(08 Marks)**

b. Create a class called complex which has two data members real part, imaginary part. Implement a friend function for overloading '+' operator which can compute the sum of two complex numbers and the function returns the complex object. **(08 Marks)**

OR

6 a. Create a base class base with two protected data members i and j and two public methods setij( ) and showij( ) to set the values of i and j and display the values of i and j respectively. Create a derived class which inherits base class as protected. Show how derived class object sets the values of i and j and display their values. **(08 Marks)**

b. What is the need for a virtual base class? Show how a virtual base class eliminates ambiguity. **(08 Marks)**

Module-4

7 a. How is run time polymorphism implemented in C++? Create a base class with a function fun1( ) defined in it. Show how fun1( ) can be overriden during runtime. **(08 Marks)**

b. What is a generic function? Write a generic function to implement bubble sort on an array of integers and an array of doubles in ascending order. **(08 Marks)**

OR

8 a. What is an exception? Handle the exception when a division by zero occurs. **(08 Marks)**

b. Give an example for handling derived class exceptions. **(08 Marks)**

Module-5

9 a. Mention the four built in streams in C++. Give an example of setting two format flags in ios. **(05 Marks)**

b. Give an example for creating our own manipulator function. **(05 Marks)**

c. Create an inventory file and write 3 items and their respective costs in it. **(06 Marks)**

OR

10 a. Create a student file and enter 3 students data like their USN and total marks. **(05 Marks)**

b. Explain the three foundational items of STL. Give one example of each. **(06 Marks)**

c. Create a vector of char of size 10 and assign the values to the elements of vector. **(05 Marks)**

* * * * *

VTU Question Paper- Even Semester 2017
Object Oriented Programming Using C++

1a.	Explain the five basic data types of C++ along with their size in bits.	04 Marks
Ans:	There are five atomic data types in the C++ subset: character, integer, floating-point, double floating-point, and valueless (**char**, **int**, **float**, **double**, and **void**, respectively). As you will see, all other data types in C are based upon one of these types. The size and range of these data types may vary between processor types and compilers. However, in all cases a character is 1 byte. The size of an integer is usually the same as the word length of the execution environment of the program. For most 16-bit environments, such as DOS or Windows 3.1, an integer is 16 bits. For most 32-bit environments, such as Windows 2000, an integer is 32 bits. The range of **float** and **double** will depend upon the method used to represent the floating-point numbers. Whatever the method, the range is quite large. Standard C specifies that the minimum range for a floating-point value is 1E−37 to 1E+37. Data type Size char 8 int 16 or 32 float 32 double 64	
1b.	What are const and volatile qualifiers? Give an example.	04 Marks

Ans:	There are two qualifiers that control how variables may be accessed or modified:
	const and **volatile**.
	const
	Variables of type **const** may not be changed by your program. (A **const** variable can be given an initial value, however.) The compiler is free to place variables of this type into read-only memory (ROM). For example,
	const int a=10;
	creates an integer variable called **a** with an initial value of 10 that your program may not modify. However, you can use the variable **a** in other types of expressions.
	#include <stdio.h>
	void sp_to_dash(const char *str);
	int main(void)
	{
	sp_to_dash("this is a test");
	return 0;
	}
	void sp_to_dash(const char *str)
	{
	while(*str) {
	if(*str== ' ') printf("%c", '-');
	else printf("%c", *str);
	str++;	

	} } volatile The modifier **volatile** tells the compiler that a variable's value may be changed in ways not explicitly specified by the program. For example, a global variable's address may be passed to the operating system's clock routine and used to hold the real time of the system. In this situation, the contents of the variable are altered without any explicit assignment statements in the program. Also, some compilers change the order of evaluation of an expression during the compilation process. The **volatile** modifier prevents these changes.	
1c.	Write a program to implement stack operations. Use constructor and destructor for the stack class.	08 Marks

Ans:	#include<iostream>
	#define max 3
	
	class stack
	{
	int top,stk[max];
	public:stack()
	{
	top=-1;
	}
	void push(int i)
	{
	if(top==max-1)
	{
	cout<<"Stack is full\n";
	}
	else
	{
	top=top+1;
	stk[top]=i;
	}
	}
	int pop()
	{
	if(top==-1)
	{
	cout<<"Stack is empty\n";
	}
	else
	{
	cout<<"Deleted item is:"<<stk[top]<<endl;
	top=top-1;
	}
	}	

```
      void display()
       {
         for(int i=0;i<top ; i++)
         {
            cout<<stk[i];
         }
}
    ~stack()
{   cout<<"Stack deleted";}
};


int main()
{
 stack s1;
 int ch,item;
 do
 {
  clrscr();
  cout<<"1;PUSH\n2;POP\n3:DISPLAY\n4;EXIT\n";
  cout<<"enter ur choice\n";
  cin>>ch;
  switch(ch)
  {
   case 1 : cout<<"Enter the item\n";
              cin>>item;
             s1push(item);
              break;
  case 2 : s1.pop();
             break;
  case 3 : s1.display();
             break;

 case 4 : exit(0);
             break;

 default : cout<<"Invalid choice\n";
              break;
 }
 }while(1);
}
```

2a.	Explain the four principles of object oriented programming.	04 Marks
Ans.	Object oriented programming can be defined as "an approach that provides a way of modularizing programs by creating partitioned memory area for both data and functions that can be used as	

templates for creating copies of such modules on demand".

Concepts of Oops:

Class:. Class is user defined data type and behave like the built-in data type of a programming language.Class is a blue print/model for creating objects.

Object: Object is the basic run time entities in an object oriented system. Object is the basic unit that is associated with the data and methods of a class.Object is an instance of a particular class.

Data Abstraction:

Abstraction refers to the act of representing essential features without including the background details. In programming languages, data abstraction will be specified by abstract data types and can be achieved through classes.

Encapsulation: The wrapping up of data and functions into a single unit is known as encapsulation. It keeps them safe from external interface and misuse as the functions that are wrapped in class can access it. The insulation of the data from direct access by the program is called data hiding.

Inheritance: It provides the concept of reusability. It is a mechanism of creating new classes from the existing classes. It supports the concept of hierarchical classification. A class which provides the properties is called Parent/Super/Base class. A class which acquires the properties is called Child/Sub/Derived class. A sub class defines only those features that are unique to it.

Polymorphism: Polymorphism is derived from two greek words Poly and Morphs where poly means many and morphis means forms. Polymorphism means one thing existing in many forms. Polymorphism plays an important role in allowing objects having different internal structures to share the same external interfaces. Function overloading and operator overloading can be used to achieve polymorphism.

2b.	Write a program which overloads a function for calculating the perimeter of geometrical figures like square, rectangle and triangle. The formulae for perimeter of a square is 4* side, for a rectangle is 2 *(length +breadth) and for a triangle is sum of its three sides respectively.	04 Marks

Ans:	#include<iostream>
	using namespace std;
	
	double perimeter(double n)
	{
	return 4*n;
	}
	double perimeter(double l, double b1)
	{
	return 2*(l+b1);
	}
	double perimeter(double a, double b, double c)
	{
	return (a+b+c);	

```
}
int main()
{
        double n,l,b1,a,b,c;
        cout<<" Enter the value of side in double to calculate the perimeter of square:\n";
        cin>>n;
        cout<<"Perimeter of Square is :"<<perimeter(n)<<endl;
        cout<<" Enter the value of length and breadth in Double to calculate the perimeter of
rectangle:\n";
        cin>>l>>b1;
        cout<<" Perimeter of Rectangle is :"<<perimeter(l,b1)<<endl;
        cout<<" Enter the value of three sides in Double to calculate the perimeter of Triangle:\n";
        cin>>a>>b>>c;
        cout<<" Perimeter of Rectangle is :"<<perimeter(a,b,c)<<endl;
        return 0;
}
```

2c.	Explain the concept of a static data member and a static member function with a suitable example.	06 Marks

Ans:	Static Data Members
	When you precede a member variable's declaration with **static**, then only one copy of that variable will exist and that all objects of the class will share that variable. No matter how many objects of a class are created, only one copy of a **static** data member exists. Thus, all objects of that class use that same variable. All **static** variables are initialized to zero before the first object is created.
	When you declare a **static** data member within a class, you are *not* defining it. We must provide a global definition for it elsewhere, outside the class. This is done by redeclaring the **static** variable using the scope resolution operator to identify the class to which it belongs. This causes storage for the variable to be allocated.
	Example
	`#include <iostream>`
	`using namespace std;`
	`class shared {`
	`static int a;`
	`int b;`
	`public:`
	`void set(int i, int j) {a=i; b=j;}`
	`void show();`
	`} ;`
	`int shared::a; // define a`
	`void shared::show()`
	`{`
	`cout << "This is static a: " << a;`
	`cout << "\nThis is non-static b: " << b;`
	`cout << "\n";`	

```
}
int main()
{
shared x, y;
x.set(1, 1); // set a to 1
x.show();
y.set(2, 2); // change a to 2
y.show();
x.show(); /* Here, a has been changed for both x and y
because a is shared by both objects. */
return 0;
}
```

```
This is static a: 1
This is non-static b: 1
This is static a: 2
This is non-static b: 2
This is static a: 2
This is non-static b: 1
```

Static Member Functions

Member functions may also be declared as **static**. There are several restrictions placed on **static** member functions. They may only directly refer to other **static** members of the class. (Of course, global functions and data may be accessed by **static** member functions.)

A **static** member function does not have a **this** pointer. There cannot be a **static** and a non-**static** version of the same function. A **static** member function may not be virtual. Finally, they cannot be declared as **const** or **volatile**.

```
#include <iostream>
using namespace std;
class cl {
static int resource;
public:
static int get_resource();
void free_resource() { resource = 0; }
};
int cl::resource; // define resource
int cl::get_resource()
{
if(resource) return 0; // resource already in use
else {
resource = 1;
return 1; // resource allocated to this object
}
}
int main()
{
cl ob1, ob2;
/* get_resource() is static so may be called independent
```

	of any object. */
	if(cl::get_resource()) cout << "ob1 has resource\n";
	if(!cl::get_resource()) cout << "ob2 denied resource\n";
	ob1.free_resource();
	if(ob2.get_resource()) // can still call using object syntax
	cout << "ob2 can now use resource\n";
	return 0;
	}	

3a.	Create a student class with three data members: student name, USN, percentage. Create an array of student objects, enter the details for the students and display the details.	05 Marks

Ans:	#include <iostream>
	using namespace std;
	#define Max 20
	
	class Student
	{
	
	char usn[Max];
	char name[Max];
	double: p;
	public :
	
	void readstudent();
	void display();
	};
	
	void Student :: readstudent()
	{
	cout<<"Enter the usn\n";
	cin>>usn;
	cout<<"Enter the name\n";
	cin>>name;
	cout<<"enter the percentage \n";
	cin>>p;
	
	}
	
	void Student :: display()
	{
	cout<<usn<<"\t"<<name<<"\t\t\t"<<p<<endl;
	}
	
	int main()	

```
{
        Student st[Max];
        int i,n;
        cout<<"Enter the number of students\n";
        cin>>n;
        for(i=0;i<n;i++)
        {
                st[i].readstudent();
        }

        cout<<"USN \t Student Name \t Percentage \n";
        for(i=0;i<n;i++)
        {
                st[i].display();
        }
        return 0;
}
```

3b.	Explain the concept of pointer to an object with an example.	05 Marks

Ans:	Just as we can have pointers to other types of variables, we can have pointers to objects. To access a member of a class through the pointer to an object, the -> operator is used. When a member function is called, it automatically passes an implicit argument. It is a pointer to the invoking objects (the object on which the function is called). It is known as 'this' pointer.
	Example:
	#include <iostream>
	using namespace std;
	class cl {
	int i;
	public:
	cl(int j) { i=j; }
	int get_i() { return i; }
	};
	int main()
	{
	cl ob(88), *p;
	p = &ob; // get address of ob
	cout << p->get_i(); // use -> to call get_i()
	return 0;
	}
	As you know, when a pointer is incremented, it points to the next element of its type. For example, an integer pointer will point to the next integer.	

3c.	What is a copy constructor? Give an example of initializing the elements of an integer array.	06 Marks

Ans:

The parameters of a constructor can be of any of the data types except an object of its own class as a value parameter. Hence declaration of the following class specification leads to an error:

```
class x
{
        private:
                ……
        public:
                x( x obj);
                ……..
};
```

A class's own object can be passed as a reference parameter.

Ex:

```
class X
{       ……..
        public:
        X()
        X( X &obj);
        X(int a);
};
```

is valid

Such a constructor having a reference to an instance of its own class as an argument is known as copy constructor.

Ex.

```
bag b3=b2;   // copy constructor invoked
bag b3(b2);   // copy constructor invoked
b3=b2;        // copy constructor is not invoked.
```

A copy constructor copies the data members from one object to another.

Program to initialize the elements of an integer array:

```
#include <iostream>
#include <new>
#include <cstdlib>
using namespace std;
class array {
int *p;
int size;
public:
array(int sz) {
try {
p = new int[sz];
} catch (bad_alloc xa) {
cout << "Allocation Failure\n";
exit(EXIT_FAILURE);
}
```

```
size = sz;
}
~array() { delete [] p; }
// copy constructor
array(const array &a);
void put(int i, int j) {
if(i>=0 && i<size) p[i] = j;
}
int get(int i) {
return p[i];
}
};
// Copy Constructor
array::array(const array &a) {
int i;
try {
p = new int[a.size];
} catch (bad_alloc xa) {
cout << "Allocation Failure\n";
exit(EXIT_FAILURE);
}
for(i=0; i<a.size; i++) p[i] = a.p[i];
                }
int main()
{
array num(10);
int i;
for(i=0; i<10; i++) num.put(i, i);
for(i=9; i>=0; i--) cout << num.get(i);
cout << "\n";
// create another array and initialize with num
array x(num); // invokes copy constructor
for(i=0; i<10; i++) cout << x.get(i);
return 0;
                }
```

4a.	Write a program to find the difference of two numbers using default arguments.	04 Marks

Ans:	`#include <iostream>`
	`using namespace std;`
	
	`void diff(int a,int b= 6);`
	`int main( )`
	`{`
	`int a,b;`	

```
    cout<<"enter any two numbers\n";
    cin>>a>>b;
    diff(a) ; // sum of default values
    diff(a,b);
    diff(b);
    return 0;
}
void diff (int a1, int a2)
{
    int temp;
    temp = a1 - a2;
    cout<<"a="<<a1<<endl;
    cout<<"b="<<a2<<endl;
    cout<<"Difference="<<temp<<endl;
}
```

4b.	Explain the usage of reference parameters with an example of swapping two numbers.	06 Marks
Ans:	C++ allows a function to assign a parameter a default value when no argument corresponding to that parameter is specified in a call to that function. Probably the most important use for a reference is to allow you to create functions that automatically use call-by-reference parameter passing. Arguments can be passed to functions in one of two ways: using call-by-value or call-by-reference. When using call-by-value, a copy of the argument is passed to the function. Call-by-reference passes the address of the argument to the function. By default, C++ uses call-by-value, but it provides two ways to achieve call-by-reference parameter passing. First, you can explicitly pass a pointer to the argument. Second, you can use a reference parameter. For most circumstances the best way is to use a reference parameter. Here is an example. This program uses reference parameters to swap the values of the variables it is called with. The **swap( )** function is the classic example of call-by-reference parameter passing.	

```
#include <iostream>
using namespace std;
void swap(int &i, int &j);
int main()
{
int a, b, c, d;
a = 1;
b = 2;
c = 3;
d = 4;
cout << "a and b: " << a << " " << b << "\n";
swap(a, b); // no & operator needed
cout << "a and b: " << a << " " << b << "\n";
cout << "c and d: " << c << " " << d << "\n";
swap(c, d);
cout << "c and d: " << c << " " << d << "\n";
```

	return 0;
	}
	void swap(int &i, int &j)
	{
	int t;
	t = i; // no * operator needed
	i = j;
	j = t;
	}
	This program displays the following:
	a and b: 1 2
	a and b: 2 1
	c and d: 3 4
	c and d: 4 3	

4c.	Explain the usage of dynamic allocation operators new and delete with an example.	06 Marks

Ans:	Dynamic memory allocation operators: new and delete.
	new: To allocate the memory.
	Syntax:
	ptr_var = new vartype;
	e.g: ptr = new int;
	ptr_var = new vartype(initial_value);
	The type of initial value should be same as the vartype;
	e.g: ptr = new int(100);
	delete: To free the memory.
	delete ptr_var;
	If there is insufficient memory then the exception bad_alloc will be raised. This exception is defined in the header <new>. It is available in standard C++.
	Advantages of new and delete:
	new and delete operators are like malloc() and free() in C Language. But they have more advantages.
	1. new automatically allocates enough memory to hold the object.
	(No need to use sizeof operator).
	2. new automatically returns the pointer to the specified type.
	It is not needed to typecast it explicitly.
	Allocating Arrays:
	ptrvar = new arrtype[size];
	Delete [ ] ptrvar;
	*** The initial values can't be given during the array allocation.
	Example:
	#include <iostream>
	#include <new>
	using namespace std;	

	int main()
	{
	int *p;
	try {
	p = new int; // allocate space for an int
	} catch (bad_alloc xa) {
	cout << "Allocation Failure\n";
	return 1;
	}
	*p = 100;
	cout << "At " << p << " ";
	cout << "is the value " << *p << "\n";
	delete p;
	return 0;
	}	

5a.	Create a class called MATRIX using two-dimensional array of integers. Implement the following operations by overloading the operator == which checks the compatibility of two matrices to be subtracted. Overload the operator '-'for matrix subtraction as m3 = m1-m2 when (m1= =m2).	08 Marks

Ans:	#include<iostream>
	#define Max 20
	using namespace std;
	class Matrix
	{
	public:
	int a[Max][Max];
	int r,c;
	void getorder();
	void getdata();
	Matrix operator -(Matrix);
	friend ostream& operator <<(ostream &, Matrix);
	int operator==(Matrix);
	
	};
	void Matrix::getorder()
	{
	cout<<"enter the number of rows\n";
	cin>>r;
	cout<<"enter the number of columns\n";
	cin>>c;
	}
	void Matrix::getdata()
	{
	int i,j;	

```
        for(i=0;i<r;i++)
        {
                for(j=0;j<c;j++)
                {
                        cin>>a[i][j];
                }
        }
}

Matrix Matrix::operator -(Matrix m2)
{
        Matrix m4;
        int i,j;
        for(i=0;i<r;i++)
        {
                for(j=0;j<c;j++)
                {
                        m4.a[i][j] = a[i][j] - m2.a[i][j];
                }
        }
        m4.r = r;
        m4.c = c;
        return m4;
}
ostream & operator <<(ostream & out, Matrix m)
{
        int i,j;
        for(i=0;i<m.r;i++)
        {
                for(j=0;j<m.c;j++)
                {
                out<<m.a[i][j]<<"\t";
                }
                out<<endl;
        }
        return out;
}
int Matrix::operator==(Matrix m2)
{
        if((r==m2.r) && (c==m2.c))
                return 1;
        else
                return 0;

}
int main()
```

```
{
        Matrix m1,m2,m4;
        cout<<"enter the order of the first matrix\n";
        m1.getorder();
        cout<<"enter the order of the second matrix\n";

        m2.getorder();

        if(m1 == m2)

        {
                cout<<"enter the elements of the first matrix\n";
                m1.getdata();
                cout<<"enter the elements of the second matrix\n";
                m2.getdata();
                m4 = m1 - m2;
                cout<<"Difference of matrices is \n";
                cout<<m4<<endl;

        } else {

                cout<<"Order of the matrices is not same";
        }
        return 0;
}
```

5b.	Create a class called complex which has two data members real part, imaginary part. Implement a friend function for overloading '+' operator which can compute the sum of two complex numbers and the function returns the complex object.	08 Marks
Ans:		

```
#include<iostream>

using namespace std;

class complex

{
        int real;
        int imag;
public:
        void read()
        {
                cout<<"enter real and imaginary";
                cin>>real>>imag;
        }
```

```
        void display()
        {
                        cout<<real<<"+"<<imag<<"i"<<endl;
        }
 friend complex operator +(complex, complex);
        };

complex operator +(complex a1,complex a2)

        {
                complex temp1;
                temp1.real=a1.real+a2.real;
                temp1.imag=a1.imag+a2.imag;
                return temp1;
        }

int main()

{
        int a;
        complex s1,s2,s3;
        s1.read();
        s2.read();
        cout<<"First Complex number";
        s1.display();
        cout<<"Second Complex number";
        s2.display();
        s3=s1+s2;
        cout<<"addition of 2 complex number\n"<<endl;
        s3.display();
        return 0;
}
```

6a.	Create a base class with two protected data members i and j and two public methods setij() and showij() to set the values of I and j and display the values of i and j respectively. Create a derived class which inherits base class as protected. Show how derived class object sets the values of i and j and display their values.	08 Marks
Ans:	It is possible to inherit a base class as **protected.** When this is done, all public and protected members of the base class become protected members of the derived class. For example, #include <iostream> using namespace std;	

	class base {
	protected:
	int i, j; // private to base, but accessible by derived
	public:
	void setij(int a, int b) { i=a; j=b; }
	void showij() { cout << i << " " << j << "\n"; }
	};
	// Inherit base as protected.
	class derived : protected base{
	int k;
	public:
	// derived may access base's i and j and setij().
	void setk() { setij(10, 12); k = i*j; }
	// may access showij() here
	void showall() { cout << k << " "; showij(); }
	};
	int main()
	{
	derived ob;
	// ob.setij(2, 3); // illegal, setij() is
	// protected member of derived
	ob.setk(); // OK, public member of derived
	ob.showall(); // OK, public member of derived
	// ob.showij(); // illegal, showij() is protected
	// member of derived
	return 0;
	}
	As you can see by reading the comments, even though **setij( )** and **showij( )** are public members of **base**, they become protected members of **derived** when it is inherited using the **protected** access specifier. This means that they will not be accessible inside **main( )**.	

6b.	What is the need for a virtual base class? Show how a virtual base class eliminates ambiguity.	08 Marks

Ans:	When two or more objects are derived from a common base class, you can prevent multiple copies of the base class from being present in an object derived from those objects by declaring the base class as **virtual** when it is inherited. You accomplish this by preceding the base class' name with the keyword **virtual** when it is inherited. For example, here is another version of the example program in which **derived3** contains only one copy of **base**:
	// This program uses virtual base classes.
	#include <iostream>
	using namespace std;
	class base {
	public:
	int i;
	};
	// derived1 inherits base as virtual.	

```
class derived1 : virtual public base {
public:
int j;
};
// derived2 inherits base as virtual.
class derived2 : virtual public base {
public:
int k;
};
/* derived3 inherits both derived1 and derived2.
This time, there is only one copy of base class. */
class derived3 : public derived1, public derived2 {
public:
int sum;
};
int main()
{
derived3 ob;
   ob.i = 10; // now unambiguous
ob.j = 20;
ob.k = 30;
// unambiguous
ob.sum = ob.i + ob.j + ob.k;
// unambiguous
cout << ob.i << " ";
cout << ob.j << " " << ob.k << " ";
cout << ob.sum;
return 0;
}
```

As you can see, the keyword **virtual** precedes the rest of the inherited **class**' specification. Now that both **derived1** and **derived2** have inherited **base** as **virtual**, any multiple inheritance involving them will cause only one copy of **base** to be present. Therefore, in **derived3**, there is only one copy of **base** and **ob.i = 10** is perfectly valid and unambiguous. One further point to keep in mind: Even though both **derived1** and **derived2** specify **base** as **virtual**, **base** is still present in objects of either type. For example, the following sequence is perfectly valid:

```
// define a class of type derived1
derived1 myclass;
myclass.i = 88;
```

The only difference between a normal base class and a **virtual** one is what occurs when an object inherits the base more than once. If **virtual** base classes are used, then only one base class is present in the object. Otherwise, multiple copies will be found.

7a.	How is run time polymorphism implemented in C++? Create a base class with a function fun1() defined in it. Show how fun1() can be overridden during runtime.	08 Marks
Ans:	One of the central aspects of object-oriented programming is the principle of "one interface, multiple methods." This means that a general class of actions can be defined, the interface to which	

	is constant, with each derivation defining its own specific operations. In concrete C++ terms, a base class can be used to define the nature of the interface to a general class. Each derived class then implements the specific operations as they relate to the type of data used by the derived type. One of the most powerful and flexible ways to implement the "one interface, multiple methods" approach is to use virtual functions, abstract classes, and run-time polymorphism. Using these features, you create a class hierarchy that moves from general to specific (base to derived). Following this philosophy, you define all common features and interfaces in a base class. In cases where certain actions can be implemented only by the derived class, use a virtual function. In essence, in the base class you create and define everything you can that relates to the general case. The derived class fills in the specific details.
	/* Here, a base class reference is used to access a virtual function. */
	#include <iostream>
	using namespace std;
	class base {
	public:
	virtual void fun1() {
	cout << "This is base's fun1().\n";
	}
	};
	class derived1 : public base {
	public:
	void fun1() {
	cout << "This is derived1's fun1().\n";
	}
	};
	class derived2 : public base {
	public:
	void fun1() {
	cout << "This is derived2's fun1().\n";
	}
	};
	int main()
	{
	base *b,b1;
	derived1 d1;
	derived2 d2;
	b=&b1; // calling a base class fun1()
	b=&d1; // calling a derived1 class fun1()
	b=&d2; // calling a derived2 class fun1()
	return 0;
	}	

7b.	What is a generic function? Write a generic function to implement bubble sort on an array of integers and an array of doubles in ascending order.	08 Marks
Ans:	A generic function defines a general set of operations that will be applied to various types of data. The type of data that the function will operate upon is passed to it as a parameter.	

A generic function is created using the keyword **template**. The normal meaning of the word "template" accurately reflects its use in C++. It is used to create a template (or framework) that describes what a function will do, leaving it to the compiler to fill in the details as needed. The general form of a template function definition is shown here:

template <class *Ttype*> *ret-type func-name*(*parameter list*)
{
// *body of function*
}
Here, *Ttype* is a placeholder name for a data type used by the function.
Program : Bubble sort using generic function:

```
#include<iostream>

using namespace std;
#define Max 100

template <class T>
void sort(T a[],int n)
{
int i,j;
for(i=0;i<n-1;i++)
        {
                for(j=0;j<n-i-1;j++)
                        {
                                if(a[j]>a[j+1])
                                {
                                        T temp=a[j];
                                        a[j]=a[j+1];
                                        a[j+1]=temp;
                                }
                        }
        }
}

int main()

{
        int a[Max],i,n;
        double d[Max];
        cout<<"enter array size\n\n";
        cin>>n;
        cout<<"enter array integer elements\n\n";
        for(i=0;i<n;i++)
                cin>>a[i];
                cout<<"enter array double elements\n\n";
        for(i=0;i<n;i++)
```

	cin>>d[i];
	cout<<"integer part\n\n";
	sort(a,n);
	for(i=0;i<n;i++)
	cout<< a[i]<<"\n";
	cout<<"double part\n\n";
	sort(d,n);
	for(i=0;i<n;i++)
	cout<<d[i]<<"\n";
	return 0;
	}	

8a.	What is an exception? Handle the exception when a division by zero occurs.	08 Marks

Ans:	Exception is run time error which stop the program execution. *Exception handling* allows you to manage run-time errors in an orderly fashion. C++ exception handling is built upon three keywords: **try**, **catch**, and **throw**. In the most general terms, program statements that you want to monitor for exceptions are contained in a **try** block. If an exception (i.e., an error) occurs within the **try** block, it is thrown (using **throw**). The exception is caught, using **catch**, and processed. The following discussion elaborates upon this general description. Code that you want to monitor for exceptions must have been executed from within a **try** block. (Functions called from within a **try** block may also throw an exception.) Exceptions that can be thrown by the monitored code are caught by a **catch** statement, which immediately follows the **try** statement in which the exception was thrown. The general form of **try** and **catch** are shown here.
	try {
	// *try block*
	}
	catch (*type1 arg*) {
	// *catch block*
	}
	catch (*type2 arg*) {
	// *catch block*
	}
	catch (*type3 arg*) {
	// *catch block*
	}...
	catch (*typeN arg*) {
	// *catch block*
	}
	This program uses exception handling to manage a divide-by-zero error.
	#include <iostream>
	using namespace std;
	void divide(double a, double b);
	int main()
	{	

	double i, j;
	do {
	cout << "Enter numerator (0 to stop): ";
	cin >> i;
	cout << "Enter denominator: ";
	cin >> j;
	divide(i, j);
	} while(i != 0);
	return 0;
	}
	void divide(double a, double b)
	{
	try {
	if(!b) throw b; // check for divide-by-zero
	cout << "Result: " << a/b << endl;
	}
	catch (double b) {
	cout << "Can't divide by zero.\n";
	}
	}	

8b.	Give an example for handling derived class exceptions.	08 Marks

Ans:	We need to be careful how you order your **catch** statements when trying to catch exception types that involve base and derived classes because a **catch** clause for a base class will also match any class derived from that base. Thus, if you want to catch exceptions of both a base class type and a derived class type, put the derived class first in the **catch** sequence. If you don't do this, the base class **catch** will also catch all derived classes. For example, consider the following program.
	// Catching derived classes.
	#include <iostream>
	using namespace std;
	class B {
	};
	class D: public B {
	};
	int main()
	{
	D derived;
	try {
	throw derived;
	}
	catch(B b) {
	cout << "Caught a base class.\n";
	}
	catch(D d) {
	cout << "This won't execute.\n";	

	} return 0; } Here, because **derived** is an object that has **B** as a base class, it will be caught by the first **catch** clause and the second clause will never execute. Some compilers will flag this condition with a warning message. Others may issue an error. Either way, to fix this condition, reverse the order of the **catch** clauses.	
9a.	Mention the four built in streams in C++. Give an example of setting two format flags in ios.	05 Marks
	When a C++ program begins execution, four built-in streams are automatically opened. They are: Stream Meaning Default Device cin Standard input Keyboard cout Standard output Screen cerr Standard error output Screen clog Buffered version of cerr Screen By default, the standard streams are used to communicate with the console. However, in environments that support I/O redirection (such as DOS, Unix, OS/2, and Windows), the standard streams can be redirected to other devices or files. **The standard input stream (cin):** The predefined object **cin** is an instance of **istream** class. The cin object is said to be attached to the standard input device, which usually is the keyboard. The **cin** is used in conjunction with the stream extraction operator, which is written as >> #include <iostream> using namespace std; int main( ) { char name[50]; cout << "Please enter your name: "; cin >> name;	

```
  cout << "Your name is: " << name << endl;

}
```

The standard output stream (cout):

The predefined object **cout** is an instance of **ostream** class. The cout object is said to be "connected to" the standard output device, which usually is the display screen. The **cout** is used in conjunction with the stream insertion operator, which is written as <<

Ex:

```
#include <iostream>

 using namespace std;

 int main( )

{

  char str[] = "Hello C++";

   cout << "Value of str is : " << str << endl;

}
```

The standard error stream (cerr):

The predefined object **cerr** is an instance of **ostream** class. The cerr object is said to be attached to the standard error device, which is also a display screen but the object **cerr** is un-buffered and each stream insertion to cerr causes its output to appear immediately.

```
#include <iostream>

using namespace std;

int main( )
{
  char str[] = "Unable to read....";

  cerr << "Error message : " << str << endl;
}
```

The standard log stream (clog):

The predefined object **clog** is an instance of **ostream** class. The clog object is said to be attached to the standard error device, which is also a display screen but the object **clog** is buffered. This means that each insertion to clog could cause its output to be held in a buffer until the buffer is filled or

	until the buffer is flushed
	
	#include <iostream>
	
	using namespace std;
	
	int main( )
	{
	  char str[] = "Unable to read....";
	
	  clog << "Error message : " << str << endl;
	}
	
	Example of setting two format flag:
	
	#include <iostream>
	using namespace std;
	int main()
	{
	cout.setf(ios::showpoint);
	cout.setf(ios::showpos);
	cout << 100.0; // displays +100.000
	return 0;
	}	

9b.	Give an example for creating our own manipulator function.	05 Marks

Ans.	Manipulator functions are special stream functions that change certain characteristics of the input and output. They change the format flags and values for a stream. The main advantage of using manipulator functions is that they facilitate that formatting of input and output streams.
	o carry out the operations of these manipulator functions in a user program, the header file input and output manipulator <iomanip.h> must be included.
	Creating our own manipulator:
	All parameterless manipulator output functions have this skeleton:
	ostream &*manip-name*(ostream &*stream*)
	{
	// *your code here*
	return *stream*;
	}
	
	Example:	

	#include <iostream>
	#include <iomanip>
	using namespace std;
	// A simple output manipulator.
	ostream &sethex(ostream &stream)
	{
	stream.setf(ios::showbase);
	stream.setf(ios::hex, ios::basefield);
	return stream;
	}
	int main()
	{
	cout << 256 << " " << sethex << 256;
	return 0;
	}
	O/P : **256 0x100**	

9c.	Create an inventory file and write 3 items and their respective costs in it.	06 Marks

	This program creates a short inventory file that contains each item's name and its cost:
	#include <iostream>
	#include <fstream>
	using namespace std;
	int main()
	{
	ofstream out("INVNTRY"); // output, normal file
	if(!out) {
	cout << "Cannot open INVENTORY file.\n";
	return 1;
	}
	out << "Radios " << 39.95 << endl;
	out << "Toasters " << 19.95 << endl;
	out << "Mixers " << 24.80 << endl;
	out.close();
	return 0;
	}
	The following program reads the inventory file created by the previous program and displays its contents on the screen:
	#include <iostream>
	#include <fstream>
	using namespace std;
	int main()
	{
	ifstream in("INVNTRY"); // input	

	if(!in) {
	cout << "Cannot open INVENTORY file.\n";
	return 1;
	}
	char item[20];
	float cost;
	in >> item >> cost;
	cout << item << " " << cost << "\n";
	in >> item >> cost;
	cout << item << " " << cost << "\n";
	in >> item >> cost;
	cout << item << " " << cost << "\n";
	in.close();
	return 0;
	}	

10a.	Create a student file and enter 3 students data like their USN and total marks.	05 Marks

Ans:	#include <iostream>
	#include <fstream>
	using namespace std;
	int main()
	{
	ofstream out("Student"); // output, normal file
	if(!out) {
	cout << "Cannot open Student file.\n";
	return 1;
	}
	out << "1CR13MCA21 " <<"Naveen"<< 69.95 << endl;
	out << "1CR13MCA22 " <<" Navneeth"<< 49.95 << endl;
	out << "1CR13MCA23 " << "Pankaj"<<54.80 << endl;
	out.close();
	return 0;
	}
	The following program reads the inventory file created by the previous program and displays its contents on the screen:
	#include <iostream>
	#include <fstream>
	using namespace std;
	int main()
	{
	ifstream in("Student"); // input
	if(!in) {
	cout << "Cannot open Student file.\n";
	return 1;
	}	

	char USN[20], name[20];
	float Marks;
	in >> USN >> name>>Marks;
	cout << USN << " " << name << Marks "\n";
	in >> USN >> name>>Marks;
	cout << USN << " " << name << Marks "\n";
	in >> USN >> name>>Marks;
	cout << USN << " " << name << Marks "\n";
	in.close();
	return 0;
	}	

10b.	Explain the three foundational items of STL. Give one example of each.	06 Marks
Ans:	At the core of the standard template library are three foundational items: *containers*, *algorithms*, and *iterators*. These items work in conjunction with one another to provide off-the-shelf solutions to a variety of programming problems. Containers *Containers* are objects that hold other objects, and there are several different types. For example, the **vector** class defines a dynamic array, **deque** creates a double-ended queue, and **list** provides a linear list. These containers are called *sequence containers* because in STL terminology, a sequence is a linear list. In addition to the basic containers, the STL also defines *associative containers*, which allow efficient retrieval of values based on keys. For example, a **map** provides access to values with unique keys. Thus, a **map** stores a key/value pair and allows a value to be retrieved given its key. Each container class defines a set of functions that may be applied to the container. For example, a list container includes functions that insert, delete, and merge elements. A stack includes functions that push and pop values. Algorithms *Algorithms* act on containers. They provide the means by which you will manipulate the contents of containers. Their capabilities include initialization, sorting, searching, and transforming the contents of containers. Many algorithms operate on a *range* of elements within a container. Iterators *Iterators* are objects that act, more or less, like pointers. They give you the ability to cycle through the contents of a container in much the same way that you would use a pointer to cycle through an array. There are five types of iterators: Iterator Access Allowed Random Access Store and retrieve values. Elements may be accessed randomly. Bidirectional Store and retrieve values. Forward and backward moving. Forward Store and retrieve values. Forward moving only. Input Retrieve, but not store values. Forward moving only. Output Store, but not retrieve values. Forward moving only.	
10c.	Create a vector of char of size 10 and assign the values to the elements of vector.	05 Marks
Ans:	// Demonstrate a vector. #include <iostream>	

```
#include <vector>
#include <cctype>
using namespace std;
int main()
{
vector<char> v(10); // create a vector of length 10
unsigned int i;
// display original size of v
cout << "Size = " << v.size() << endl;
// assign the elements of the vector some values
for(i=0; i<10; i++) v[i] = i + 'a';
// display contents of vector
cout << "Current Contents:\n";
for(i=0; i<v.size(); i++) cout << v[i] << " ";
cout << "\n\n";
cout << "Expanding vector\n";
/* put more values onto the end of the vector,
it will grow as needed */
for(i=0; i<10; i++) v.push_back(i + 10 + 'a');
// display current size of v
cout << "Size now = " << v.size() << endl;
// display contents of vector
cout << "Current contents:\n";
for(i=0; i<v.size(); i++) cout << v[i] << " ";
cout << "\n\n";
// change contents of vector
for(i=0; i<v.size(); i++) v[i] = toupper(v[i]);
cout << "Modified Contents:\n";
for(i=0; i<v.size(); i++) cout << v[i] << " ";
cout << endl;
return 0;
}
```

The output of this program is shown here:

```
Size = 10
Current Contents:
a b c d e f g h i j
Expanding vector
Size now = 20
Current contents:
a b c d e f g h i j k l m n o p q r s t
Modified Contents:
A B C D E F G H I J K L M N O P Q R S T
```