


CMR INSTITUTE OF TECHNOLOGY	USN <input type="text"/>								

Internal Assesment Test - III

Sub:	Object Oriented Programming Using C++	Code:	16MCA22
Date:	30.05.2017	Duration:	90 mins
		Max Marks:	50
		Sem:	II
		Branch:	MCA

Answer Any FIVE FULL Questions

		Marks	OBE	
			CO	RBT
1(a)	Explain STL container classes.	[10]	CO3	L2
2(a)	What is Object Oriented Programming? Bring out the underlying concepts of Oops?	[10]	CO1	L2
3(a)	Explain manipulators. Write a Program to create your own manipulators.	[5]	CO3	L2
(b)	What is pure virtual function? Discuss its significance.	[5]	CO3	L2
4(a)	What is stream? Discuss the four streams which are automatically opened when a C++ Program begins execution.	[6]	CO3	L2
(b)	Explain setw and setfill manipulators with example.	[4]	CO3	L2

5(a)	What is copy Constructor? Explain with one suitable example.	[5]	CO2	L2
(b)	What is friend function and friend class? Why it is used?	[5]	CO2	L2
6(a)	What are inserter and extractor? Explain how to create your own inserter and extractor with an example.	[10]	CO3	L2
7(a)	Explain inline function with an example.	[5]	CO2	L2
(b)	What is iterator ? Write a Program to access vector through iterator.	[5]	CO3	L3
8(a)	1. Define i) seekg() ii)seekp() iii)tellg() iv)precision() v)fill()	[10]	CO3	L2

Course Outcomes		PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8
CO1:	Differentiate between object oriented programming and procedure oriented programming & Disseminate the importance of Object oriented programming	1	1	-	-	-	-	3	3
CO2:	Apply C++ features such as Classes, objects, constructors, destructors, inheritance, operator overloading, and Polymorphism, Template and exception handling in program design and implementation.	2	2	3	-	-	-	2	3
CO3:	Use C++ to demonstrate practical experience in developing object-oriented solutions.	1	3	3	1	-	-	3	3
CO4:	Analyze a problem description and build object-oriented software using good coding practices and techniques.	1	2	3	2	-	-	3	3
CO5:	Implement an achievable practical application and analyze issues related to object-oriented techniques in the C++ programming language.	1	1	2	-	-	-	3	3

Cognitive level	KEYWORDS
L1	List, define, tell, describe, identify, show, label, collect, examine, tabulate, quote, name, who, when, where, etc.
L2	summarize, describe, interpret, contrast, predict, associate, distinguish, estimate, differentiate, discuss, extend
L3	Apply, demonstrate, calculate, complete, illustrate, show, solve, examine, modify, relate, change, classify, experiment, discover.
L4	Analyze, separate, order, explain, connect, classify, arrange, divide, compare, select, explain, infer.
L5	Assess, decide, rank, grade, test, measure, recommend, convince, select, judge, explain, discriminate, support, conclude, compare, summarize.

PO1 - Apply *knowledge*; PO2 - *Problem analysis*; PO3 - *Design/development of solutions*; PO4 - team work; PO5 - *Ethics*; PO6 - Communication; PO7- *Business Solution*; PO8 – Life-long learning;

Q1(a). Explain STL container classes.

Ans: 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

It also defines associative containers, which allow efficient retrieval of values based on keys. For example, a **map** provides access to values with unique keys.

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.

Container	Description	Required Header
bitset	A set of bits.	<bitset>
deque	A double-ended queue.	<deque>
list	A linear list.	<list>
map	Stores key/value pairs in which each key is associated with only one value.	<map>
multimap	Stores key/value pairs in which one key may be associated with two or more values.	<map>
multiset	A set in which each element is not necessarily unique.	<set>
priority_queue	A priority queue.	<queue>
queue	A queue.	<queue>
set	A set in which each element is unique.	<set>
stack	A stack.	<stack>
vector	A dynamic array.	<vector>

Table 24-1. The Containers Defined by the STL

Ex:

```
#include <vector>
#include <iostream>
int main()
{
using namespace std;
vector<int> vect;
for (int nCount=0; nCount < 6; nCount++)
vect.push_back(10 - nCount); // insert at end of array
for (int nIndex=0; nIndex < vect.size(); nIndex++)
cout << vect[nIndex] << " ";
cout << endl;
}
```

Q2(a) What is Object Oriented Programming? Bring out the underlying concepts of Oops?

Ans: To overcome the flaws in procedural approach such as global variables, emphasis on functions rather than data, top down approach in program design etc, object oriented approach is introduced.

Oops treat data as a critical element in the program development and does not allow it to flow freely around the system. The data of an object can be accessed only by the functions associated with that object.

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:

1. Class is user defined data type and behave like the built-in data type of a programming language.
2. Class is a blue print/model for creating objects.
3. Class specifies the properties and actions of an object.
4. Class does not physically exist.
5. Once a class has been defined, we create any number of objects belonging to that class. Thus, class is a collection of objects of similar type.

Object:

1. Object is the basic run time entities in an object oriented system.
2. Object is the basic unit that are associated with the data and methods of a class.
3. Object is an instance of a particular class.
4. Object physically exists.
5. Objects take up space in memory and have an associated address.
6. Objects communicate by sending messages to one another.

Data Abstraction and Encapsulation:

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.

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:

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

Polymorphism:

1. Polymorphism is derived from two greek words Poly and Morphis where poly means many and morphis means forms.
2. Polymorphism means one thing existing in many forms.

3. Polymorphism plays an important role in allowing objects having different internal structures to share the same external interfaces.

4. Polymorphism is extensively used in implementing inheritance.

5. Function overloading and operator overloading can be used to achieve polymorphism.

Dynamic Binding:

1. Binding refers to the linking of a procedure call to be executed in response to the call.

2. If the binding occurs at runtime then it is called as dynamic binding.

3. It is also called as late binding as binding of a call to the procedure is not known until runtime.

4. Dynamic Binding is associated with polymorphism and inheritance.

Message Binding:

1. Objects communicate with each other by sending and receiving information using functions.

2. The basic steps to perform message passing are

* Creating classes that define objects and their behaviour.

* Creating objects from class definitions.

* Establishing communication among objects.

3. Message passing involves specifying the name of the object, name of the function and the information to be sent as

```
objectname.functionname(message);
```

4. A message for an object is a request for execution of a procedure and therefore will invoke a function in receiving object that generates the desired result.

5. Communication with an object is feasible as long as it is alive.

Q3(a) Explain manipulators. Write a Program to create your own manipulators.

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.

The following are the list of standard manipulator used in a C++ program. To carry out the operations of these manipulator functions in a user program, the header file `<iomanip>` is used.

manipulator <iomanip.h> must be included.

Manipulator	Purpose	Input/Output
boolalpha	Turns on boolalpha flag.	Input/Output
dec	Turns on dec flag.	Input/Output
endl	Output a newline character and flush the stream.	Output
ends	Output a null.	Output
fixed	Turns on fixed flag.	Output
flush	Flush a stream.	Output
hex	Turns on hex flag.	Input/Output
internal	Turns on internal flag.	Output
left	Turns on left flag.	Output
noboolalpha	Turns off boolalpha flag.	Input/Output
noshowbase	Turns off showbase flag.	Output
noshowpoint	Turns off showpoint flag.	Output
noshowpos	Turns off showpos flag.	Output

Table 20-1. *The C++ Manipulators*

Manipulator	Purpose	Input/Output
noskipws	Turns off skipws flag.	Input
nounitbuf	Turns off unitbuf flag.	Output
nouppercase	Turns off uppercase flag.	Output
oct	Turns on oct flag.	Input/Output
resetiosflags (fmtflags <i>f</i>)	Turn off the flags specified in <i>f</i> .	Input/Output
right	Turns on right flag.	Output
scientific	Turns on scientific flag.	Output
setbase(int <i>base</i>)	Set the number base to <i>base</i> .	Input/Output
setfill(int <i>ch</i>)	Set the fill character to <i>ch</i> .	Output
setiosflags(fmtflags <i>f</i>)	Turn on the flags specified in <i>f</i> .	Input/output
setprecision (int <i>p</i>)	Set the number of digits of precision.	Output
setw(int <i>w</i>)	Set the field width to <i>w</i> .	Output
showbase	Turns on showbase flag.	Output
showpoint	Turns on showpoint flag.	Output
showpos	Turns on showpos flag.	Output
skipws	Turns on skipws flag.	Input
unitbuf	Turns on unitbuf flag.	Output
uppercase	Turns on uppercase flag.	Output
ws	Skip leading white space.	Input

Table 20-1. *The C++ Manipulators (continued)*

Ex :

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
cout << hex << 100 << endl;
cout << setfill('?') << setw(10) << 2343.0;
return 0;
}
```

This displays

```
64
??????2343
```

Creating our own inserter:

All parameterless manipulator output functions have this skeleton:

```
ostream &manip-name(ostream &stream)
{
// your code here
return stream;
}
```

```
#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

Q3(b) What is pure virtual function? Discuss its significance.

Ans: When a virtual function is not

redefined by a derived class, the version defined in the base class will be used. When there is no meaningful definition of a virtual function within a base class i.e., when a base class may not be able to define an object sufficiently to allow a base-class virtual function to be created.

Thus we can ensure that all derived classes override a virtual function by using pure virtual function.

A pure virtual function is a virtual function that has no definition within the base class.

To declare a pure virtual function, use this general form:

```
virtual type func-name(parameter-list) = 0;
```

When a virtual function is made pure, any derived class must provide its own definition. If the derived class fails to override the pure virtual function, a compile-time error will result.

Ex:

```
#include <iostream>
```

```
using namespace std;
```

```
class number
```

```
{  
protected:
```

```
    int val;
```

```
public:
```

```
    void setval(int I)
```

```
    {  
        val = i;  
    }
```

```
    // show() is a pure virtual function
```

```
    virtual void show() = 0;
```

```
};
```

```
class hextype : public number
```

```
{
```

```
public:
```

```
    void show()
```

```
    {  
        cout << hex << val << "\n";  
    }
```

```
};
```

```
class dectype : public number
```

```
{
```

```
public:
```

```
    void show()
```

```
    {  
        cout << val << "\n";  
    }
```

```
};
```

```
class octtype : public number
```



```

{
public:
    void show()
    {
        cout << oct << val << "\n";
    }
};

int main()
{
    dectype d;
    hextype h;
    octtype o;
    d.setval(20);
    d.show(); // displays 20 - decimal
    h.setval(20);
    h.show(); // displays 14 - hexadecimal
    o.setval(20);
    o.show(); // displays 24 - octal
    return 0;
}

```

In the above example, a base class may not be able to meaningfully define a virtual function. In **number** class simply provides the common interface for the derived types to use. There is no reason to define show() inside number since the base of the number is undefined. We can always create a placeholder definition of a virtual function. By making show() as pure also ensures that all derived classes will redefine it to meet their own needs.

Q4(a). What is stream? Discuss the four streams which are automatically opened when a C++ Program begins execution.

Ans: A stream is a logical device that either produces or consumes information.

A stream is linked to a physical device by the I/O system. All streams behave in the

same way even though the actual physical devices they are connected to may differ substantially. Because all streams behave the same, the same I/O functions can operate on virtually any type of physical device.

Template Class based Class	Character-based class	Wide-Character-
basic_streambuf	streambuf	wstreambuf
basic_ios	ios	wios
basic_istream	istream	wistream
basic_ostream	ostream	wostream
basic_iostream	iostream	wiostream
basic_fstream	fstream	wfstream
basic_ifstream	ifstream	wifstream
basic_ofstream	ofstream	wofstream

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;
}
```

Q4(b) Explain setw and setfill manipulators with example.

Ans:

setfill(int *ch*) Set the fill character to *ch* .It is used to format an Output.

setw(int *w*) Set the field width to *w* . It is used to format an Output.

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
    cout << hex << 100 << endl;
    cout << setfill('?') << setw(10) << 2343.0;
    return 0;
}
```

This displays

```
64
??????2343
```

Q5(a) What is copy Constructor? Explain with one suitable example.

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.

Q5(b) What is friend function and friend class? Why it is used?

Ans: A friend function of a class is defined outside that class' scope but it has the right to access all private and protected members of the class. Even though the prototypes for friend functions appear in the class definition, friends are not member functions.

A friend can be a function, function template, or member function, or a class or class template, in which case the entire class and all of its members are friends.

To declare a function as a friend of a class, precede the function prototype in the class definition with keyword **friend** as follows:

```
class Box {
    double width;
public:
    double length;
    friend void printWidth( Box box );
    void setWidth( double wid );
};
```

To declare all member functions of class ClassTwo as friends of class ClassOne, place a following declaration in the definition of class ClassOne:

```
friend class ClassTwo;
```

Consider the following program:

```
#include <iostream>

using namespace std;

class Box {
    double width;
public:
    friend void printWidth( Box box );
    void setWidth( double wid );
};

// Member function definition
void Box::setWidth( double wid ) {
    width = wid;
```

```

}

// Note: printWidth() is not a member function of any class.
void printWidth( Box box ) {
    /* Because printWidth() is a friend of Box, it can
    directly access any member of this class */
    cout << "Width of box : " << box.width <<endl;
}

// Main function for the program
int main( ) {
    Box box;

    // set box width with member function
    box.setWidth(10.0);

    // Use friend function to print the width.
    printWidth( box );

    return 0;
}

```

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

```
Width of box : 10
```

Q6(a). What are inserter and extractor? Explain how to create your own inserter and extractor with an example.

Ans: In the language of C++, the << output operator is referred to as the *insertion operator* because it inserts characters into a stream. Likewise, the >> input operator is called the *extraction operator* because it extracts characters from a stream.

Creating Your Own Inserters

It is quite simple to create an inserter for a class that you create. All inserter functions have this general form:

```
ostream &operator<<(ostream &stream, class_type obj)
{
// body of inserter
return stream;
}
```

```
}
```

the function returns a reference to a stream of type **ostream**. Further, the first parameter to the function is a reference to the output stream. The second parameter is the object being inserted.

Creating Your Own Extractors

Extractors are the complement of inserters. The general form of an extractor function is

```
istream &operator>>(istream &stream, class_type &obj)
```

```
{  
// body of extractor  
return stream;  
}
```

Extractors return a reference to a stream of type **istream**, which is an input stream. The first parameter must also be a reference to a stream of type **istream**. Notice that the second parameter must be a reference to an object of the class for which the extractor is overloaded. This is so the object can be modified by the input (extraction) operation.

```
#include <iostream>  
#include <cstring>  
using namespace std;  
class phonebook {  
char name[80];  
int areacode;  
int prefix;  
int num;  
public:  
phonebook() { };  
phonebook(char *n, int a, int p, int nm)  
{  
strcpy(name, n);  
areacode = a;  
prefix = p;  
num = nm;  
}  
friend ostream &operator<<(ostream &stream, phonebook o);  
friend istream &operator>>(istream &stream, phonebook &o);  
};  
// Display name and phone number.  
ostream &operator<<(ostream &stream, phonebook o)  
{  
stream << o.name << " ";  
stream << "(" << o.areacode << ") ";  
stream << o.prefix << "-" << o.num << "\n";  
return stream; // must return stream  
}  
// Input name and telephone number.  
istream &operator>>(istream &stream, phonebook &o)  
{  
cout << "Enter name: ";  
stream >> o.name;  
cout << "Enter area code: ";  
stream >> o.areacode;  
cout << "Enter prefix: ";  
stream >> o.prefix;  
cout << "Enter number: ";  
stream >> o.num;  
cout << "\n";  
return stream;  
}  
int main()  
{  
phonebook a;  
cin >> a;  
cout << a;  
return 0;  
}
```


Q7(a). Explain inline function with an example.

Ans: **Inline Functions:**

If a function is inline, the compiler places a copy of the code of that function at each point where the function is called at compile time.

To inline a function, place the keyword **inline** before the function name and define the function before any calls are made to the function. The compiler can ignore the inline qualifier in case defined function is more than a line.

Ex:

```
#include <iostream>

using namespace std;

inline int Max(int x, int y)
{
    return (x > y)? x : y;
}

// Main function for the program
int main( )
{

    cout << "Max (20,10): " << Max(20,10) << endl;
    cout << "Max (0,200): " << Max(0,200) << endl;
    cout << "Max (100,1010): " << Max(100,1010) << endl;
    return 0;
}
```

Q7(b). What is iterator ? Write a Program to access vector through iterator.

Ans: *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.

```

#include <iostream>
#include <vector>
#include <cctype>
// contents of vector
p = v.begin();
while(p != v.end()) {
    *p = toupper(*p);
    p++;
}
// display contents of vector
cout << "Modified Contents:\n";
p = v.begin();
while(p != v.end()) {
    cout << *p << " ";
    p++;
}
cout << endl;
return 0;
}

```

The output from this program is

```

Original contents:
a b c d e f g h i j
Modified Contents:
A B C D E F G H I J

```

Q8(a). Define

- | | | |
|-----------------|-------------|--------------|
| i) seekg() | ii) seekp() | iii) tellg() |
| iv) precision() | v) fill() | |

Ans: **i) seekg()**

Sets the position of the *get pointer*. The *get pointer* determines the next location to be read in the source associated to the stream.

Syntax:

seekg (position);

Using this function the stream pointer is changed to the absolute position (counting from the beginning of the file).

ii) seekp()

The *seekp* method changes the location of a stream object's file pointer for output (put or write.) In most cases, seekp also changes the location of a stream object's file pointer for input (get or read.)

Syntax:

seekp (position);

Using this function the stream pointer is changed to the absolute position (counting from the beginning of the file).

iii) tellg()

The tellg() function is used with input streams, and returns the current "get" position of the pointer in the stream.

Syntax:

pos_type tellg();

It has no parameters and return a value of the member type pos_type, which is an integer data type representing the current position of the get stream pointer.

iv) precision()

When outputting floating-point values, We can determine the number of digits of precision by using the **precision()** function.

Its prototype is shown here:

```
streamsize precision(streamsize p);
```

Here, the precision is set to *p*, and the old value is returned. The default precision is 6.

In some implementations, the precision must be set before each floating-point output.

If it is not, then the default precision will be used.

v)fill()

By default, when a field needs to be filled, it is filled with spaces. we can specify the fill character by using the **fill()** function.

Its prototype is

```
char fill(char ch);
```

After a call to **fill()**, *ch* becomes the new fill character, and the old one is returned.