2 a. How are data and functions organized in an object-oriented program?

Ans: Data and function are packed inside one module in which data is only accessible by the function of its own module and the function is the only way data can interact with other modules. Encapsulation is the process of combining data and functions into a single unit called class. Using the method of encapsulation, the programmer cannot directly access the data. Data is only accessible through the functions existing inside the class. Data encapsulation led to the important concept of data hiding. Data hiding is the implementation details of a class that are hidden from the user. The concept of restricted access led programmers to write specialized functions or methods for performing the operations on hidden members of the class. Attention must be paid to ensure that the class is designed properly. The organization of data and functions in OOP is shown below:

b. List a few areas of application of OOP technology.

Ans: OOP can be used for such diverse applications as Real-time systems, simulation and modeling, AI and Expert systems parallel programming and Neural networks, Decision support systems, Office automation systems and other. (Explain briefly)

c. What are the applications of void data type in C++? Write a C++ program that illustrates the concept.

Ans: The void data type is used when a function doesn't return any value, and/or when it has no parameters at all. Pointer type 'void *' is a generic pointer.

A void pointer is used when it needs to be assigned to different data types later on in a program. Since it avoids type checking, void pointers should be used with care (Write a small example program)

3 a. Write a program to represent two matrices of dimension 2 X 3 in an array. Write functions for adding and subtracting these two matrices.

Ans: Page No. 88 of Text Book: Object-oriented Programming with C++, Poornachandra Sarang, PHI, 2004

3 b. Write a program to create a structure called date (month, day, year). Accept today`s date and determine tomorrow`s date. (Note: You need to consider end of month and end of year)

Ans: Page No. 89 of Text Book: Object-oriented Programming with C++, Poornachandra Sarang, PHI, 2004

4 a. How does a C++ structure differ from a C++ class? Explain.

Ans:

1. Structures in C++ doesnot provide datahiding where as a class provides datahiding.

2. classes support polymorphism, whereas structures donot.

3. By default everything in structure is public and everything in class is private, but you can change it.

4. By default inheritance of structure is public and inheritance of class is private.

5. The default protection is public for structures and private for classes:

```
struct MyStruct
{
int a; // a is public
};

class MyClass
{
int a; // a is private
};
```

6. The default inheritance is public for structures and private for classes:

```
struct MyOtherStruct : MyStruct
// the same as "struct MyOtherStruct : public MyStruct"
{
};

class MyOtherClass : MyClass
// the same as "class MyOtherClass : private MyClass"
{
};
```

4 c. What are advantages and disadvantages of using In-line function? Show by an example how inline function is used in C++.

Ans: The advantage of in-line functions is that they can be executed much faster than normal functions.
The disadvantage of in-line functions is that if they are too large and called to often, your program grows larger. For this reason, in general only *short* functions are declared as in-line functions.

```
using namespace std;
inline int even(int x) {
return !(x%2);
}
int main( ) {
 if (even(10)) cout << "10 is even\n";
if (even(11)) cout << "11 is even\n";
return 0;
}
```

In this example the function even ( ) which return true if its argument is even, is declared as being in-line.

5 a. List some special properties of the constructor function.

Ans: Special properties of the constructor function:

- They should be declared in the public section.
- They are invoked automatically when the objects are created.
- They do not have return types, not even void and therefore, they cannot return values.
- They cannot be inherited, though a derived class can call the base class constructor.
- Like other C++ functions, they can have default arguments.
- Constructors cannot be virtual.

5b. Write a C++ program that overload '+' operator to add two coordinates from a class 'coord' having X and Y-coordinate.

Ans:

```
#include <iostream>
using namespace std;
class coord {
int x, y; // coordinate values
public:
coord( ) { x = 0; y = 0; }
coord(int i, int j) { x = i; y = j; }
void get_xy(int &i, int &j) { i = x; j = y; }
coord operator+(coord ob2);
};
// Overload + relative to coord class.
coord coord::operator+(coord ob2) {
coord temp;
temp.x = x + ob2.x;
temp.y = y + ob2.y;
return temp;
}
int main( ) {
```

```
coord o1(10, 10), o2(5, 3), o3;
int x, y;
o3 = o1 + o2; //add to objects,
// this calls operator+()
o3.get_xy(x, y);
cout << "(o1+o2) X: " << x << ", Y: " << y << "\n";
return 0;
 }
```

6 a. "When a base class and a derived class both have constructor and destructor functions, the constructor functions are executed in order of derivation. The destructor functions are executed in reverse order." Justify the statement giving suitable C++ statement.

Ans:

```
#include <iostream>
using namespace std;
class base {
public:
base( ) { cout << "Constructing base\n"; }
~base( ) { cout << "Destructing base\n"; }
};
class derived : public base {
public:
derived( ) { cout << "Constructing derived\n"; }
};
int main( ) {
derived obj;
return 0;
}
```

This program displays:

```
Constructing base
Constructing derived
Destructing derived
Destructing base
o3 = o1 + o2; //add to objects,
// this calls operator+()
o3.get_xy(x, y);
cout << "(o1+o2) X: " << x << ", Y: " << y << "\n";
return 0;
}
```

6 b. What is a virtual base class? When do we make a class virtual? Illustrate with a suitable C++ program the concept of virtual class.

Ans: A potential problem exists when multiple base classes are directly inherited by a derived class. For example the base class *Base* is inherited by both *Derived1* and *Derived2*. *Derived3* directly inherits both *Derived1* and *Derived2*. However, this implies that *Base* is actually inherited twice by *Derived3*. First it is inherited through *Derived1*, and then again through *Derived2*. This causes ambiguity when a member of *Base* is used by *Derived3*. Since two copies of *Base* are included in *Derived3*, is a reference to a member of *Base* referring to the *Base* inherited indirectly through *Derived1* or to the *Base* inherited indirectly through *Derived2*? To resolve this ambiguity, C++ includes a mechanism by which only one copy of *Base* will be included in *Derived3*. This feature is called a *virtual base class*.

In situations like this, in which a derived class indirectly inherits the same base class more than once, it is possible to prevent multiple copies of the base from being present in the derived class by having that base class inherited as virtual by any derived classes. Doing this prevents two or more copies of the base from being present in any subsequent derived class that inherits the base class indirectly. The virtual keyword precedes the base class access specifier when it is inherited by a derived class.

```
// This program uses a virtual base class.
#include <iostream>
using namespace std;
class Base {
public:
int i;
};
// Inherit Base as virtual
class Derived1 : virtual public Base {
public:
int j;
};
// Inherit Base as virtual here, too
class Derived2 : virtual public Base {
public:
int k;
};
// Here Derived3 inherits both Derived1 and Derived2.
// However, only one copy of base is inherited.
class Derived3 : public Derived1, public Derived2 {
public:
int product( ) { return i*j*k; }
};
int main( ) {
Derived3 ob;
ob.i = 10; // unambiguous because virtual Base
ob.j = 3;
ob.k = 5;
```

```
    cout << "Product is: " << ob.product( ) << "\n";
    return 0;
}
```

7 a. What is polymorphism? Discuss in brief two different types of polymorphism.

Ans: Explanation of Static and Dynamic polymorphism

7 b. Can you write an exception handler that catch all exceptions instead of just a certain type? How?

Ans:

In some circumstances we may want an exception handler to catch all exceptions instead of just a certain type. For this, use this form of catch:

```
    catch(...) {
    // process all exception
    }
```

Also, you can control what type of exceptions a function can throw outside itself. In fact, you can also prevent a function from throwing any exceptions whatsoever. To apply these restrictions, you must add a throw clause to the function definition. The general form is as follows,

ret-type-func-name(*arg-list*) throw(*type-list*)

```
    {
    // ....
}
    #include <iostream>
    using namespace std;
    void Xhandler(int test) {
    try {
    if (test==0) throw test; // throw int
    if (test==1) throw 'a'; // throw char
    if (test==2) throw 123.23;// throw double
    }
    catch(...) { // catch all exceptions
    cout << "Caught one!\n";
    }
    }
    int main( ) {
    cout << "start\n";
  Xhandler(0);
    Xhandler(1);
    Xhandler(2);
    cout << "end";
    return 0;
    }
```

This program displays:

```
start
Caught one!
Caught one!
Caught one!
end
```

7 c. Write an example C++ program that shows ‘re-throwing an exception’ concept.

Ans:

```
// Rethrowing an exception
#include <iostream>

using namespace std;
void Xhandler( ) {
try {
catch(char *) { // catch a char *
cout << "Caught char * inside Xhandler\n";
throw ; // rethrow char * out of function
}
}
int main( ) {
cout << "start\n";
try {
Xhandler( );
}
catch(char *) {
cout << "Caught char * inside main\n";
}
cout << "end";
return 0;
}
```

This program displays:

```
start
Caught char * inside Xhandler
Caught char * inside main
end
```

8 a. What do you mean by ‘Generic function’? Write a generic function that swaps the values of the two variables it is called with.

Ans:
```
#include <iostream>
using namespace std;
// This is a function template
template <class X> void swapargs(X &a, X &b) {
X temp;
temp = a;
a = b;
b = temp;
}
int main( ) {
int i=10, j=20;
float x=10.1, y=23.3;
cout << "Original i, j: " << i << j <<endl;
 cout << "Original x, y: " << x << y <<endl;
swapargs(i, j); // swap integers
swapargs(x, y); // swap floats
cout << "Swapped i, j: " << i << j <<endl;
cout << "Swapped x, y: " << x << y <<endl;
return 0;
}
```

8 b. What is the basic difference between a template and an overloaded function? Show by a suitable program in C++, how generic functions can be overloaded.

Ans: Generic functions are similar to overloaded functions except they are more restrictive. When functions are overloaded, you can have different actions performed with the body of each function. But generic functions *must perform the same general action for all versions*.
Even though a template function overloads itself as needed, you can explicitly overload one, too. If you overload a generic function, that overloaded function overrides (or 'hides') the generic function relative to that specific version.

```
// Function template example
#include <iostream>
using namespace std;
// Overriding a template function
template <class X> void swapargs(X &a, X &b) {
X temp;
temp = a;
a = b;
b = temp;
}
// This overrides the generic version of swapargs( )
void swapargs(int a, int b) {
cout << "this is inside swapargs(int, int)\n";
```

```
}
int main( ) {
int i=10, j=20;
float x=10.1, y=23.3;
cout << "Original i, j: " << i << j <<endl;
cout << "Original x, y: " << x << y <<endl;
swapargs(i, j); // calls overloaded swapargs( )
swapargs(x, y); // swap floats
cout << "Swapped i, j: " << i << j <<endl;
cout << "Swapped x, y: " << x << y <<endl;
return 0;
}
```

9 a. Explain return value and meaning of the following error handling functions during file operations: eof(), fail(), bad(), good()

Ans: The C++ I/O system can determine whether an error has occurred is by using one of these ios member functions:

```
bool bad( );
bool eof( );
bool fail( );
bool good( );
```

The eof( ) function returns true (non-zero value) if end-of-file is encountered while reading; otherwise returns false (zer). The bad( ) function returns true if badbit is set. The fail( ) function returns true if failbit is set. The good( ) function returns true if there are no errors. Otherwise, they return false.

(Write a small C++ program that shows use of these function).

9 b. Write a programme to copy contents of file story.txt into newstory.txt

Ans:

```
#include <fstream.h>
void main()
{
char ch,ch1;
ifstream y;
y.open("story.txt");
ofstream u;
u.open("newstory.txt");
while(ch!=EOF)
{
ch=y.get();
```

```
u.put(ch);
}
y.close();
u.close();
}
ifstream r;
 r.open("newstory.txt");
while(ch1!=EOF)
{
ch1=r.get();
cout<<ch1;
}
}
```