For 100% Result Oriented IGNOU Coaching and Project Training
Call CPD: 011-65164822, 08860352748

Question 1:

Explain why Object Oriented Programming approach is better than Structured Programming Approach.
(5 Marks)

ANS 1 (a)

Structured programming is a programming paradigm aimed on improving the clarity, quality, and development time of a computer program by making extensive use of subroutines, block structures and for and while loops—in contrast to using simple tests and jumps such as the goto statement which could lead to spaghetti code which is both difficult to follow and to maintain.

Object-oriented programming (OOP) is a programming paradigm that represents concepts as "objects" that have data fields (attributes that describe the object) and associated procedures known as methods. Objects, which are usually instances of classes, are used to interact with one another to design applications and computer programs.

Object Oriented Programming has many benefits over Structured programming. Some of them are reusability, extensibility, reliability and maintainability. OOP also helps to reduce large problems to smaller, more manageable problems. In terms of extensibility and reusability, for instance: Encapsulation allows the internal implementation of a class to be modified without requiring changes to its services (i.e. methods). It also allows new classes to be added to a system, without major modifications to the system. Inheritance allows the class hierarchy to be further refined, and combined with polymorphism, the superclass does not have to “know” about the new class, i.e. modifications do not have to be made at the superclass.

component-specific behavior - making details on how to handle a particular component the responsibility of the smaller component-specific machine ensures any time that component is handled, its machine will do so appropriately;

polymorphic expressions - because component-specific machines performs operations tailored to its particular component, the same message sent to different machines can act differently;

type abstraction - it often makes sense for several different types of components to use the same vocabulary for the operations their machines do;

separation of concerns - leaving component-specific details to their machines means the process machine only needs to handle the more general, larger concerns of its process and the data required to manage it; plus, it's less likely to be affected by changes in other components;

For 100% Result Oriented IGNOU Coaching and Project Training
Call CPD: 011-65164822, 08860352748

adaptability - components that focus on their area of speciality can be adapted to unforeseen use simply by changing the components it uses, or making it available to another process machine;

code reuse - components with a narrow focus and greater adaptability can leverage their development cost by being put to use more often.

Question 1: (b)
Write a simple C++ program to explain the basic structure of C++ programming. **(5 Marks)**

Ans 1 (b)

```
// Header Files Declaration Section

#include<iostream.h>

#include<conio.h>

// Global Declaration Section

class book

{

 private :

            int bookno;

            char bname[30];

            char auname[30];

            float bprice;

 public :

           void getdata()

           {

            cout<<"Enter the details"<<endl;

            cout<<"Enter the book no:"; cin>>bookno;
```

CPD

For 100% Result Oriented IGNOU Coaching and Project Training
Call CPD: 011-65164822, 08860352748

```
        cout<<"Enter the book name: ";cin>>bname;
        cout<<"Enter author name : "; cin>>auname;
        cout<<"Enter the book price: ";cin>>bprice;
       }
       void display()
       {
        cout<<endl<<"The entered record is: "<<endl;
        cout<<"Book no: "<<bookno<<endl;
        cout<<"Book name: "<<bname<<endl;
        cout<<"Author name: "<<auname<<endl;
        cout<<"Book price: "<<bprice<<endl;
       }
};
// Main Function Section
int main()
{
 class book obj;
 clrscr();
 obj.getdata();
 obj.display();
 getch();
 return 0;
}
```

CPD

For 100% Result Oriented IGNOU Coaching and Project Training
Call CPD: 011-65164822, 08860352748

Question1 : (c)Explain the usage of the following C++ operators with the help of an example program.
(6 Marks)

(a) sizeof operator

Ans 1 (c) (a)

The sizeof is a keyword, but it is a compile-time operator that determines the size, in bytes, of a variable or data type.

The sizeof operator can be used to get the size of classes, structures, unions and any other user defined data type.

The syntax of using sizeof is as follows:

sizeof (data type)

Where data type is the desired data type including classes, structures, unions and any other user defined data type.

Try following example to understand all the sizeof operator available in C++. Copy and paste following C++ program in test.cpp file and compile and run this program.

```
#include <iostream>

using namespace std;


int main()

{

  cout << "Size of char : " << sizeof(char) << endl;

  cout << "Size of int : " << sizeof(int) << endl;

  cout << "Size of short int : " << sizeof(short int) << endl;

  cout << "Size of long int : " << sizeof(long int) << endl;

  cout << "Size of float : " << sizeof(float) << endl;

  cout << "Size of double : " << sizeof(double) << endl;
```

For 100% Result Oriented IGNOU Coaching and Project Training
Call CPD: 011-65164822, 08860352748

```
  cout << "Size of wchar_t : " << sizeof(wchar_t) << endl;

  return 0;

}
```

When the above code is compiled and executed, it produces following result which can vary from machine to machine:

Size of char : 1

Size of int : 4

Size of short int : 2

Size of long int : 4

Size of float : 4

Size of double : 8

Size of wchar_t : 4

Question1 (c) (b) Logical Operators

Ans 1 (c) (b)

The logical AND operator (&&) returns the boolean value true if both operands are true and returns falseotherwise. The operands are implicitly converted to type bool prior to evaluation, and the result is of type bool. Logical AND has left-to-right associativity.

```
// expre_Logical_AND_Operator.cpp

// compile with: /EHsc

// Demonstrate logical AND

#include <iostream>

using namespace std;

int main() {

  int a = 5, b = 10, c = 15;
```

For 100% Result Oriented IGNOU Coaching and Project Training
Call CPD: 011-65164822, 08860352748

```
  cout  << boolalpha
        << "The true expression "
        << "a < b && b < c yields "
        << (a < b && b < c) << endl
        << "The false expression "
        << "a > b && b < c yields "
        << (a > b && b < c) << endl;
}
```

The logical OR operator (||) returns the boolean value true if either or both operands is true and returns falseotherwise. The operands are implicitly converted to type bool prior to evaluation, and the result is of type bool. Logical OR has left-to-right associativity.

```
// expre_Logical_OR_Operator.cpp
// compile with: /EHsc
// Demonstrate logical OR
#include <iostream>
using namespace std;
int main() {
  int a = 5, b = 10, c = 15;
  cout  << boolalpha
        << "The true expression "
        << "a < b || b > c yields "
        << (a < b || b > c) << endl
        << "The false expression "
```

For 100% Result Oriented IGNOU Coaching and Project Training
Call CPD: 011-65164822, 08860352748

```
    << "a > b || b > c yields "
    << (a > b || b > c) << endl;
}
```

The logical negation operator (!) reverses the meaning of its operand. The operand must be of arithmetic or pointer type (or an expression that evaluates to arithmetic or pointer type). The operand is implicitly converted to type bool. The result is true if the converted operand is false; the result is false if the converted operand is true. The result is of type bool.

```
// expre_Logical_NOT_Operator.cpp
// compile with: /EHsc
#include <iostream>
using namespace std;

int main() {
  int i = 0;
  if (!i)
    cout << "i is zero" << endl;
}
```

Question 1 (c) (c) Scope resolution operator

Ans 1 (c) (c)

In computer programming, scope is an enclosing context where values and expressions are associated. The scope resolution operator helps to identify and specify the context to which an identifier refers. The specific uses vary across different programming languages with the notions of scoping.

The scope resolution operator (::) in C++ is used to define the already declared member functions (in the header file with the .hpp or the .h extension) of a particular class. In the .cpp file one can define the usual global functions or the member functions of the class. To differentiate between the normal

For 100% Result Oriented IGNOU Coaching and Project Training
Call CPD: 011-65164822, 08860352748

functions and the member functions of the class, one needs to use the scope resolution operator (::) in between the class name and the member function name i.e. ship::foo() where ship is a class and foo() is a member function of the class ship. The other uses of the resolution operator is to resolve the scope of a variable when the same identifier is used to represent a global variable, a local variable, and members of one or more class(es).

Example:

This example has two variables named amount. The first is global and contains the value 123. The second is local to the main function. The scope resolution operator tells the compiler to use the global amount instead of the local one.

```
// expre_ScopeResolutionOperator.cpp

// compile with: /EHsc

// Demonstrate scope resolution operator

#include <iostream>

using namespace std;

int amount = 123;   // A global variable

int main() {

  int amount = 456;   // A local variable

  cout  << ::amount << endl   // Print the global variable

       << amount << endl;    // Print the local variable

}
```

Question 2 (a) Define the class Book with all the basic attributes such as title, author, publisher, price etc. Define the default constructor, member functions **display_data()** for displaying the Book details. Use appropriate access control specifiers in this program. **(8 Marks)**

Ans 2(a)

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

For 100% Result Oriented IGNOU Coaching and Project Training
Call CPD: 011-65164822, 08860352748

```
#include<conio.h>
class book
{
 private :
            int bookno;
            char bname[30];
            char auname[30];
            float bprice;
 public :
           void getdata()
           {
            cout<<"Enter the details"<<endl;
            cout<<"Enter the book no:"; cin>>bookno;
            cout<<"Enter the book name: ";cin>>bname;
            cout<<"Enter author name : "; cin>>auname;
            cout<<"Enter the book price: ";cin>>bprice;
           }
           void display_data()
           {
            cout<<endl<<"The entered record is: "<<endl;
            cout<<"Book no: "<<bookno<<endl;
            cout<<"Book name: "<<bname<<endl;
            cout<<"Author name: "<<auname<<endl;
```

For 100% Result Oriented IGNOU Coaching and Project Training
Call CPD: 011-65164822, 08860352748

```
        cout<<"Book price: "<<bprice<<endl;
       }
};
int main()
{
 class book obj;
 clrscr();
 obj.getdata();
 obj.display_data();
 getch();
 return 0;
}
```

Question 2 (b) Explain the following terms in the context of object oriented programming. Also explain how these concepts are implemented in C++ by giving an example program for each. **(8Marks)**

(a) Abstraction

Ans 2 (b) (a)

DATA ABSTRACTION Definition- The act of representing essential features without including the background details or explanations.

Data Abstraction Example:

Any C++ program where you implement a class with public and private members is an example of data abstraction. Consider the following example:

```
#include <iostream>
using namespace std;
class Adder{
  public:
```

For 100% Result Oriented IGNOU Coaching and Project Training
Call CPD: 011-65164822, 08860352748

```
    // constructor
    Adder(int i = 0)
    {
      total = i;
    }
    // interface to outside world
    void addNum(int number)
    {
       total += number;
    }
    // interface to outside world
    int getTotal()
    {
       return total;
    };
  private:
    // hidden data from outside world
    int total;
};
int main( )
{
  Adder a;
```

CPD

For 100% Result Oriented IGNOU Coaching and Project Training
Call CPD: 011-65164822, 08860352748

```
  a.addNum(10);
  a.addNum(20);
  a.addNum(30);
 cout << "Total " << a.getTotal() <<endl;
  return 0;
}
```

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

Total 60

Above class adds numbers together, and returns the sum. The public members addNum and getTotalare the interfaces to the outside world and a user needs to know them to use the class. The private member total is something that the user doesn't need to know about, but is needed for the class to operate properly.

(b) Encapsulation

Ans 2 (b) (b)

DATA ENCAPSULATION Definition – The wrapping up of data and functions into a single unit (which is called class). Encapsulation means that some or all of an object's internal structure is hidden from the outside world. Hidden information may only be accessed through the object's methods, called the object's public interface. Access to object is safe, controlled.

Data Encapsulation Example:

Any C++ program where you implement a class with public and private members is an example of data encapsulation and data abstraction. Consider the following example:

```
#include <iostream>
using namespace std;
class Adder{
  public:
    // constructor
```

For 100% Result Oriented IGNOU Coaching and Project Training
Call CPD: 011-65164822, 08860352748

```
    Adder(int i = 0)
    {
      total = i;
    }
    // interface to outside world
    void addNum(int number)
    {
       total += number;
    }
    // interface to outside world
    int getTotal()
    {
       return total;
    };
  private:
    // hidden data from outside world
    int total;
};
int main( )
{
  Adder a;
  a.addNum(10);
  a.addNum(20);
```

CPD

For 100% Result Oriented IGNOU Coaching and Project Training
Call CPD: 011-65164822, 08860352748

```
  a.addNum(30);

 cout << "Total " << a.getTotal() <<endl;

  return 0;

}
```

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

Total 60

Above class adds numbers together, and returns the sum. The public members addNum and getTotalare the interfaces to the outside world and a user needs to know them to use the class. The private member total is something that is hidden from the outside world, but is needed for the class to operate properly.

(c) Operator Overloading

Ans 2 (b) (c)

Operators overloading in C++:

You can redefine or overload most of the built-in operators available in C++. Thus a programmer can use operators with user-defined types as well.

Overloaded operators are functions with special names the keyword operator followed by the symbol for the operator being defined. Like any other function, an overloaded operator has a return type and a parameter list.

Box operator+(const Box&);

declares the addition operator that can be used to add two Box objects and returns final Box object. Most overloaded operators may be defined as ordinary non-member functions or as class member functions. In case we define above function as non-member function of a class then we would have to pass two arguments for each operand as follows:

Box operator+(const Box&, const Box&);

Following is the example to show the concept of operator over loading using a member function. Here an object is passed as an argument whose properties will be accessed using this object, the object which will call this operator can be accessed using this operator as explained below:

For 100% Result Oriented IGNOU Coaching and Project Training
Call CPD: 011-65164822, 08860352748

```
#include <iostream>

using namespace std;

class Box

{

  public:

 double getVolume(void)

    {

      return length * breadth * height;

    }

    void setLength( double len )

    {

       length = len;

    }

 void setBreadth( double bre )

    {

       breadth = bre;

    }

void setHeight( double hei )

    {

       height = hei;

    }

    // Overload + operator to add two Box objects.

    Box operator+(const Box& b)
```

CPD

For 100% Result Oriented IGNOU Coaching and Project Training
Call CPD: 011-65164822, 08860352748

```
    {
      Box box;
      box.length = this->length + b.length;
      box.breadth = this->breadth + b.breadth;
      box.height = this->height + b.height;
      return box;
    }
  private:
    double length;     // Length of a box
    double breadth;    // Breadth of a box
    double height;     // Height of a box
};
// Main function for the program
int main( )
{
  Box Box1;            // Declare Box1 of type Box
  Box Box2;            // Declare Box2 of type Box
  Box Box3;            // Declare Box3 of type Box
  double volume = 0.0;    // Store the volume of a box here
 // box 1 specification
  Box1.setLength(6.0);
  Box1.setBreadth(7.0);
  Box1.setHeight(5.0);
```

For 100% Result Oriented IGNOU Coaching and Project Training
Call CPD: 011-65164822, 08860352748

```
   // box 2 specification
   Box2.setLength(12.0);
   Box2.setBreadth(13.0);
   Box2.setHeight(10.0);
  // volume of box 1
   volume = Box1.getVolume();
   cout << "Volume of Box1 : " << volume <<endl;
  // volume of box 2
   volume = Box2.getVolume();
   cout << "Volume of Box2 : " << volume <<endl;
// Add two object as follows:
   Box3 = Box1 + Box2;
 // volume of box 3
   volume = Box3.getVolume();
   cout << "Volume of Box3 : " << volume <<endl;
return 0;
}
```

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

```
Volume of Box1 : 210
Volume of Box2 : 1560
Volume of Box3 : 5400
```

For 100% Result Oriented IGNOU Coaching and Project Training
Call CPD: 011-65164822, 08860352748

(d) Static Member

Ans 2 (b) (d)

Static Members

We can define class members static using static keyword. When we declare a member of a class as static it means no matter how many objects of the class are created, there is only one copy of the static member.

A static member is shared by all objects of the class. All static data is initialized to zero when the first object is created, if no other initialization is present. We can't put it in the class definition but it can be initialized outside the class as done in the following example by redeclaring the static variable, using the scope resolution operator :: to identify which class it belongs to.

Example:

```
#include<iostream.h>
#include<conio.h>
class example
{
        private :  static int num;
        public :
           static int total()
                        {
                                return num++;
                        }
                        void out()
                        {
                                cout<<"objects "<<num<<" In created & count= "<<num<<endl;
                        }
```

For 100% Result Oriented IGNOU Coaching and Project Training
Call CPD: 011-65164822, 08860352748

```
};

int example :: num;

int main()

{

        clrscr();

        example obj1;

        example :: total();

        obj1.out();

        example obj2;

        example :: total();

        obj2.out();

        getch();

        return 0;

}
```

Question 3:

(a) What is polymorphism? What are different forms of polymorphism? Explain implementation of polymorphism with the help of a C++ program. **(8 Marks)**

Ans 3 (a)

Polymorphism: The word 'poly' is a Greek word meaning many and 'morphism' means forms. Thus Polymorphism means many forms. Polymorphism is one of the concepts of OOP. Polymorphism is defined as the ability of a member function to act differently in different situations which in programming sense means that one or more than one member functions can be defined with the same name in one or more than one classes and then the member function performs different operations according to different number, order and types of arguments passes to it. The same concept can also be applied in many other areas. Polymorphism is generally of following types:

For 100% Result Oriented IGNOU Coaching and Project Training
Call CPD: 011-65164822, 08860352748

Function overloading

Function overriding

Late/run-time binding

Operator overloading

Example Program: This program illustrates the late binding type of polymorphism.

```
#include<iostream.h>
#include<conio.h>
class personal
{
        private:
                char name[20];
                int age;
                char sex;
        public:
                virtual void getdata();
                virtual void display();
};
class academic : public personal
{
        private:
                char name[20];
                char trade[20];
                int rollno;
```

CPD

For 100% Result Oriented IGNOU Coaching and Project Training
Call CPD: 011-65164822, 08860352748

```
        public:
                void getdata();
                void display();
};
void personal :: getdata()
{
        cout<<"(virtual function)***Enter the data***"<<endl;
        cout<<"Name : ";
        cin>>name;
        cout<<"Age : ";
        cin>>age;
        cout<<"Sex : ";
        cin>>sex;
}
void personal :: display()
{
        cout<<"***The entered data is***" <<endl;
        cout<<"Name : "<<name<<endl;
        cout<<"Age : "<<age<<endl;
        cout<<"Sex : "<<sex<<endl;
}
void academic  ::  getdata()
{
```

For 100% Result Oriented IGNOU Coaching and Project Training
Call CPD: 011-65164822, 08860352748

```
        cout<<" (member function of class academic called) "<<endl;
        cout<<"***Enter the data***"<<endl;
        cout<<"Name : ";
        cin>>name;
        cout<<"Trade : ";
        cin>>trade;
        cout<<"Rollno : ";
        cin>>rollno;
}
void academic :: display()
{
        cout<<"***The entered data is***" <<endl;
        cout<<"Name : "<<name<<endl;
        cout<<"Trade : "<<trade<<endl;
        cout<<"Rollno : "<<rollno<<endl;
}
int main()
{
        class personal *ptr;
        class academic obj;
        ptr = &obj;
        clrscr();
        ptr -> getdata();
```

CPD

For 100% Result Oriented IGNOU Coaching and Project Training
Call CPD: 011-65164822, 08860352748

```
        ptr -> display();
getch();
return 0;
}
```

(b) What is friend function? How it is implemented in C++? Explain advantages of using friend function with the help of an example. **(8 Marks)**

Ans 3 (b)

There come very few situations when non-public data of a class is to be accessed by the member which do not belong to the class. Under these circumstances, friend functions are needed to help in accessing non- public members of a class. Thus friend function is a function which is not a member of the class but is used to access non-public members of that class.

Friend function is implemented in c++ by using 'friend' keyword as follows:

```
friend   return_type   user_defined_name(arguments list)
```

Friend function example program:

```
#include<iostream.h>
#include<conio.h>
class biggest
{
        private :
                        int a, b, c, large;
        public :
                        void getdata();
        friend int big (biggest abc);        // friend function declared
};
```

For 100% Result Oriented IGNOU Coaching and Project Training
Call CPD: 011-65164822, 08860352748

```
void biggest :: getdata ()
{
        cout<<"Enter any three numbers : "<<endl;
        cin>>a>>b>>c;
}
int big(biggest abc)
{
        abc.large = abc.a;                  // accessing of private data members
        if (abc.b > abc.large)
        {
                abc.large = abc.b;
        }
        if (abc.c > abc.large)
        {
                abc.large = abc.c;
        }
cout<<endl<<"The biggest of the entered numbers is : "<<abc.large;
return 0;
}
int main()
{
        class biggest obj;
        clrscr();
```

For 100% Result Oriented IGNOU Coaching and Project Training
Call CPD: 011-65164822, 08860352748

```
    obj.getdata();

    big(obj);                    // calling of friend function

    getch();

    return 0;

}
```

Question 4 :

(a) Explain the following functions for manipulating file pointers, with the help of example program: **(8 Marks)**

- seekg()
- seekp()
- tellg()
- tellp()

Ans 4 (a)

Every file in C++ has two pointers called get_pointer and put_pointer which tells the current position of the file pointer and moves the file pointer on any location with in the file. get_pointer works with input mode file and put_pointer works with output mode file. These pointers help to attain random access in files. That means to moving directly to any location in the file instead of moving through it sequentially.

Seekg() : seekg() Sets the position of the get pointer in any location in input mode file.

Seekp() : Sets the position of the put pointer in any location in output mode file.

Tellg() : tellg() returns the current position of get pointer.

Tellp() : tellp() returns the current position of put pointer.

Example :

```
int main()

{

    ofstream fout;
```

For 100% Result Oriented IGNOU Coaching and Project Training
Call CPD: 011-65164822, 08860352748

```
int i; char ch;
// File opening in  output mode both
fout.open("d:/test.txt",ios::in);
if(!fout)
{
        cout<<"Error in creating file..\n"; return 0;
}
// Writting A-Z in file.
for(i=0;i<26;i++){
        fout.put((char)(65+i));
}
cout<<"Total bytes are : "<< fout.tellp() << endl;
fout.close();
// reading file ..
ifstream fin;
fin.open("d:/test.txt",ios::in);
if(!fin)
{
        cout<<"Error in opening file..\n"; return 0;
}
cout<<"\nCASE-1:\nFile's get pointer is in " << fin.tellg() <<" Position. \n";
cout<<"Characters are ... :\n";
while(!fin.eof()){
```

For 100% Result Oriented IGNOU Coaching and Project Training
Call CPD: 011-65164822, 08860352748

```
                fin.get(ch);
                cout<< ch;
        }
        // sets the get pointer to 10th byte
        fin.clear(); // reset flags
        fin.seekg(10,ios::beg);
        cout<<"\n\nCASE-2:\nFile's get pointer is in " << fin.tellg() <<" Position. \n";
        cout<<"Characters are ... :\n";
        while(!fin.eof()){
                fin.get(ch);
                cout<< ch;
        }
// sets the get pointer to -20 bytes back
        fin.clear(); // reset flags
        fin.seekg(-20,ios::cur);
        cout<<"\n\nCASE-2:\nFile's get pointer is in " << fin.tellg() <<" Position. \n";
        cout<<"Characters are ... :\n";
        while(!fin.eof()){
                fin.get(ch);
                cout<< ch;
        }
        cout<< endl;
return 1;
```

For 100% Result Oriented IGNOU Coaching and Project Training
Call CPD: 011-65164822, 08860352748

}

Output

Total bytes are : 26

CASE-1:

File's get pointer is in 0 Position.

Characters are ... :

ABCDEFGHIJKLMNOPQRSTUVWXYZ

CASE-2:

File's get pointer is in 10 Position.

Characters are ... :

KLMNOPQRSTUVWXYZ

CASE-2:

File's get pointer is in 6 Position.

Characters are ... :

GHIJKLMNOPQRSTUVWXYZ

(b) What is an exception? How an exception is different from an error? Explain how exceptions are handled in C++, with the help of an example program. **(8 Marks)**

Ans 4 (b)

An exception is a hardware generated condition which is triggered when processing fails for some reason in a CPU. It could be caused by dividing by zero or trying to access memory at a non-existent address. Thus an exception is a problem that arises during the execution of a program. A C++ exception is a response to an exceptional circumstance that arises while a program is running.

Difference between exception and error:

An error can occur at compile time or run-time, but an exception always occurs at the run-time.

For 100% Result Oriented IGNOU Coaching and Project Training
Call CPD: 011-65164822, 08860352748

An error is caused due to syntactic or logical mistake of the programmer, but an exception is caused by the CPU.

An error is handled by correcting the errorneous code in program but an exception is handled by the exception-handling mechanism of the programming language.

Handling of Exceptions in c++ :

Exceptions provide a way to transfer control from one part of a program to another. C++ exception handling is built upon three keywords: try, catch, and throw.

throw: A program throws an exception when a problem shows up. This is done using a throwkeyword.

catch: A program catches an exception with an exception handler at the place in a program where you want to handle the problem. The catch keyword indicates the catching of an exception.

try: A try block identifies a block of code for which particular exceptions will be activated. It's followed by one or more catch blocks.

Assuming a block will raise and exception, a method catches an exception using a combination of thetry and catch keywords. A try/catch block is placed around the code that might generate an exception.

Code within a try/catch block is referred to as protected code, and the syntax for using try/catch looks like the following:

```
try
{
  // protected code
}catch( ExceptionName e1 )
{
  // catch block
}catch( ExceptionName e2 )
{
  // catch block
```

For 100% Result Oriented IGNOU Coaching and Project Training
Call CPD: 011-65164822, 08860352748

```
}catch( ExceptionName eN )
{
  // catch block
}
```

You can list down multiple catch statements to catch different type of exceptions in case your try block raises more than one exceptions in different situations.

The following is an example which throws a division by zero exception and we catch it in catch block.

```
#include <iostream>
using namespace std;
double division(int a, int b)
{
  if( b == 0 )
  {
     throw "Division by zero condition!";
  }
  return (a/b);
}
int main ()
{
  int x = 50;
  int y = 0;
  double z = 0;
 try {
```

For 100% Result Oriented IGNOU Coaching and Project Training
Call CPD: 011-65164822, 08860352748

```
    z = division(x, y);
  cout << z << endl;
  }catch (const char* msg) {
    cerr << msg << endl;
  }
  return 0;
}
```

Because we are raising an exception of type const char*, so while catching this exception, we have to use const char* in catch block. If we compile and run above code, this would produce following result:

Division by zero condition!

Question 5:

(a) What is template? Write appropriate statements to create a template class for Stack data structure in C++. **(8 Marks)**

Ans 5(a)

Definition: A template is a mechanism that allows us to create functions and classes that can be defined with any data type.

Template class in c++ for stack data structure:

Template class in c++ for stack data structure: This program takes the simple stack given elsewhere, and writes it in a "templated" version, that is, in generic form using C++ templates. In the listings below, code related to templates appears in red, while parameters to the template are given in orange. In the implementation, whenever a stack is instantiated, the actual parameters to the template are textually substituted in for the formal parameters to the stack. Thus below, first char and 10 are substituted for Typ and MaxStack. Next double and 4 are substituted for Typ and MaxStack. After this substitution, the compiler compiles the resulting code, which contains only references to specific types, and so can be optimized.

The main function below shows the instantiation of two stacks, the first a stack of up to 10 chars, and the second a stack of up to 4 doubles. The code pushes items on each stack until they are full, and then

For 100% Result Oriented IGNOU Coaching and Project Training
Call CPD: 011-65164822, 08860352748

pops them for printing until the stack is empty. As in the earlier programs, this code is simplified, without even any checking for underflow or overflow. This version allocated the actual memory locations for the stack on the program stack, so here there is no memory leak.

This version will work for user-defined types also. These C++ features allow the creation of much more complex templated "container" classes in the C++ Standard Template Library (STL).

C++ Stack With Template: Three separate files: stack.h, stack.cpp, and stacktest.cpp

```
// stack.h: header file

template< class Typ, int MaxStack >

class Stack {

  int EmptyStack;

  Typ items[MaxStack];

  int top;

public:

  Stack();

  ~Stack();

  void push(Typ);

  Typ pop();

  int empty();

  int full();

};

 // stack.cpp: function definitions

#include "stack.h"

template< class Typ, int MaxStack >

Stack< Typ, MaxStack >::Stack() {
```

For 100% Result Oriented IGNOU Coaching and Project Training
Call CPD: 011-65164822, 08860352748

```
  EmptyStack = -1;
  top = EmptyStack;
}
template< class Typ, int MaxStack >
Stack< Typ, MaxStack >::~Stack()
{  delete[] items; }
template< class Typ, int MaxStack >
void Stack< Typ, MaxStack >::push(Typ c)
{  items[ ++top ] = c; }
template< class Typ, int MaxStack >
Typ Stack< Typ, MaxStack >::pop()
{  return items[ top-- ]; }
template< class Typ, int MaxStack >
int Stack< Typ, MaxStack >::full()
{  return top + 1 == MaxStack; }
template< class Typ, int MaxStack >
int Stack< Typ, MaxStack >::empty()
{  return top == EmptyStack; }
 // stacktest.cpp: use templated stack
#include <iostream.h>
#include "stack.h"
int main() {
  Stack<char, 10> s; // 10 chars
```

For 100% Result Oriented IGNOU Coaching and Project Training
Call CPD: 011-65164822, 08860352748

```
  char ch;
  while ((ch = cin.get()) != '\n')
    if (!s.full()) s.push(ch);
  while (!s.empty())
    cout << s.pop();
  cout << endl;
  Stack<double, 4> ds; // 4 doubles
  double d[] =
    {1.0, 3.0, 5.0, 7.0, 9.0, 0.0};
  int i = 0;
  while (d[i] != 0.0 && !ds.full())
    if (!ds.full()) ds.push(d[i++]);
  while (!ds.empty())
    cout << ds.pop() << "  ";
  cout << endl;
  return 0;
}
```

Execution:

```
% CC -o stack stack.cpp stacktest.cpp
stack.cpp:
stacktest.cpp:
% stack
mississippi
```

For 100% Result Oriented IGNOU Coaching and Project Training
Call CPD: 011-65164822, 08860352748

ppississim

7 5 3 1

(b) What is inheritance? What are different types of inheritance? Explain advantages of using inheritance. **(8 Marks)**

Ans 5 (b)

Inheritance is one of the basic concepts of OOP through which we can create new classes by deriving them from pre-existing classes and then the resulting classes inherit attributes and behavior from pre-existing classes called base classes, superclasses, or parent classes. The resulting classes are known as derived classes, subclasses, or child classes. The relationships of classes through inheritance gives rise to a hierarchy.

Types Of Inheritance

Single Inheritance: It is the inheritance hierarchy wherein one derived class inherits from one base class.

Multiple Inheritance:It is the inheritance hierarchy wherein one derived class inherits from multiple base class(es)

Hierarchical Inheritance: It is the inheritance hierarchy wherein multiple subclasses inherits from one base class.

Multilevel Inheritance: It is the inheritance hierarchy wherein subclass acts as a base class for other classes.

Hybrid Inheritance:The inheritance hierarchy that reflects any legal combination of other four types of inheritance.

Advantages of inheritance:

• It permits code reusability.
• Reusability saves time in program development.
• It encourages the reuse of proven and debugged high-quality software, thus reducing problem after a system becomes functional.