Answers to some of the selected questions

Q1.

	Procedure Oriented Programming	Object Oriented Programming
Programming Pattern	Code is divided into functions	Code is divide into small parts called objects
Focus/ Fundamental Unit	Functions	Data
Information Processing/ Knowledge ordering/ Approach/ Programming Style	Top-Down	Bottom-up
Data movement/ access	Data can move from function to function in the system to be read or written into.	Data imposes access restriction to non-member functions/ methods
Data security	NO	HIGH
Program goals	Action sequence	OO programs are collection of interdependent components/ objects, each providing a service specified by its interface.
Code reusability	Yes	Higher than POP
Data hiding	No	Yes
Polymorphism	No	Yes
Changes to system once implemented	Difficult, system will be unstable	Easy without altering or changing the entire system

Q3.

3 Principles of OOPs
Encapsulation, Inheritance, Polymorphism

- Encapsulation: Encapsulation is the mechanism that binds together code and the data it manipulates, and keeps both safe from outside interference and misuse.
- Inheritance: Inheritance is the process by which one object acquires the properties of another object. This is important because it supports the concept of hierarchical classifications.
- Polymorphism: Polymorphism is a feature that allows one interface to be used for a general class of actions. The concept of polymorphism is expressed by the phrase “one interface, multiple methods. Polymorphism allows to create clean, sensible, readable, and resilient code.

Q5
Program sample and outline.
Note: Students can illustrate with any example
Step 1:

```
/*Date.h*/
struct date
{
Int m;
Int d;
Int y;
};

Void next_day(struct date *);
Void get_sys_date(struct date *);
```

Step 2:

```
/*Date.c*/
#include “date.h”

Void next_day(Struct date *p){
//Calculate the date that immediately follow the one
//represented by *p and set it to *p
}
Void set_sys_date(struct date *p){
//determine the current system date and set it to *p
}
```

Step 3:

```
/*dateUser.c*/
#include”date.h”
Void main()
{
Struct date d;
d.d=28;
d.m=2;
d.y=1999;

if(condition is true)
d.m++;
}
```

In the above code the line d.m++ will cause run time error.
Detecting such piece of code in thousand lines of code is difficult.

Absence of a facility to bind the data and the code that can have the exclusive rights to manipulate the data, can lead to difficult-to-detect run time bugs.

Q6.
Console output program

```
#include<iostream.h>
Void main()
{
Cout<<"Welcome to C++ World";
}
```

- Cout is a standard stream object representing a monitor, it is an alias for console output. It is the object of class ostream_withassign.
- << is originally a left shift binary operator, is an insertion operator, it is overloaded to insert the character stream headed towards the monitor.
- In the above program the string "Welcome to C++ world" is inserted into standard output stream by the << operator.

3

Console input program

```
#include<iostream.h>
Void main()
{
Int x;
Cin>>x;
Cout<<x;
}
```

3

- Cin is a standard input object representing keyboard, is an alias for console input, is an object of class istream_with assign.
- >> is originally a right shift binary operator, is an extraction operator, it is overloaded to extract from the standard input stream, convert it to related type and assign it to a identifier.
- In the above program cin>>x will assign a value to the variable x.

Q8.
/*program to swap two numbers by using reference variable*/

```
#include<iostream.h>
Void swap(int &,int &);
Void main()
{
Int x,y;
Cout<<"Enter two integer numbers"<<endl;
Cin>>x>>y;
Cout<<"Before swapping"<<endl;
Cout<<"X= "<<x<<" Y=" <<y;
Swap(x,y);
```

6

```
Cout<<"After swapping"<<endl;
Cout<<"X= "<<x<<" Y=" <<y;
}
Void swap(int & a,int & b)
{
Int temp;
temp=a;
a=b;
b=temp;
}
```

Q9.

Function Overloading

- C++ allows two or more functions to have the same name.
- For this, however, they must have different signatures.
- Signature of a function means the number, type, and sequence of formal arguments of the function.
- In order to distinguish amongst the functions with the same name, the compiler expects their signatures to be different.
- Depending upon the type of parameters that are passed to the function call, the compiler decides which of the available definitions will be invoked.
- For this, function prototypes should be provided to the compiler for matching the function calls. Accordingly, the linker, during link time, links the function call with the correct function definition.

3 + 3 + 1 = 7

```
/*Beginning of funcOverload.cpp*/
 #include<iostream.h>
 int add(int,int); //first prototype
int add(int,int,int); //second prototype
void main()
{
int x,y;
x=add(10,20); //matches first prototype
 y=add(30,40,50); //matches second prototype
cout<<x<<endl<<y<<endl;
}
int add(int a,int b)
{
return(a+b);
 }
int add(int a,int b,int c)
 {
 return(a+b+c);
} /*End of funcOverload.cpp*/
```

Output

```
 30
```

- The compiler decides which function is to be called based upon the number, type, and sequence of parameters that are passed to the function call.
- Function overloading is possible because of the necessity to prototype functions.

Q10.
Inline function:
Inline functions are used to increase the speed of executable files. An inline function is a function whose compiled codes is 'in line' with the rest of the program. That is, the compiler replaces the function call with the corresponding function code. With inline codes, the program does not have to jump to another location to execute the code and then jump back to continue with execution.
Avoids Overhead like context switching between processes.
Code snippet:

```
inline Double cube(double n){return n*n*n;}
Void main()
{
Double x;
x=cube(2)
Cout<<x;
x=cube(3);
Cout<<x;

}
```

Will be code expanded by compiler as:

```
Void main()
{
Double x;
{
Double n;
n=2
x=n*n*n;
Cout<<x;
}
{
Double n;
n=3
x=n*n*n;
Cout<<x;
}

}
```

Conditions when inline expansions may not be considered:

- The function is recursive.
- There are looping constructs in the functions.

- There are static variables in the function.

Q11.
Scope resolution operator (::)
It is used to qualify hidden names so that they can be used. It is a unary operator.
e.g
```
int count=0;
void main()
{
Int count=0;
::count=1;//set global count to 1
Count=2; //set local count to 2
}
```
this pointer:
this pointer always points at the object with respect to which the function was called. It is always a constant pointer.
```
Class distance{
Int iFeet;
Float fInches;
Distance add(Distance d);
}
Distance Distance::add(Distance d)
{
Distance temp;
temp.iFeet=this.iFeet+d.iFeet; // this points to the invoking //object
temp.fInches=this.fInches+d.fInches; //this points to the //invoking object
return temp;
}
Void main()
{
Distance d1,d2,d3;
d1=d2.add(d3);
}
```
2 + 2 + 2 = 6

Arrow operator:
This is called pointer-to-member access operator. It is a binary operator. It is used to access member function and data member of a class.
```
Class X
{
Public:
Int x;
Void disp();
};
Void X::disp()
{
Cout<<"Hello "<<x<<endl;
```

```
}

Void main()
{
X x;
X *ptr;
Ptr=&x;
Ptr->x=10;
Ptr->disp();
}
```

Q14.

- Class: Class is a user defined data type. It is a blueprint of an object. It is a template of an object. Class defines the state and behavior of an object.
- Object: Object is an instance of a class. Object is an variable of a class.
- Mutable data member: Is member of class which is never constant. It is modifiable inside constant functions also.
- Friend functions: A friend function is a non-member function that has special rights to access private data members of any object of the class of whom it is a friend.
- Friend Classes: A class can be friend of another class. Member functions of a friend class can access private data members of objects of the class of which it is a friend.
- Namespaces: Namespaces enables the C++ programmer to prevent pollution of the global namespace. They help prevent name clashes.
- Nested Classes: One class can be defined inside another class. Such a class is known as a nested class. The class that contains the nested class is known as the enclosing class.

1 Mark each (1 X 7 = 7)

Q15.

```
/*Program to calculate the area of a circle, triangle and rectangle using method overloading*/
#include<math.h>
Class Figure
{
double area(double radius)
{
 Return 3.14*radius*radius;
}
Double area(double a, double b, double c)
{
Double p=(a+b+c)/2;
Return(sqrt(p*(p-a)*(p-b)*(p-c)));
```

6

```
}
Double area(double length, double width)
{
Return (length,width);
}
};
Void main()
{
Figure circle,rectangle,triangle;
Cout<<"Area of circle "<<circle.area(9)<<endl;
Cout<<"Area of triangle "<<triangle.area(3,4,2.5)<<endl;
Cout<<"Area of rectangle "<<rectangle.area(9,9)<<endl;


}
```

Q16.

Constructor: A very special function with the same name as the class and no return type is used to initialize the data members of the object.

Code snippet for constructor

```
Class X
{
Int a;
Float b;
X();
X(float, float);
};
X::X(){a=0;b=0;} //zero parameter constructor
X::X(float a, float b){this.a=a; this.b=b;}
```

Destructor: The destructor gets called for each object that is about to go out of scope. The destructor guarantees deinitialization of member data of a class and frees up the resources acquired by the object during its lifetime.

2 + 2 + 2 = 6

Code snippet for destructor

```
Class X
{
Public:
Int a;
Float b;
X( );
X(float, float);
~X( )
};
X::X(){a=0;b=0;} //zero parameter constructor
X::X(float a, float b){this.a=a; this.b=b;}
X::~X(){cout<<"destructor called";}
```

Copy constructor: The copy constructor is a special type of parameterized constructor. It copies one object to another.

Code snippet for copy constructor

```
Class X
{
Public:
Int a;
Float b;
X( );
X(float, float);
~X( )
};
X::X(){a=0;b=0;} //zero parameter constructor
X::X(float a, float b){this.a=a; this.b=b;}
X::~X(){cout<<"destructor called";}
Void main()
{
X x;
X x1=x; // copy constructor called
}
```

Q17.

- Static member functions can only access static data members only. They can be called without declaring any objects. A member function is declared as a static member of a class by prefixing its declaration in the class by the keyword static.
- Static member functions cannot access non static member functions.
- Static member functions can invoke objects.

Program to show static data members and functions

```
Class Account                                   4
{                                               +
Static float interest_rate;                     3
Public:                                         =
Static void set_interest_rate(float);           7
};
Float Account::interest_rate=9.99;
Void Account::set_interest_rate(float p)
{
Interest_rate=p;
}
Void main()
{
```

Account::set_interest(8.75);
}
Q18.
Java Buzz Words:

- Simple

Java has a concise, cohesive set of features that makes it easy to learn and use.

- Secure

Java provides a secure means of creating Internet applications.

- Portable

Java programs can execute in any environment for which there is a Java run-time system.

- Object-oriented

Java embodies the modern, object-oriented programming philosophy.

- Robust

Java encourages error-free programming by being strictly typed and performing run-time checks.

- Multithreaded 7

Java provides integrated support for multithreaded programming.

- Architecture-neutral

Java is not tied to a specific machine or operating system architecture.

- Interpreted

Java supports cross-platform code through the use of Java bytecode.

- High performance

The Java bytecode is highly optimized for speed of execution.

- Distributed

Java was designed with the distributed environment of the Internet in mind.

- Dynamic

Java programs carry with them substantial amounts of run-time type information that is used to verify and resolve accesses to objects at run time

Q20.

Category	Types	Size (bits)	Minimum Value	Maximum Value	Precision	Example
	byte	8	-128	127	From +127 to -128	byte b = 65;
	char	16	0	2^{16}-1	All Unicode characters	char c = 'A'; char c = 65;
Integer	short	16	-2^{15}	2^{15}-1	From +32,767 to -32,768	short s = 65;
	int	32	-2^{31}	2^{31}-1	From +2,147,483,647 to -2,147,483,648	int i = 65;
	long	64	-2^{63}	2^{63}-1	From +9,223,372,036,854,775,807 to -9,223,372,036,854,775,808	long l = 65L;
Floating-point	float	32	2^{-149}	$(2-2^{-23})\cdot 2^{127}$	From 3.402,823,5 E+38 to 1.4 E-45	float f = 65f;

6

	double	64	2^{-1074}	$(2-2^{-52})\cdot 2^{1023}$	From 1.797,693,134,862,315,7 E+308 to 4.9 E-324	double d = 65.55;
Other	boolean	1	--	--	false, true	boolean b = true;
	void	--	--	--	--	--

Q22.

```
//Labeled break
Class Break
{
Public static void main(String args[])
{
Boolean t=true;
First:{
Second:{
Third:{
System.out.println(“Before the break\n”);
If(t) break second;
System.out.println(“This won’t execute”);
}
System.out.println(“This won’t execute”);
}
System.out.println(“This is after second block”);
}
```

4 + 3 = 7

```
}
}
//labeled continue
Class ContinueDemo
{
Public static void main(String args[])
{
Outer: for(int i=0;i<10;i++)
{
For(int j=0;j<10;j++)
{
If(j>i)
{
System.out.println();
Continue outer;
}
System.out.println(“ “+(i*j));

}
}
System.out.println();
}
```

}
Q24.
Unsigned right shift operator (>>>):]
automatically fills the higher order bit with zeroes. This situation is common when working with pixel-based values and graphics.

Code snippet

Int a=-1;
A=a>>>24;

The same operation in binary format
11111111 11111111 11111111 11111111 -1 in binary as in int
>>>24
00000000 00000000 00000000 11111111 255 in binary as an int

3 + 3 = 6

Foreach in Java:

- Java doesn't have foreach keyword.
- Java adds the the for-each capability by enhancing the for statement.
- The general form:

for(type itr-var: collection) statement-block

code snippet

int nums[]={1,2,3,4,5,6,7,8,9};
int sum=0;

for(int x:nums)
{
System.out.println(x);
sum+=x;
}
System.out.println("Sum="+x);