**Q- What are the major issues facing the software industry today?**

A- Developments in software technology continue to be dynamic. New tools and techniques are announced in quick succession. This has forced the software engineers and industry to continuously look for new approaches to software design and development, and they are becoming more and more critical in view of the increasing complexity of software systems as well as the highly competitive nature of the industry. These rapid advances appear to have created a situation of crisis within the industry. The following issued need to be addressed to face the crisis:

• How to represent real-life entities of problems in system design?

• How to design system with open interfaces?

• How to ensure reusability and extensibility of modules?

• How to develop modules that are tolerant of any changes in future?

• How to improve software productivity and decrease software cost?

• How to improve the quality of software?

• How to manage time schedules?

**Q- Describe how data are shared by functions in procedure-oriented programs?**

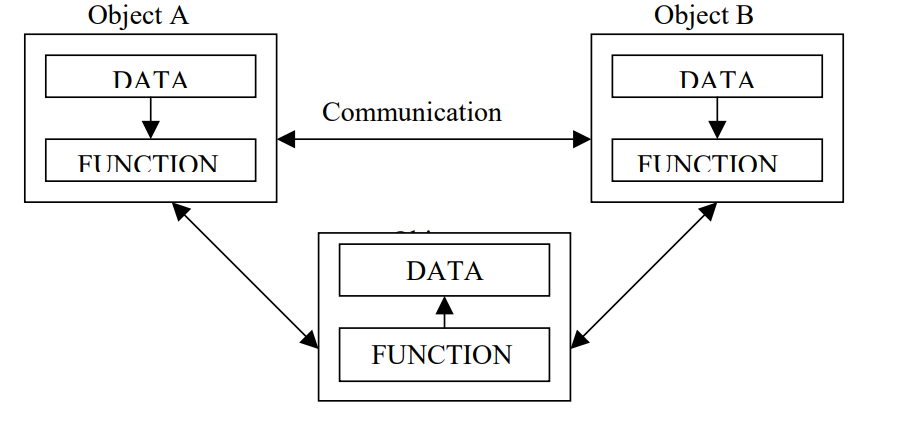
A- Procedure oriented programming basically consists of writing a list of instructions for the computer to follow, and organizing these instructions into groups known as functions. We normally use flowcharts to organize these actions and represent the flow of control from one action to another. In a multi-function program, many important data items are placed as global so that they may be accessed by all the functions. Each function may have its own local data. Global data are more vulnerable to an inadvertent change by a function. In a large program it is very difficult to identify what data is used by which function. In case we need to revise an external data structure, we also need to revise all functions that access the data. This provides an opportunity for bugs to creep in.

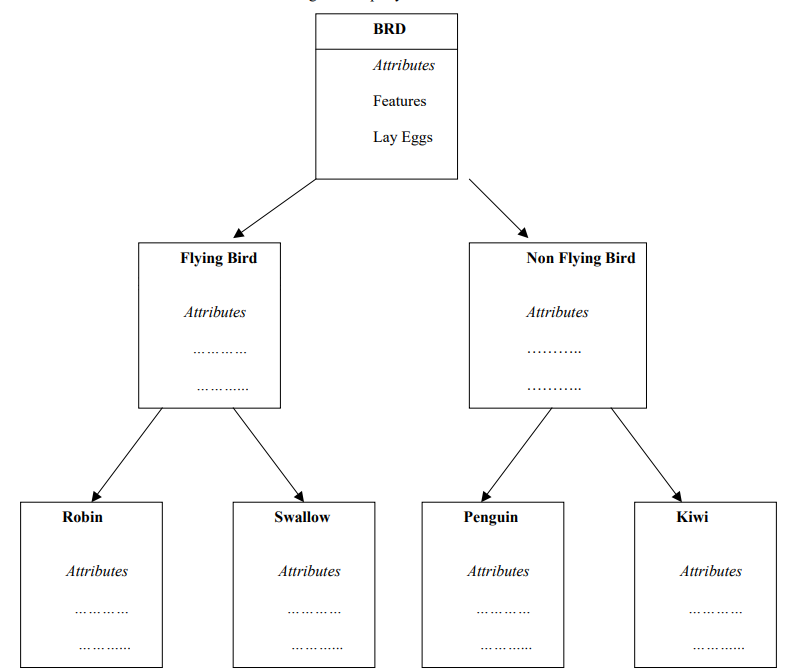
**Q- What are the difference between POP and OOP?**

| **Sr. No.** | **Key** | **OOP** | **POP** |
| --- | --- | --- | --- |
| 1 | Definition | OOP stands for Object Oriented Programing. | POP stands for Procedural Oriented Programming. |
| 2 | Approach | OOP follows bottom up approach. | POP follows top down approach. |
| 3 | Division | A program is divided to objects and their interactions. | A program is divided into funtions and they interacts. |
| 4 | Inheritance supported | Inheritance is supported. | Inheritance is not supported. |
| 5 | Access control | Access control is supported via access modifiers. | No access modifiers are supported. |
| 6 | Data Hiding | Encapsulation is used to hide data. | No data hiding present. Data is globally accessible. |
| 7 | Example | C++, Java | C, Pascal |

A-

**Q- How are data and functions organized in an object-oriented program?**

A- OOP treats data as a critical element in the program development and does not allow it to flow freely around the system. It ties data more closely to the function that operate on it, and protects it from accidental modification from outside function. OOP allows decomposition of a problem into a number of entities called objects and then builds data and function around these objects. The data of an object can be accessed only by the function associated with that object. However, function of one object can access the function of other objects.



**Q- Describe inheritance as applied to OOP.**

A- Inheritance is the process by which objects of one class acquired the properties of objects of another classes. It supports the concept of hierarchical classification. For example, the bird, ‘robin’ is a part of class ‘flying bird’ which is again a part of the class ‘bird’. In OOP, the concept of inheritance provides the idea of reusability. This means that we can add additional features to an existing class without modifying it. This is possible by deriving a new class from the existing one. The new class will have the combined feature of both the classes. The real appeal and power of the inheritance mechanism is that it

**Q- What is the use of pre-processor directive #include?**

A- The header iostream contains the functions required for input/output data streaming like cout, cin, etc. As our programs grow larger or the functionality becomes complex, we might want to divide our program into various files or import functionality from the other files. In this case, we make use of user-defined files. To include user-defined files in our program we can make use of the following syntax of #include directive.

**Q- How does a main () function in c++ differ from main () in c?**

In *C* **main()**function *can be****called through other functions.***

while in*C++***main()** function ***cannot be called through other functions.***

1. In C++ you should always use **int main()**.
2. Using **void main()**results in an error for all compilers except ancient boreland compilers, and is not standard.

* It's also worth noting that in**C++**, int main() can be**left without a return value** at which point it *defaults to returning 0*. This is also true with a C99 program. Whether return 0 should be omitted or not is open to debate. The range of valid C program main signatures is much greater.
* Also, efficiency is not an issue with the main function. It can only be entered and left once (marking program start and termination) according to the C++ standard. For C, the case is different and re-entering main() is allowed, but should probably be avoided.

**Q- What do you mean by dynamic binding? How it is useful in OOP?**

A- Binding refers to the linking of a procedure call to the code to be executed in response to the call. Dynamic binding means that the code associated with a given procedure call is not known until the time of the call at run time. It is associated with polymorphism and inheritance. A function call associated with a polymorphic reference depends on the dynamic type of that reference. Its algorithm is, however, unique to each object and so the draw procedure will be redefined in each class that defines the object. At run-time, the code matching the object under current reference will be called.

**Q- Distinguish between the following terms:**

**(a) Object and classes (b) Data abstraction and data encapsulation (c) Inheritance and polymorphism (d) Dynamic binding and message passing**

**Q – What are the key concepts of OOP?**

A – Objects and Classes

**Q-** **Why OOP?**

A - Object-oriented programming was developed because limitations were discovered in earlier approaches to programming. C, Pascal, FORTRAN, and similar languages are procedural languages. That is, each statement in the language tells the computer to do something: Get some input, add these numbers, divide by six, display that output. A program in a procedural language is a list of instructions.

For very small programs, no other organizing principle (often called a paradigm) is needed. The programmer creates the list of instructions, and the computer carries them out.

As programs grow ever larger and more complex, even the structured programming approach begins to show signs of strain. You may have heard about, or been involved in, horror stories of program development. The project is too complex, the schedule slips, more programmers are added, complexity increases, costs skyrocket, the schedule slips further, and disaster ensues.

In a large program, there are many functions and many global data items. The problem with the procedural paradigm is that this leads to an even larger number of potential connections between functions and data

**Q – What is the fundamental idea behind OOP?**

A- The fundamental idea behind object-oriented languages is to combine into a single unit both data and the functions that operate on that data. Such a unit is called an object.

An object’s functions, called member functions in C++, typically provide the only way to access its data. If you want to read a data item in an object, you call a member function in the object. It will access the data and return the value to you. You can’t access the data directly. The data is hidden, so it is safe from accidental alteration. Data and its functions are said to be encapsulated into a single entity. Data encapsulation and data hiding are key terms in the description of object-oriented languages.’

**Q – Why use comments?**

A - Use comments to explain to the person looking at the listing what you’re trying to do. The details are in the program statements themselves, so the comments should concentrate on the big picture, clarifying your reasons for using a certain statement or group of statements.

**Q – What are identifiers?**

A - The names given to variables (and other program features) are called identifiers. You can use upper- and lowercase letters, and the digits from 1 to 9. You can also use the underscore (\_). The first character must be a letter or underscore. Identifiers can be as long as you like, but most compilers will only recognize the first few hundred characters. The compiler distinguishes between upper- and lowercase letters, so Var is not the same as var or VAR. You can’t use a C++ keyword as a variable name. A keyword is a predefined word with a special meaning. int, class, if, and while are examples of keywords.

**Q – What are integer constants?**

A - The number 20 is an integer constant. Constants don’t change during the course of the program. An integer constant consists of numerical digits. There must be no decimal point in an integer constant, and it must lie within the range of integers.

**Q – What is Cascading >> ?**

**Q – What are expressions?**

A - Any arrangement of variables, constants, and operators that specifies a computation is called an expression. Thus, alpha+12 and (alpha-37)\*beta/2 are expressions. When the computations specified in the expression are performed, the result is usually a value. Thus if alpha is 7, the first expression shown has the value 19. Parts of expressions may also be expressions. In the second example, alpha-37 and beta/2 are expressions. Even single variables and constants, like alpha and 37, are considered to be expressions. Note that expressions aren’t the same as statements. Statements tell the compiler to do something and terminate with a semicolon, while expressions specify a computation. There can be several expressions in a statement.

**Q – Difference between double and long double?**

A - The larger floating point types, double and long double, are similar to float except that they require more memory space and provide a wider range of values and more precision. Type double requires 8 bytes of storage and handles numbers in the range from 1.7x10–308 to 1.7x10308 with a precision of 15 digits. Type long double is compiler-dependent but is often the same as double.

**Q – What is a const Qualifier?**

A - Besides demonstrating variables of type float, the CIRCAREA example also introduces the qualifier const. It’s used in the statement const float PI = 3.14159F; //type const float The keyword const (for constant) precedes the data type of a variable. It specifies that the value of a variable will not change throughout the program. Any attempt to alter the value of a variable defined with this qualifier will elicit an error message from the compiler. The qualifier const ensures that your program does not inadvertently alter a variable that you intended to be a constant, such as the value of PI in CIRCAREA. It also reminds anyone reading the listing that the variable is not intended to change. The const modifier can apply to other entities besides simple variables

**Q – What are casts?**

**Q- What are arithmetic Assignment Operators?**

A - There are arithmetic assignment operators corresponding to all the arithmetic operations: +=, -=, \*=, /=, and %= (and some other operators as well)

**Q – What are arguments?**

A - An argument is the input to the function; it is placed inside the parentheses following the function name. The function then processes the argument and returns a value; this is the output from the function.

**Q – What are the differences between header files and library files?**

**Q - Write a statement that displays the variable Arjun in a field ten characters wide.**

A - Cout<<setw(10)<<ARJUN;

**Q – What are Header Files used for?**

A - Header files are used for declaration, data for library, objects and over loaded operators.

**Q – Throw some light on “default” keyword.**

**Q – What are member functions?**

**Q – What are whitespaces?**

**Q – What are header Files?**

**Q – What is “using” directive used for?**

**Q – What are identifiers?**