**Object oriented programming**

**Introduction to Object oriented programming**

**Introduction**

Object Oriented Programming (OOP) was invented to overcome some flaws associated with procedural-oriented programming. OOP treats data as critical element in the program development and does not allow it to flow freely around the system. It ties data more closely to the functions that operate on it, and protects it from accidental modifications from outside functions. It allows decomposition of a problem into a number of entities called objects and then built data and functions around these objects. The data of an object can be accessed only by functions associated with that object, but functions of one object can access the functions of other objects.



Figure 1: *Organization of data and functions in OOP*

**Features of OOP**

* Emphasis on data rather than procedures
* Programs are divided into objects
* Data structures are designed such that they characterize the objects
* Functions that operate on the data of an object are tied together in the data structure
* Data is hidden and cannot be accessed by external functions
* Objects may communicate with each other through functions
* New data and functions can be easily added through functions
* Follows bottom-up approach design

Therefore OOP 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*. Thus an object is considered to be partitioned area of computer memory that stores data and set operations that can access that data. Since the memory partitions are independent, the objects can be used in a variety of different programs without modifications.

*OBJECT-ORIENTATION 1 is a set of tools and methods that enable software engineers to build reliable, user friendly, maintainable, well documented, reusable software systems that fulfills the requirements of its users. It is claimed that object-orientation provides software developers with new mind tools to use in solving a wide variety of problems. Object-orientation provides a new view of computation. A software system is seen as a community of objects that cooperate with with each other by passing*

*messages in solving a problem. An object-oriented programming laguage provides support for the following objectoriented concepts:*

* *Objects and Classes*
* *Inheritance*
* *Polymophism and Dynamic binding*

**Evolution of OOP**

The use of object-oriented technology is not restricted to any particular language; rather, it is applicable to a wide spectrum of object-based and object-oriented programming languages. As important as analysis and design are, however, we cannot ignore the details of coding, for ultimately our software architectures must be expressed in some programming language. Indeed, as Wulf has suggested, a programming language serves three purposes:

* It is a design tool
* It is a vehicle for human consumption
* It is a vehicle for instructing a computer

Herein provided is a summary description of a number of the more important languages, together with a common example that provides a basis for comparing the syntax, semantics, and idioms of two of the more interesting object-oriented programming languages, namely C++ and Smalltalk.

**Concepts**

Currently, there are over 2,000 different high-order programming languages. We see so many different languages because each was shaped by the particular requirements of its perceived problem domain. Furthermore, the existence of each new language enabled developers to move on to more and more complex problems.

Programming languages may be grouped into four generations, according to whether they support ***mathematic***, ***algorithmic***, ***data***, or ***object-oriented*** abstractions. The most recent advances in programming languages have been due to the influence of the object model. There are currently over 100 different object-based and object-oriented programming languages today. A language is considered object-based if it directly supports data ***abstraction*** and ***classes***. An object-oriented language is one that is object-based, but also provides support for ***inheritance*** and ***polymorphism***.

The common ancestor of almost every contemporary object-based and object-oriented programming language is Simula, developed in the 1960s. Simula built upon the ideas of ALGOL, but added the concepts of ***encapsulation*** and ***inheritance***. Figure 2 shows the genealogy of the most influential and widely used object-based and object-oriented programming languages. We examine several of these languages relative to the support they offer to the elements of the object model.



Figure 2: *A Genealogy of Object-Based and Object-Oriented Programming Languages*

1. **Smalltalk**

Created by the members of the Xerox Palo Alto Research Center Learning Research Group as the software element of the Dynabook, a visionary project of Alan Kay.

Simula was its primary influence, although Smalltalk also took some ideas from the language FLEX. It represents both a language and a software development environment. It is a pure object-oriented programming language, in that everything is viewed as object even integers are classes. It is perhaps the most important object-oriented programming language, because its concepts have influenced not only the design of almost every subsequent object-oriented programming language, but also the look and feel of graphic user interfaces such as the Macintosh user interface, Windows, and Motif, all of which are now largely taken for granted.

1. **Object Pascal**

Object Pascal was created by developers from Apple Computer (some of whom were involved in the development of Smalltalk), Object Pascal’s immediate ancestor is Clascal, an object-oriented version of Pascal for the Lisa. Object Pascal was made publicly available in 1986 and is the first object-oriented programming language supported by the Macintosh Programmer's Workshop (MPW), the development environment for Apple's family of Macintosh computers. The class library for MPW, called MacApp, provides the frameworks for constructing applications that conform to the Macintosh user interface guidelines.

1. **C++**

C++ was designed by Bjarne Stroustrup of AT&T Bell Laboratories. The immediate ancestor of C++ is a language called C with classes, also developed by Stroustrup in 1980. In turn, C with Classes was heavily influenced by the languages C and Simula. C++ is largely a superset of C. However, in one sense, C++is simply a better C, in that it provides type checking, overloaded functions, and many other improvements. Most importantly, however, C++ adds object-oriented programming features to C.

1. **Common Lisp Object System**

There are literally dozens of dialects of Lisp, including MacLisp, Standard Lisp, SpiceLisp, etc. Starting in the early 1980's, a plethora of new dialects of Lisp emerged that supported object-oriented programming, many of which were invented to support ongoing research in knowledge representation. Spurred by the success in standardizing Common Lisp, a similar effort was undertaken in 1986 to standardize these object-oriented dialects.

1. **Ada**

The United States Department of Defense (DoD) is perhaps the largest user of computers in the world. By the mid-1970s, software development for its systems had reached crisis proportions: projects were often late, over budget, and they often failed to meet their stated requirements. it was evident that the problems would only worsen as software development costs continued to rise and the demand for software increased at an exponential rate. To help resolve these problems, which were further compounded by the proliferation of hundreds of different languages, the DoD) sponsored the development of a single, common, high-order programming language. In a sense, Ada represents one of the first engineered production quality languages. A set of requirements was developed starting in 1975 and culminated in the Steelman document, which was released in 1978. An international request for proposal (RFP) was then issued, inviting companies to design a language based upon these requirements. The RFP drew seventeen responses. This number was reduced to four, then two, and then one by an extensive design and evaluation period involving hundreds of computer scientists throughout the world. The winning design was originally called the Green language (so called because of its color code during the competition), and was then renamed Ada, in honor of *Ada Augusta, Countess of Lovelace*, who was noted for her early observations on the potential power of the computer. According to its designers, Ada was designed with three concerns in mind:

* Program reliability and maintenance
* Programming as a human activity
* Efficiency

1. **Eiffel**

Eiffel was created by Bertrand Meyer not only as an object-oriented programming language, but also as a software engineering tool. While Eiffel is influenced by Simula, it was designed from the beginning to be an independent object-oriented language and development environment.

**Programming paradigms/Techniques**

With the development of programming languages the different programming techniques also evolved. The distinguishable techniques are listed below.

1. Unstructured or monolithic programming.
2. Procedural Programming
3. Modular Programming
4. Object oriented Programming

**Unstructured/monolithic programming**

The general structure of a monolithic program consists of global data and statements which modify the data and finally it contains the output statements. A sample is shown below

// Main program

Data

Statement1

Statement2

Statement

——————

Statement1

Statement2

end

The program is executed from top to bottom, statement by statement. If similar evaluations are to be carried out at several places in the program, for example, statement1 and statement2 in above illustration, all the statements concerning that evaluation have to be repeated at all the places where the evaluation is desired. This makes the program very lengthy. Besides, any modification in the process of the evaluation has to be corrected at so many places in the program. Such programs are lengthy, difficult to debug and difficult to maintain.

**Procedural programming**

These programs are an improvement over the monolithic programs. If a group of statements carry out similar action at several places in the program, such a group is taken out of the main program and is placed in a subprogram called subroutine or procedure or function. In the main program the subroutines or functions are called. When a subroutine is called, the main program is paused and control shifts to the subroutine till it is finished. Its return value if any is fed to the main program which is resumed from where it was left. It is illustrated in Figure 3.



Figure 3: Procedural programming

A procedural program is more structured. The different procedures or subroutines can be checked independently of the main program, thus debugging becomes easier. It is also easier to maintain the program. The drawback is that if the subroutine is small and is called many times in the program, the overload of calling makes the program inefficient.

**Modular programming**

In this type of programming the similar procedures are grouped together into modules. Thus the whole program may consist of modules and connecting statements. The main program supplies data and co-ordinates the different modules. It is illustrated in Figure 4.



Figure 4: Modular programming

**Object oriented programming**

In this technique an important factor is the data abstraction. The different procedures or functions are built around the abstract data. Thus the data and functions are encapsulated in a single structure called ‘class’. The classes are the bases of object oriented programming (OOP). The classes create new data types besides the fundamental types which are already defined in C++. In a class, the different functions/procedures are defined according to the data that an object of the class must have. In the following the concept of classes and objects are explained with analogies from the physical world.

**Advantages and disadvantages of OOP**

**Advantages of OOP**

* **Reduced Maintenance:** The primary goal of object-oriented development is the assurance that the system will enjoy a longer life while having far smaller maintenance costs. Because most of the processes within the system are encapsulated, the behaviors may be reused and incorporated into new behaviors.
* **Real-World Modeling:** Object-oriented systems tend to model the real world in a more complete fashion than do traditional methods. Objects are organized into classes of objects, and objects are associated with behaviors. The model is based on objects, rather than on data and processing.
* **Improved Reliability and Flexibility:** Object-oriented system promise to be far more reliable than traditional systems, primarily because new behaviors can be "built" from existing objects. Because objects can be dynamically called and accessed, new objects may be created at any time. The new objects may inherit data attributes from one, or many other objects. Behaviors may be inherited from super-classes, and novel behaviors may be added without effecting existing systems functions.
* **High Code Reusability:** When a new object is created, it will automatically inherit the data attributes and characteristics of the class from which it was spawned. The new object will also inherit the data and behaviors from all superclasses in which it participates. When a user creates a new type of a widget, the new object behaves "wigitty", while having new behaviors which are defined to the system.

**Disadvantages of OOP**

* **Object-oriented Development is not a panacea** - Object-oriented Development is best suited for dynamic, interactive environments, as evidenced by its widespread acceptance in CAD/CAM and engineering design systems. Wide-scale object-oriented corporate systems are still unproved, and many bread-and-butter information systems applications (i.e. payroll, accounting), may not benefit from the object-oriented approach.
* **Object-oriented Development is not a technology** - Although many advocates are religious in their fervor for object-oriented systems, remember that all the "HOOPLA" is directed at the object-oriented approach to problem solving, and not to any specific technology.

**Object oriented programming Concepts**

**Concepts associated with OOP**

What does it mean to be object-oriented? The concepts used extensively in OOP include:

1. **Classes -** The entire set of data and code of an object can be made a user-defined data type with the help of a class. In C++ a class is a sort of blueprint or design in programming that describes the characteristics of a group of objects which share some common characters. A class comprises data members and function members. Some of the function members are defined with data members as parameters while others may serve as an interface for other functions declared private in the class, still others may be friend functions, etc. When the class is implemented, the functions members operate on the actual data of an object to bring out characteristics of the object. The actual object data is substituted for the data members in the functions contained in the class. Precisely objects are variables of type class. Once a class has been defined, we can create any number of objects belonging to that class. Each object is associated with the data of type class with which they are created. A class is thus a collection of objects of similar type e.g. mango, apple and orange are members of class fruit. Classes are user-defined data types and behave like the built-in types of a programming language. Therefore a class is a template for an object describing attributes and methods that will exist for all instances of the class.
2. **Objects** - They are the basic run-time entities in an object-oriented system that may represent a person, a place, a bank account, a table of data or any item that the program has to handle. Objects may also represent user-defined data such as vectors, time and lists. They should be chosen such that they match closely with real-world objects. They take up space in memory and have an associated address like record in Pascal, or a structure in C. When a program is executed the objects interact by sending messages to one another e.g. customer & account. Each object contains data, and code to manipulate the data. Objects can interact without having to know details of each other’s data or code. It is sufficient to know the type of message accepted and the type of response returned by the objects.



Figure 5: Two ways of representing an object

1. **Data abstraction and encapsulation** – The wrapping up of data and functions into a single unit called class is known as encapsulation. Data encapsulation is the most striking feature of a class. The data is not accessible to the outside world and only those functions which are wrapped in the class can access it. These functions provide the interface between the object’s data and the program. This insulation of the data from direct access by program is called **data hiding** or **information hiding**. Abstraction refers to the fact of representing essential features without including the background details or explanations. Classes use the concept of abstraction and are defined as a list of abstract attributes such as size, weight and cost and functions to operate on these attributes. They encapsulate all essential properties of the objects that are to be created. The attributes are sometimes called ***data members*** because they hold information. The functions that operate on these data are sometimes called ***methods*** or ***member functions***. Implementation details are hidden from the outside world e.g. we all know how to use a phone, few of us care how it works. The packaging of operations and attributes representing state into an object type so that state is accessible or modifiable only through the objects' interface. Encapsulation lets builders of objects reuse already-existing objects, and if those objects have already been well-tested, much larger and more complex systems can be created.
2. **Inheritance** – It is the process by which objects of one class acquire the properties of objects of another class. In OOP inheritance provides the idea of reusability i.e. we can add additional features to an existing class without modifying it. It is made possible by deriving a new class from existing class; the new class will have combined features of both the classes. Inheritance allows the programmer to reuse a class that is almost but not exactly. A subclass is derived from a superclass e.g. an employee is a person. The subclass inherits the attributes and behavior of the superclass. The subclass can override the behavior of the superclass. Notice the use of the ``Is-A'' phrase to describe inheritance.
3. **Polymorphism** – It is a greek word that means the ability to take more than one form i.e. literally means “many forms”. An operation may exhibit different behaviours in different instances, and the behaviours depend on the types of data used in the operation. It plays an important role in allowing objects having different internal structures to share the same external interface i.e. a general class of operations may be accessed in the same manner even though specific actions associated with each operation may differ. It is used extensively in implementing inheritance. A method can have many different forms of behavior. Commonly used between a set of classes that have a common superclass. The sender of a message does not have to know the type/class of the receiver. A single operation or attribute may be defined upon more than one class and may take on different implementations in each of those classes. An attribute may point to different objects at different times.
4. **Message passing:** An OO program consists of a set of objects that communicate with each other.The ability to send messages from one object to another.Objects communicate by sending messages.Messages convey some form of information.An object requests another object to carry out an activity by sending it a message.Most messages pass arguments back and forth.Meilir Page-Jones defines three types of messages:

* Informative - send information for the object to update itself.
* Interrogative - ask an object to reveal some information about itself
* Imperative - take some action on itself, or another object

Grady Booch defines four types of messages:

* Synchronous - receiving object starts only when it receives a message from a sender, and it is ready.
* Balking - sending object gives up on the message if the receiving object is not ready to accept it.
* Timeout - sending object waits only for a certain time period for the receiving object to be ready to accept the message.
* Asynchronous - sender can send a message to a receiver regardless of whether the receiver is ready to receive it.

1. **Dynamic binding -** Binding refers to the linking of a procedure call to the code to be executed in responses to the call. Dynamic binding also called late 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.

**Benefits of OOP**

* Through inheritance, we can eliminate redundant code and extend the use of existing classes.
* We can build programs from the standard working modules that communicate with one another, rather than having to start writing the code from scratch.
* The principle of data hiding helps the programmer to build secure programs that cannot be invaded by code in other parts of the program.
* It is possible to have multiple instances of an object to co-exist without any interference.
* It is easy to partition the work in a project based on objects.
* Data centered design approach enables us to capture more details of a model in implementable form.
* Software complexity can be easily managed.

**Application of OOP**

The promising areas for application of OOP include:

* Real-time systems
* Simulation and modeling
* OO databases
* Hypertext, hypermedia and expertext
* AI and expert systems
* Neural networks and parallel programming
* Decision support and office automation systems

**Language structure of Object oriented programming**

**Introduction to C++**

A typical program in C++ may comprise a list of statements involving variables (objects whose values may change during the execution of program), constants (whose values do not change), operators like +, –, etc. and functions, etc. Computer recognises them by their names just like you are recognised by your name. Obviously, no two variables or constants should have same name. However, functions may have same name if their parameters are different. It must be emphasised here that **C++ is a case sensitive language**, which, means that it will take ‘A’ and ‘a’ as two different objects. Similarly Area, area and AREA are three different objects. Therefore, while writing a program the name of a variable, constant or a function should be written consistently in the same fashion throughout the program.

Like any other high level programming language, C++ also has certain rules, special syntax and keywords to help the programmers to write a program and execute it. Keywords have special meanings for the compiler and are used to control and execute the program. Naturally, these words should be used in a program only for the purpose they are meant for. Their use as names of variables, constants, functions or objects will create errors in the program. Besides keywords, there are a number of files called **header files** and functions in **C++ Standard Library** which help in the execution of the programs. Therefore, every program has to include the header files which are required by the program.

The keywords as well as names of files and functions in C++ Standard Library are in general defined in lower case. As already mentioned the keywords should not be used as names, however, if the case of any letter in a keyword is changed it is no longer a keyword. Thus a safe bet is that the starting letter in a name may be made capital.

Also C++ is highly typed language which means that the data is categorized into different **types**. For example, whole numbers form a category called integers. So when a variable whose value can only be in whole numbers, is declared we write **int** (short form of integer) before its name. The int is its **type**. Variables which have values in floating decimal point numbers such as 2.4 or 3.14159, etc. form another category. For declaration of such variables we write **float** or **double** before their names. For variables which have values in form of characters are of type **char.**

**Assignment: 20 marks**

1. Discuss how characters are manipulated in C/C++ programming.
2. Write short notes on destructors
3. Analyse the features which are found in C and not in C++, and those in C++ but not in C.