10

Programming Techniques

A program is a set of instructions to control data manipulation in a computer. The process of writing instructions for a computer is called programming, and the persons who write the program are called programmers. A program is otherwise called software, and is considered as the lifeblood of computers. Program development requires thorough understanding of the problem to be solved and the program logic. The programming logic is the order in which the computer execute the statements in a program. It is the sequence of operations is to be performed to arrive at a solution.

A good program must be able to perform the assigned task with in the specified conditions. It must be consistent, easy to use and understand, must follow a logical flow and consist of program documents like specifications, and program design methods. The meaning of different variables and a detailed` description of the working of the program must also be given. Various stages in program development include understanding of user requirements, planning of the system functions, software requirements etc, development of operational system, and maintenance of software. During program development, it is better to break a large problem into a series of smaller and more understandable tasks. In order to solve a problem, the programmer first develops a main program specifying the order in which each substitute module in the program will be processed. All these modules are integrated with the control (main) program, and each module is invoked to perform the assigned task as and when required.

Current Trends in Programming Techniques

In today's society, where evoluation is swift, programming has changed profoundly in the past decade. Theoretical improvements have updated the definition of programming notations. This facilitates the understanding of concurrent computations, making synchronization explicit, and simplifying formal correctness and proofs. In addition, the wide spread ability to attain inexpensive processors has made it possible to construct distributed systems and multiprocessors once unachievable. As a result of these developments, concurrent programming is no longer the sole province of those who design and implement operating systems, it has become important to programmers of all kinds of applications.

As software becomes more and more complex, it is more and more important to structure it well. Well-structured software is easy to write and debug, and provides a collection of modules that can be re-used to reduce future programming costs. Conventional programming languages place conceptual limits on the way problems can be modularized. Functional languages push those limits back. Conventional programming approaches like procedural languages, structured languages, etc. tell the computer to do something. A programmer in BASIC, PASCAL, or C, creates a list of instructions and asks the computer to perform it. But, when problems become more and more complex, it is difficult to manage the list of instructions, which requires breaking up of programs into small manageable functions, When programs ever grow larger, even the structured programming approach begins to show signs of strains. Object-oriented programming languages were designed and developed, to resolve the limitations of these traditional programming approaches.

In object-oriented languages, both data and functions that operate on the data are combined to form a single unit, called an 'object. It is a way of organizing programs, in a company, various departments like sales, accounting, personnel department etc. constitute the objects, and provides an approach to develop co-operation among these objectives and provides an approach to develop co-operation among these objects. The people in each department control and operate on that departments data. So, the division of the function into smaller objects help to comprehend and control the activity of the companies, and to maintain the integrity of information used.

The concept of recursive functions has developed recently, with the development of another programming technique called recursive programming that helps in reduction of time and cost. Another important feature in this field is the development of concurrent programming that offers the advantages of data sharing and synchronization. The latest addition in programming techniques is the introduction of functional programming. Which contributes the concept of modularity, which is the key to successful programming and is vitally important to the world.

Structural Programming

Structural programming concepts were developed during the late 60's and early 70's. Structural programming involves the use of standard programming constructs. In structured programming, all programs consist of a number of modules with little interaction between them. These modules are made up of a group of instructions performing one identifiable function, independent of other modules. This approach to programming is also referred to as procedure-oriented programming, where a problem is considered as a sequence of things like reading, calculating and printing. The problem is divided into various sub-problems called 'functions' and involves the writing of a list of instructions for the computer to follow and organize these instructions into groups. Here, functions, subroutines or procedures are introduced to make the programs more comprehensive. Each function does have a clearly defined purpose and interface with other functions in the programs.

It is a structured program; the primary purpose of which is divided into smaller pieces called 'functions' within the programs, having its own data and logic. Message is passed between these functions using parameters and functions having local data that can't be accessed outside the scope of the function. Since the structured program provides for isolating processes within functions, it minimizes the chance of one procedure affecting the other. It helps in writing a clear code and maintaining control over each function. The need for global variables can be minimized by replacing them with local variables that are easily controllable.

Structured programming uses the system approach, where the problem is divided into sub problems that are further divided into small manageable problems. It employs standard tools and program constructs like sequence, selection and iteration. The programmers use standard methods for coding sequence for the normal flow of control in the natural order, that is, from top to bottom, and from left to right, unless there is an intervening control structure that changes the flow of control. Control is exercised in the first instruction and when it is over, the control passes to the next instruction in that sequence. Selection involves the design regarding flow control and such constructs include if then, if condition then', if then else', etc. Iteration, allows repeated execution of a set of codes without repeating codes in the programs. Loops like 'Do... while',' Repeat... until', and for... next', etc., are examples of iteration constructs.

An important concept of structural programming is obstruction, which permits the programmer to look at something without being concerned with its internal details. In a structured program, the problem is to know which task is performed by a function, how the task is performed is immaterial. This is known as functional abstraction, considered as the corner stone of structured programming. The problem in a procedural language is that the whole emphasis is on doing things. Data is given only a secondary status. Data types are processed in many functions and when changes occur in data types, modifications are needed at every location on these data types within the program. It is a highly time consuming and frustrating task. Moreover, the functions and data structures do not model the real world situations. The hierarchical structure of structural programming is depicted in the following figure:

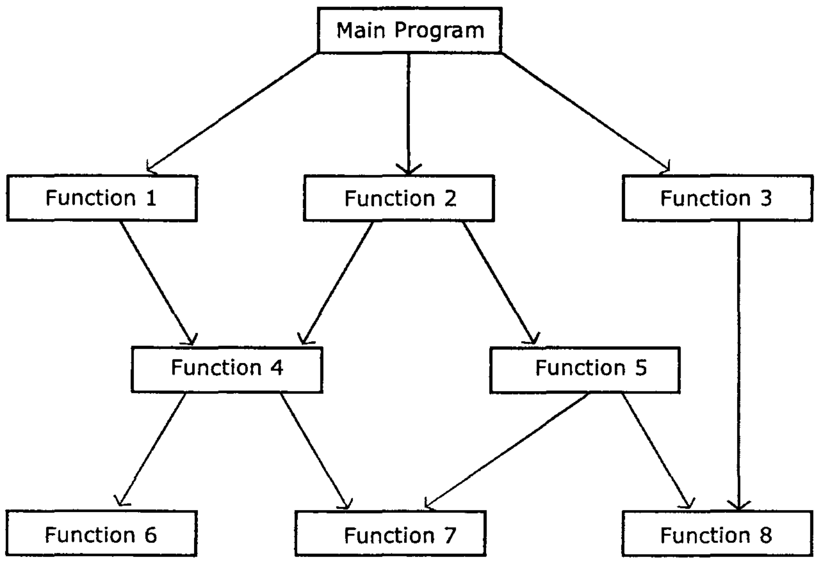

Fig. 10.1: Structure of a standard program

The important features of structured programming are: the emphasis on doing things (algorithms), division of large programs into functions, sharing global data, open movement of data around the system from function to function, transformation of data from one form to another by various functions, and employment of top down approach in the design of the program. The important structural programming languages are COBOL, FORTRAN, and C.

Recursive Programming

For many problems, recursion is the natural method of solution. Such problems occur frequently in mathematical logic and the use of recursion will often results in programs, which are both elegant and simple. A programming technique that involves the use of recursive functions is called recursive programming. A recursive function is one which helps to reduce a problem to a sequence of simple steps. It requires a recursive step and a stopping condition. A recursive program to find out the factorial is illustrated below.

```
        →( defun factoral (n))
           ( cond ( ( zerop n)1))
           ( t ( *n ( factorial(-n1))))
FACTORIAL
        →(factorial 6)
        720
        →
```

A recursive program which defines the member function, called 'newmember', can be represented as:

```
        →defun newmember ( el 1st}
           ( cond ((null 1st) nil)
           ( ( equal el ( car1st))1st)
           ((t ( newmember el ( cdr 1st )))
        NEWMEMBER
        →
```

Recursive programming provides for the application of same recursive function in several problems. Thus time can be saved, and the workload reduced.

Object Oriented Programming (OOP)

The Object Oriented Programming concept is built on the foundation laid by structured programming concepts and data abstraction. It refers to the expression of computer programs in such a way that they reflect the perception of the people in the modern object oriented world. This approach considers data as a critical resource in the program development and thus prohibits the free flow of data in the system environment. In object oriented programming, the problem is decomposed into a number of objects, and then data and functions are built around these objects. Such data can be accessed only with the help of functioning associated with these objects. Object-oriented programming allows the programmers to deal with their problem domains by concentrating on the objects and by permitting the features of these objects to determine the procedures to follow. Thinking in terms of objects has a surprisingly helpful effect on how easily programs can be designed.

The division of problem into objects offers a valid revolution in programs design. In reality the term 'object-oriented programming conveys different meanings to different people, and is the most recent development in the programming paradigms. "It is an approach that provides a way of modulating 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".

Features of Object Oriented Programming

The basic features of Object oriented programming are:

(a) Programs are divided into objects;

(b) Programs are designed by following bottom up approach;

(c) Objects are communicated with each other with the help of functions;

(d) The method focus on data rather than on procedures;

(e) Insertion of new data and functions is very easy;

(f) Data is hidden and cannot be accessed by external functions;

(g) Data structures are designed in such a way that they characterize the objects;

(h) Functions that operate on the data of an object are tied together in the data structure.

Concepts Underlying Object Oriented Programming

Object-Oriented Programming uses an unfamiliar battery of vocabulary regarding the major elements of object-oriented languages. The important terms used in object oriented programmings are:

(i) Objects: In OOP, a program is divided into various objects that are the basic run-time entities in an object oriented system: It may be a person, place, bank, table of data, employees on a payroll, data structure like linked lists, stacks, queues, etc.; GUI elements such as windows, menus, icons, etc.; hardware devices like disk drive, keyboard, printer, etc.; various games in computer and its elements such as cannons guns, animals, etc.; customers sales persons in sales tracking system and computers in a network model.

A problem is analyzed in OOP in terms of objects and the nature of communication between them. Program objects need to be closely matching with the real world objects that take up space in memory and have an associated address like a record in PASCAL or a structure in C. While executing a program the objects interact by sending messages to one another. Suppose the customer and account are two objects in a program, they interact by sending a message for the bank balance. Since each object contains data and a code to manipulate it, one object can interact with the other only if it knows the details of the data and code of the other. An object can be represented in two ways. The popularly used notations are:

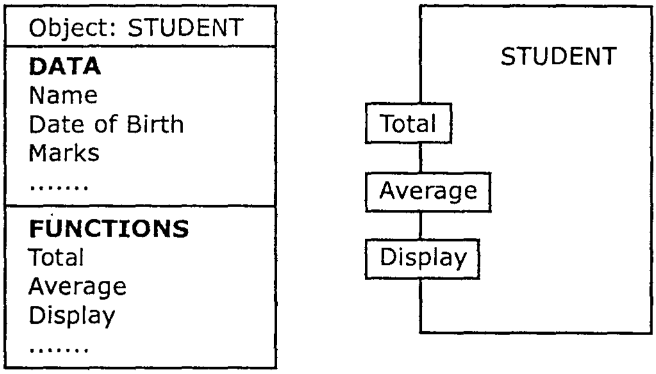

Fig. 10.2: Representation of objects

(ii) Classes: The entire set of data and code of an object can be made user-defined data types with the help of a class. The programmer can define the format and behavior of a language with the help of a class. For instance, there can be a user-defined data type to represent dates. Programmers have to define the behavior of dates by defining the date class, which gives the format and operations to be performed on the date. A class is a blueprint or action or a template specifying what data and function will be included in objects of what class. A class contains a number of objects associated with the data type of the class. For example, fruit is taken, as a class, then mango, orange, apple, etc., are members of that class. Classes' behave like built in type programming languages. If 'fruit' has been defined as a class then the statement, 'fruit mango' will create "object mango ' in the class of fruit.

(iii) Data abstraction and encapsulation: Wrapping of data functions into a single unity class is called encapsulation. The data is not accessible to the outsiders, only the functions wrapped in it can be accessed by them. The functions provide interface between the objects, data and the program. The insulation of data from direct access by the program is called 'data hiding'.

Data abstraction refers to the act of representing the essential feature of classes, without including the background details or explanation. Classes use the concept of data abstraction and define a list of abstract attributes such as size, weight and cost, and functions to operate on these attributes. They encapsulate all the essential prospects of the objects to be created. Since the classes use the concept of data abstraction, they are called Abstract Data Types (ADT)

(iv) Inheritance: It is the process by which the objects of one class acquire the properties of objects of another class. It supports the concept of hierarchical classification. With the help of this concept new classes can be built from old ones. Then the new class is referred to as a derived class, which can inherit the data structures and functions of the original or the base class, it can also add data elements and functions to those, which it inherits from its base class.

The concept of inheritance provides the idea of reusability. It means that additional features can be inserted into an existing class without modifying it. The real benefit of the inheritance mechanism is that it allows the programmer to reuse a class in such a way that it does not introduce any undesirable side effects. For example, the bird robin is a part of the class' flying bird', which is again a part of the class 'bird'. It can be illustrated as follows:

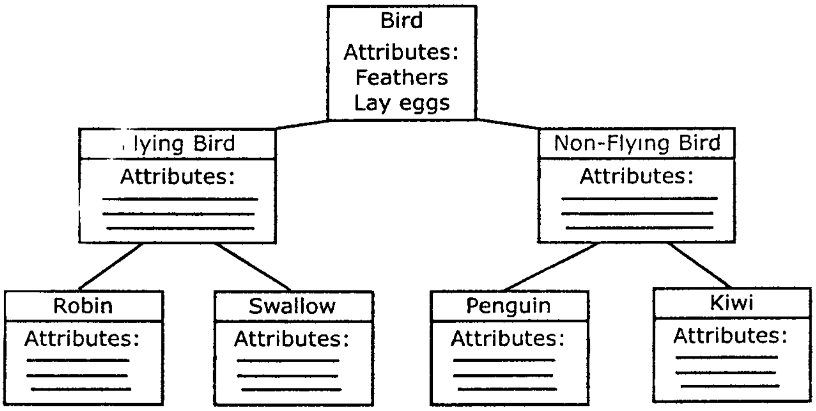

Fig. 10.3: Inheritance

(v) Polymorphism: It refers to the ability to take more than one form. It has a significant role in allowing objects having different internal structures to share the same external interface. Thus, a general class of operation may be accessed with the same manner, even though specific actions associated with each operation may differ. It is highly useful in implementing the concept of inheritance in OOP. In polymorphism, an operation may exhibit different behaviors in different circumstances. Consider the operation of addition. For any two numbers, the operation will give a sum. If the operands are strings, then the operation would produce a third string. The concept of polymorphism is reflected in the following figure.

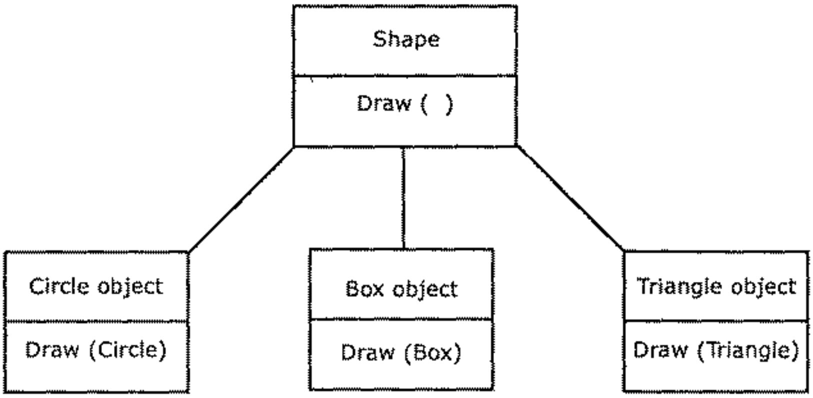

Fig. 10.4: Polymorphism

(vi) Dynamic Binding: 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 procedure call is not known until the time of the call at run-time. It is associated with polymorphism and inheritance.

(vii) Message Communication: Object-oriented programming involves three steps such as: creation of classes to define the objects from class definition, and establishment of communication among various objects. Message communication is possible by sending and receiving information among various objects. A message from an object is a request for the execution of a procedure and will invoke a function in receiving the object that produces the required result. Message passing involves specifying the name of the object, name of the function (message), and the information to be sent. It can be illustrated with the help of the example given in the following figure:

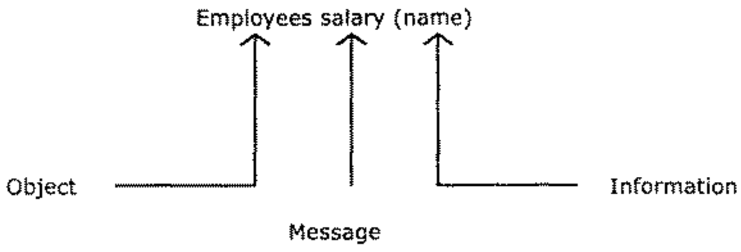

Fig. 10.5: Message communiction

Object orientation provides solutions to a number of problems connected with the development and quality of software products. It ensures greater productivity and quality of software products. It ensures greater productivity of the programmer, better quality programmers, and lesser maintenance cost. It offers several advantages, like elimination of redundant code, extension of the uses of existing classes, building up of programs from standard working modules that communicate with one another, saving of development time and ensuring higher productivity, building up of secure programs that cannot be invaded by the code in other parts of the program, co-existence of an object without any interference, mapping of objects in the problem domain to those objects in the program, easy partition of work in a project on the basis of objects, capturing more details of a model in implementable form, easy aggregations, simple interface description with external system, and easy management of software complexity.

Object Oriented Language

Object-oriented programming can be performed with the help of languages like C and PASCAL. Languages that are specifically designed to support the OOP can be divided into two categories, like object-based programming languages, and object oriented programming languages. Object-based programming supports encapsulation and object identity, and requires features like data encapsulation, data hiding and access mechanism, automatic initialization and clear-up of objectives and operator overloading. An example of objectives based programming language is Ada.

Object-oriented programming languages involves the features of object based programming languages along with data inheritance and dynamic binding. Examples of Object-oriented programming languages are C++, small talk, and object PASCAL.

Object-oriented programming is highly significant in the present-day context, since it can support the functions of software engineers in the performance of their functions. It is highly useful in the area of user interface design like windows. It is also useful in real business applications involving complex problems. The other areas to which OOP can contribute are real-time systems, simulation and modeling, object-oriented databases, hypertext, hypermedia and expert text, Artificial Intelligence and Expert Systems, natural network and parallel programming, decision support and office automation system, and CAM/CAD/CIM systems.

System Development through Object-Oriented Technology

System development is a complex process beginning with the understanding of end-user requirements. The user requirements are presented in terms of data information and access needs. The use of object-oriented technology requires programming languages for implementation. Such languages are higher order languages, and they stand out differently over second and third generation languages. The requirement specification emerges after close interaction with the people involved in the system. The system developers will not understand and will not be in a position to use the requirement specification in a text mode. It needs to be modeled for ease of understanding and communication. Object oriented technology uses the following three steps to develop a requirement model:

(a) System domain defining the broad scope of the system for which the model is to be built,
(b) Process function performed in the system known as use case,
(c) Interface which describes the association and connection with other use cases.

Concurrent Programming

Concurrent programming involves the functioning of sequential processes as that are carried out simultaneously.

The processes operate on common tasks by exchanging data through shared variables. A concurrent program is an extension of a sequential program, where it specifies two or more sequential programs that may be executed simultaneously/concurrently as parallel processes. A concurrent program can be executed either by allowing processes to share one or more processors or by running each process on its own processor. This task is applied in the execution of most parts of multi programming and multi processing.

Earlier distributed systems were programmed in traditional sequential languages. But, with new innovations in the field of IT, the traditional languages get obsolete and thus it is essential for researchers to develop a variety of design goals, size, performance and applications. These new programming techniques address three problems, such as: ability to execute different pieces of a program on different processors, ability to make the pieces co-operate with each other, ability to cope with partial failure of the distributed system. A distributed computing system consists of a number of autonomous processors that do not share primary memory, but co-operate by sending messages over a communication network. In a system of this structure it executes it own instruction streams in both stored memory and its local memory. In multi-processors communication takes place through shared memory. Generally, there are two types of distributed systems, a closely coupled distributed system and a loosely coupled system.

The important features of concurrent programming are data sharing and synchronization. Data sharing involves public data variables to its program, and it must be able to share data structure like buffer. Otherwise, concurrent processes cannot exchange data and co-operate on common tasks. All the processors need to know that they can send and receive data through these shared variables. However, the sending and receiving of data must communicate through shared memory.

In order to co-operate, concurrent-executing processes must communicate and synchronize. Communication allows the execution of one process to influence the execution of another. Because processes are executed with unpredictable speeds, synchronization is necessary. Synchronization is the process of two or more processors calling upon each other's information to conclude their own process. Synchronization mechanisms can delay the execution of a process until a given condition is true, and can be used to ensure that a block of statements is an indivisible operation, therefore elimination of the possibility of statements in other processes, which interfering with assertions appearing within the proof of that block of statements.

The important strategies for implementing concurrent programming are:

(a) Vector computers: They can use many processors that simultaneously apply the same arithmetic operations to different data and are highly useful for computational intensive numerical applications

(b) Multicomputers: They consist of several autonomous processors instead of communicating by sending messages over a communication network.

(c) Workstations or minicomputers: They are connected by LAN or WAN, and are frequently the targets of distributed operating systems.

(d) Dataflow and reduction machines: They can apply different operations to different data.

(e) Multiprocessors: They involve several autonomous processors sharing a common primary memory and are best suited for running different subtasks of the same program simultaneously.

Functional Programming

Functional programming is so called because a program consists entirely of functions. The main program itself is written as a function, which receives the program's input as its argument, and delivers the output as its result. Typically the main function is defined in terms of other functions, which, in turn, redefined in terms of still more functions until at the bottom level the functions are language primitives. These functions are much like mathematical functions. The special advantages of functional programming are:

(a) Functional programs do not contain assignment statements, and hence variables once given a value never change.

(b) Functional programs do not have any side effects.

(c) A function call has no effect other than computing its result, and thus eliminating source of bugs.

(d) A function can be evaluated at any time, and it relieves the programmer of the burden of prescribing the flow of control.

(e) Functional programs are referentially transparent.

The advantages of structural programming are very similar in spirit to those of functional programming. Functional languages provide two new important kinds of glue. They allow greatly improved modularization. The new kind of glue enables the simple functions to be glued together to make more complex ones. It can be illustrated with a simple list-processing problem – adding up the elements of a list, thus:

List x : : = nil / cons x (list of x)

This means that a list of x's is either nil representing a list with no elements, or it in a cons of an x and another list of x's. A cons represents a list where first element is the x and whose second and subsequent element are the elements of the other list of x's (x stands for any type).

LISP (LISt Programming)

LISt Programming has historically been the most commonly used language for Artificial Intelligence programming. It is a special purpose language that is suited for areas like list processing artificial intelligence, robotics, etc., and is designed to manipulate the non-numeric data. Several competing dialects of LISP are available, but common LISP is accepted as a standard. For writing parallel programs, parallel LISP models are available, such as Multiples, QLISP and Paralation model.

Variables in LISP are symbolic (non-numeric) atoms that are assigned values with the function setq. Setq takes two arguments. The first argument must be a variable. It is never evaluated and should not be in quotation marks. The second argument is evaluated and the result is bound to the first argument. The variable retains this value until a new assignment is made. When variables are evaluated they return the last value bound to them. Comments in LISP code may be placed anywhere after a semicolon. Examples of the use of setq are:

```
  →( Setq x 10)       ; the number 10 evaluates to itself
     10               ; is bound to x and 10 is returned
  → x                 ; the variable x is evaluated to
     10               ; return the value it is bound to
  → ( Setq x (+35)    ; x is reset to the value (+35)
 8                    ; and that value is returned
  → ( Setq xi (+35)   ; x is reset to the literal value
 (+35)                ; (+35), quote inhibits evaluation
  → y                 ; the variable Y was not previously
 Unbound variable:y   ; bound to a value, causing an error.
```

It is to be noted that in LISP, trying to evaluate an undefined variable (not previously bound to a value) results in an error. Some basic LIST manipulation functions in LISP are:

Function Call	*Value Returned*	*Remarks*
(car' (a b c))	a	Care takes one argument, a list and retunes the first element.
(cdr' (abc))	(b c)	Cdr takes one argument, a list with first element removed.
(Cons'a'(bc))	(a b c)	Cons takes two arguments ,an element and a list and returns a list with the element at the beginning
(list'a' (bc))	(a (b c)	List takes any number of arguments and returns a list with the arguments as elements.

Other useful list manipulation functions are: Append, Last, Member and Reverse. Append meets arguments of one or more lists into a single unit; List takes one argument and returns a list containing the last element. Member takes two arguments and returns the remainder of the second argument list starting with the elements matching the first argument. Revese take a list as its argument and returns a list with top elements in reverse order. The additional list manipulation functions in LISP are:

Function Call	*Value Returned*	*Remarks*
(append '(a)'(bc)	(a b c)	Merges two or more lists into a single list.
(last '(ab c d))	(d)	Returns a list containing the last element.
(member 'b' (a b d))	(b d)	Returns the remainder of the second argument list starting with the element matching the first argument.
(reverse ' (a (bc) d))	(d (b c) a)	Returns the list with top elements in the reverse order.

One of the unique and most useful features of LISP as an AI language is its ability to assign properties to atoms. In LISP, any object, for example an atom, which represents a person, can be given a number of properties which characterize the person, like height, weight, sex, habit, colour, address and profession. Property list functions in LISP permit one to assign such properties to an atom and to retrieve, replace or remove them as required. The function put prop assigns properties to an atom and takes three arguments such as object name, attribute (property) name, and attribute value. A LISP statement giving attributes or properties to a car can be written as:

```
→(putprop 'car'ford'make)
FORD
→(put prop 'car'1988' year)
```

```
1988
→(put prop 'car' 'red' colour )
RED
→(Put prop' car' Four-door' style)
FOUR-DOOR
→
```

With the help of these statements made in put prop function, we can assign properties like : make year, colour and style, to a car. To retrieve a property value in LISP, such as colour, style and year., we can use another function called'get', which takes two arguments, objects and attributes. Single or multiple dimension arrays may be defined in LISP using make-array function. The items stored in the array may be any LISP object to store items in array we use the self function. Certain other functions included in the presentation of LISP are(a) mapping functions like Mapcar that can be used in a variety of ways in lieu of interactive functions, and can be applied either to user-defined lists or to built in lists, (b) Lamda functions through which the LISP provides a method of writing unnamed or anonymous functions that are evaluated only when they are encountered in a program.

Logic Programming

Logic programming is a programming language paradigm (organizational principles), in which logical assertions are viewed as programs. The most popular logical programming system is PROLOG (PROgramming LOGic). Invented by Alain Colmerauer and Associates at the University of Marseilles, during the early 1970s. It is described as a series of logical assertions, each of which is a home clause. Home clause is a clause that has almost one positive literal. PROLOG uses the syntax of predicate logic to perform symbolic and logical computations. Programming in PROLOG is accomplished by creating a database of facts and rules about objects, their properties and their relationship with other objects. Queries can be posed about the objects, and valid conclusions will be arrived at and returned by the program. In PROLOG, quantification is made implicitly by way of interpreted variables, explicit symbols are used for 'and' (^), 'and/or' (v), etc. in logic.

In PROLOG, explicit symbol is used for 'and' (,) , but there is none for 'or'.

Facts in PROLOG are declared with predictions and constants, written in lowercase letters. The arguments of predicate are enclosed in parentheses and separated by commas. For instance, facts about family relationships can be written in PROLOG as follows:

Sister (sue,bill)

Parent (ann, sam)

Parent (joe,ann)

Male (joe)

Female (ann)

In Logic, implication of the form" p implies q" is written as

$p \rightarrow q$. In PROLOG , it is written as q:-p "backward". Following is an example of a knowledge base represented in logical notation and in PROLOG

Representation in LOGIC

$\forall_x$: pet (x) ^ small (x) $\rightarrow$apartment pet (x)

$\forall_x$: cat (x) v dog (x) $\rightarrow$pet (x)

$\forall_x$: poodle (x) $\rightarrow$dog (x) ^ small (x)

Poodle (fluffy)

Representation in PROLOG

Apartment(x) : - pet (x), small (x)

Pet (x) : - cat (x).

Pet (x) : - dog(x).

Dog(x) : - poodle (x).

Small (x) : - poodle (x).

Poodle (fluffy).

Rules in PROLOG are composed of a condition, or"if" part, and a conclusion, the" then" part, separated by the symbol': - which is read as 'If'. Rules may contain variables which must begin with uppercase letters. Lists in PROLOG are similar to list data structures in LISP. A PROLOG list is written as a sequence of items separated by commas and enclosed in square brackets as follows:

[tom, sue, joe, mary, bill]

A number of list manipulation predicates are available in most PROLOG implementations, including Append, Member, Cone (concatenate), Add, Delete, etc. They have numeric functions, relations and list handling capabilities, which give them some similarity to LISP.

A great advantage of logic programming is that the programmer needs only to specify rules and facts, since a search engine is built directly into the language, and a search control mechanism is fixed in it.

Conclusion

Programming techniques are used for translating the information system design into a language, which is understood by the computer. A set of instructions or commands conveying the design of the information system in terms of input-process-output, called source code, is read by the computer with the help of a computer, which in turn produces a object code. The object code is taken by the compilers, which translates it into a code that drives the macro code logic in the CPU of the computer. The desirable features of a programming technique are: easy to use, compiler efficiency, source code portability, availability of development tools, and maintainability. Moreover, the following guidelines would serve as a measure of a better programming technique: (a) use of simple arithmetic and logical expressions (b) use of simple nested loops, (c)minimum use of multidimensional arrays, (d) separate treatment to different data types, (e) use of formulae in arithmetic operations, and (f) use of boolean expressions for quick processing.

Exercise

Short Answer Questions

1. What do you mean by a program?
2. What are the characteristic features of a good program?
3. Explain recent trends in programming techniques.
4. What is mean by a structured program?
5. Explain the concept of structured programming.
6. Describe recursive programming.
7. What do you mean by object-oriented programming?
8. Explain the important features of object-oriented programming.
9. What are the concepts underlying object oriented programming?
10. What do you mean by abstract data type?
11. Explain the concept of inheritance in object-oriented programming.
12. Write short note on data abstraction and encapsulation
13. Describe the concept of polymorphism.
14. What are the important object oriented programming language?
15. Explain the significance of object oriented programming
16. Explain the concurrent programming techniques.
17. What is a concurrent program?
18. What are the features of concurrent programming?
19. What are the important strategies for implementing concurrent programming?
20. What do you mean by functional programming?
21. Describe the merits and demerits of functional programming.
22. What do you mean by LISP?
23. What are the important List manipulation functions?
24. What do you mean by Artificial Intelligence?
25. Briefly explain Logic programming.
26. What is PROLOG?
27. Explain the importance of programming techniques.
28. What do you mean by recursive function?
29. Explain system approach in structured programming.
30. Explain the concept of dynamic binding in object-oriented programming.

Essay Questions

1. Explain various object oriented programming techniques.
2. What do you mean by structural programming and what is its significance?
3. Describe the trends in programming techniques.
4. Explain concurrent, logic and functional programming.
5. Discuss various programming techniques.