

Chapter 3: Designing Efficient Programs

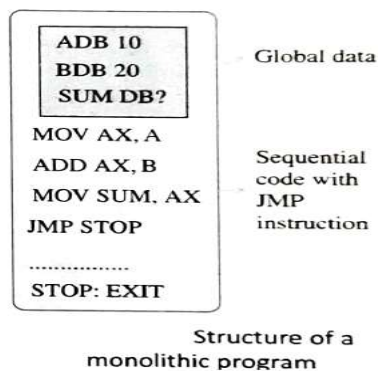
3.1 PROGRAMMING PARADIGMS

A programming paradigm is a fundamental style of programming that defines how the structure and basic elements of a computer program will be built. These paradigms, in sequence of their application, can be classified as follows:

- Monolithic programming - emphasizes on finding a solution
- Procedural programming - lays stress on algorithms
- Structured programming - focuses on modules
- Object-oriented programming - emphasizes on classes and objects
- Logic-oriented programming - focuses on goals usually expressed in predicate calculus
- Rule-oriented programming - makes use of 'if-then- else' rules for computation
- Constraint-oriented programming - utilizes invariant relationships to solve a problem

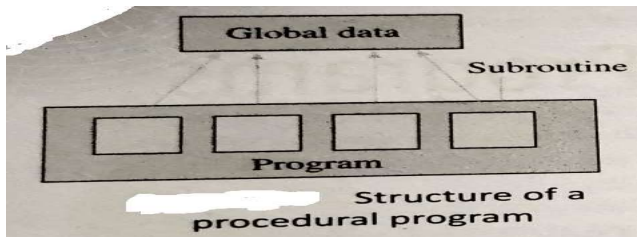
Monolithic Programming – (emphasizes on finding a solution)

Programs written using monolithic programming languages such as assembly language and BASIC consist of global data and sequential code. The global data can be accessed and modified (knowingly or mistakenly) from any part of the program, thereby posing a serious threat to its integrity. A sequential code is one in which all instructions are executed in the specified sequence. In order to change the sequence of instructions, jump statements or 'goto' statements are used.



Procedural Programming - lays stress on algorithms

In procedural languages, a program is divided into subroutines that can access global data. To avoid repetition of code, each subroutine performs a well-defined task. With jump, goto, and call instructions, the sequence of execution of instructions can be altered. FORTRAN and COBOL are two popular procedural programming languages.



Advantages

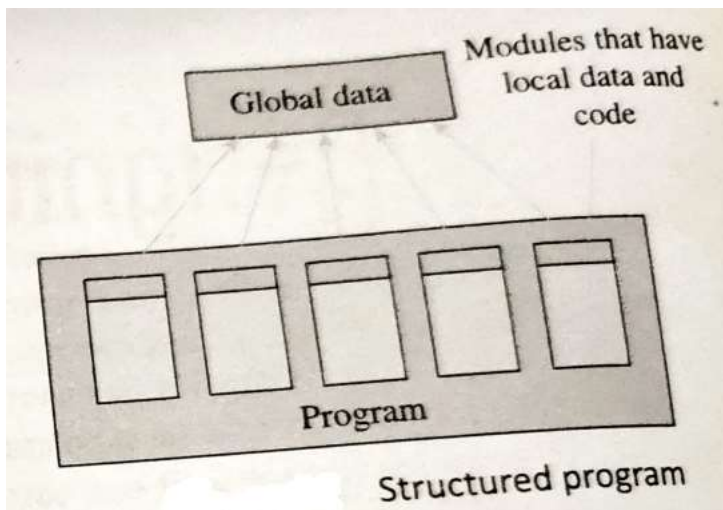
- The only goal is to write correct programs
- Programs are easier to write as compared to monolithic programming

Disadvantages

- No concept of reusability
- Requires more time and effort to write programs
- Programs are difficult to maintain
- Global data is shared and therefore may get altered (mistakenly)

Structured Programming - focuses on modules

Structured programming, also referred to as modular programming. Structured programming employs a top-down approach in which the overall program structure is broken down into separate modules. Modules are coded separately and once a module is written and tested individually, it is then integrated with other modules to form the overall program structure. Almost every modern programming language similar to C, Pascal, supports the concepts of structured program.



Besides sequence, structured programming also supports selection and repetition.

Selection - allows for choosing any one of a number of statements to execute, based on the current status of the program. Ex- if, else, switch.

Repetition – A selection statement remains active until the program reaches a point where there is a need for some other action to take place. Ex – while, do, for

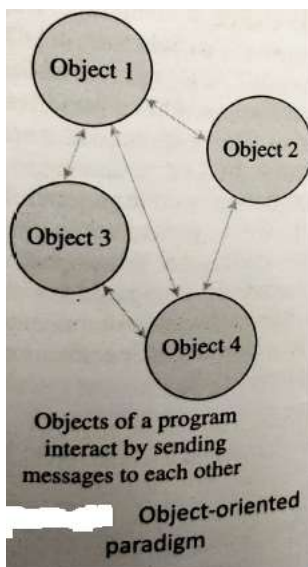
Advantages

- The goal of structured programming is to write correct programs that are easy to understand and change.
- Modules enhance programmers' productivity by allowing them to look at the big picture first and focus on details later.
- With modules, many programmers can work on a single, large program, with each working on a different module.
- A structured program takes less time to be written than other programs. Modules or procedures written for one program can be reused in other programs as well.
- Each module performs a specific task.
- Each module has its own local data.
- A structured program is easy to debug.
- Individual procedures are easy to change as well as understand.
- More emphasis is given on the code and the least importance is given to the data.

Disadvantages

- Not data centered
- Global data is shared and therefore may get modified
- Main focus is on functions

Object-oriented programming - emphasizes on classes and objects



In this paradigm, all the relevant data and tasks are grouped together in entities known as objects. For example, consider a list of numbers- In the object-oriented paradigm, the list and the associated operations are treated as one entity known as an object. In this approach, the list is considered an object consisting of the list, along with a collection of routines for manipulating the list. In the list object, there may be routines for adding a number to the list, deleting a number from the list, sorting the list, etc.

The striking features of OOP include the following :

- Programs are data centered.
- Programs are divided in terms of objects and not procedures.
- Functions that operate on data are tied together with the data.
- Data is hidden and not accessible by external functions.
- New data and functions can be easily added as and when required.
- Follows a bottom-up approach for problem solving

3.2 EXAMPLE OF A STRUCTURED PROGRAM

We want to create a program to manage the names and addresses of a list of students. For this, we would need to break down the program into the following modules:

- Enter new names and addresses
- Modify existing entries
- Sort entries
- Print the list

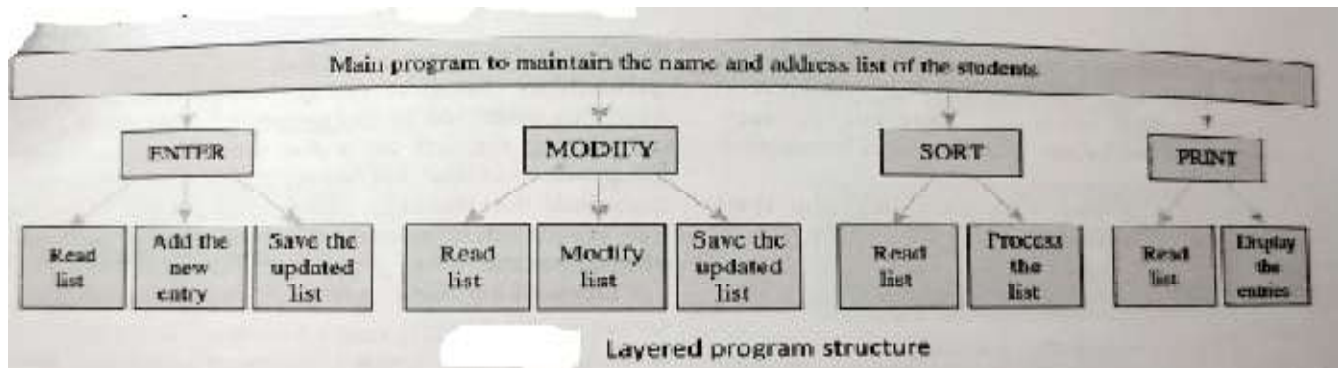
Now, each of these modules can be further broken down into smaller modules. For example, the first module can be subdivided into modules such as follows:

- Prompt the user to enter new data
- Read the existing list from the disk
- Add the name and address to the existing list
- Save the updated list to the disk

Similarly, 'Modify existing entries can be further divided into modules such as follows:

- Read the existing list from disk
- Modify one or more entries
- Save the updated list to disk

The two sub-modules 'Read the existing list from disk' and 'Save the updated list to disk' are common to both the modules. Hence, once these sub-modules are written, they can be used in both the modules, which require the same tasks to be performed. The structured programming method thus results in a hierarchical or layered program structure.



3.3 DESIGN AND IMPLEMENTATION OF EFFICIENT PROGRAMS (SDLC)

The entire program or software (collection of programs) development process is divided into a number of phases, where each phase performs a well-defined task. Moreover, the output of one phase provides the input for its subsequent phase.

The phases in the SDLC process can be summarized as follows:

Requirements analysis In this phase, the user's expectations are gathered to know why the program /software has to be built. Then, all the gathered requirements are analysed to arrive at the scope or the objective of the overall software product. The last activity in this phase includes documenting every identified requirement of the users in order to avoid any doubts or uncertainty regarding the functionality of the programs.

Design The requirements documented in the previous phase acts as an input to the design phase. In the design phase, a plan of actions is made before the actual development process can start. This plan will be followed throughout the development process. Moreover, in the design phase, the core structure of the software program is broken down into modules. The solution of the program is then specified for each module in the form of algorithms or flowcharts. The design phase, therefore, specifies how software

Implementation In this phase, the designed algorithms are converted into program code using any of the high-level languages. The particular choice of language will depend on the type of program, such as whether it is a system or an application program. While is preferred for writing system programs, Visual Basic might be preferred for writing an application program. The program codes are tested by the programmer to ensure their correctness.

This phase is also called construction or code generation phase as the code of the software is generated in this phase.

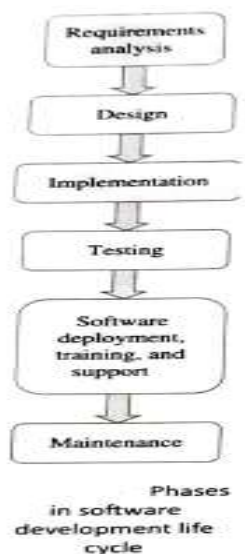
Testing In this phase, all the modules are tested together to ensure that the overall system works well as a whole product. Although individual pieces of codes are already tested by the programmers in the implementation phase, there is always a chance for bugs to creep into the program when the individual modules are integrated to form the overall program structure.

In this phase, the software is tested using a large number of varied inputs, also known as test data, to ensure

that the software is working as expected by the user's requirements that were identified in the requirements analysis phase.

Software deployment, training, and support After the code is tested and the software or the program has been approved by the users, it is installed or deployed in the production environment. This is a crucial phase that is often ignored by most developers. As a part of the deployment phase, it has become very crucial to have training classes for the users of the software.

Maintenance Maintenance and enhancements are ongoing activities that are done to cope with newly discovered problems or new requirements. Such activities may take a long time to complete as the requirement may call for the addition of new code that does not fit the original design or an extra piece of code, required to fix an unforeseen problem.



3.4 PROGRAM DESIGN TOOLS: ALGORITHMS, FLOWCHARTS, PSEUDOCODES

3.4.1 Algorithms

The algorithm gives the logic of the program, that is, a step-by-step description of how to arrive at a solution.

In general term an algorithm provides a blueprint to writing a program to solve a particular problem.

In order to qualify as an algorithm, a sequence of instructions must possess the following characteristics:

- It should have finite number of steps
- Be precise
- Be unambiguous
- Not even a single instruction must be repeated infinitely.
- After the algorithm gets terminated, the desired result must be obtained.

Control Structures Used In Algorithms

An algorithm may employ three control structures, namely, sequence, decision, and repetition.

Sequence - Sequence means that each step of the algorithm is executed in the specified order.

Decision – These statements are used when the outcome of the process depends on some condition.








Repetition - Repetition, which involves executing one or more steps for a number of times, can be implemented using constructs such as the while, do-while, and for loops. These loops execute one or more steps until some condition is true.

3.4.2 Flowcharts

A flowchart is a graphical or symbolic representation of a process. It follows a top down approach in solving a problem.

Significance of flowcharts

- *It is drawn in the early stages of formulating computer solutions*
- *It facilitates communication between programmers and users*
- Programmers can make users understand the solution easily and clearly

Symbol	Name	Function
	Process	Indicates any type of internal operation inside the Processor or Memory
	input/output	Used for any Input / Output (I/O) operation. Indicates that the computer is to obtain data or output results
	Decision	Used to ask a question that can be answered in a binary format (Yes/No, True/False)
	Labelled Connector	Allows the flowchart to be drawn without intersecting lines or without a reverse flow.
	Predefined Process	Used to invoke a subroutine or an Interrupt program.
	Start/Stop	Indicates the starting or ending of the program, process, or interrupt program
	Flow Lines	Shows direction of flow.

Advantages

- They are very good communication tools to explain the logic of a system to all concerned.
- They are also used for program documentation.
- They act as guide or blueprint for the programmers.
- They direct the programmers to go from the starting point of the program to the ending point without missing any step in between.
- They can be used to debug programs that have errors.
- They help the programmers to easily detect, locate, and remove mistakes in the program in a systematic manner.

Limitations

- Drawing flowcharts is a laborious and a time-consuming activity.
- The flowchart of a complex program becomes complex and clumsy
- At times, a little bit of alteration in the solution may require complete redrawing of the flowchart
- The essentials of what is done may get lost in the technical details of how it is done,
- There are no well-defined standards that limit the details that must be incorporated into a flowchart

3.4.3 Pseudocodes

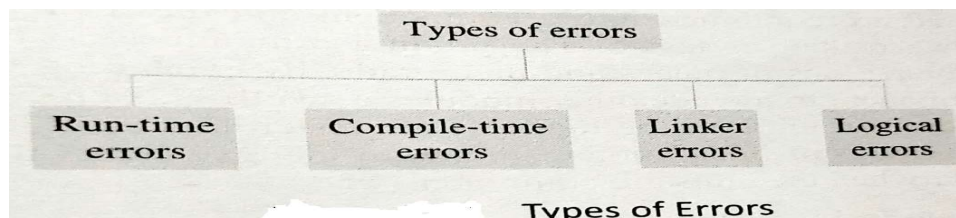
Pseudocode is a compact and informal high-level description of an algorithm that uses the structural conventions of a programming language. An ideal pseudocode must be complete, describing the entire logic of the algorithm, so that it can be translated straightaway into a programming language.

Features

- The sole purpose of pseudocodes is to enhance human understandability of the solution.
- They are commonly used in textbooks and scientific publications and for documenting algorithms, and for sketching out the program structure before the actual coding is done.
- This helps even non-programmers to understand the logic of the designed solution.
- There are no standards defined for writing a pseudocode, because a pseudocode is not an executable program. Flowcharts can be considered graphical alternatives to pseudocodes, but require more space on paper.

8.5 TYPES OF ERRORS

While writing programs, very often we get errors in our programs. These errors if not removed will either give erroneous output or will not let the compiler to compile the program.



Run-time Errors - Run-time errors occur when the program is being run executed. Such errors occur when the program performs some illegal operations like -

- dividing a number by zero
- opening a file that already exists
- lack of free memory space
- finding square root or logarithm of negative numbers

Run-time errors may terminate program execution, so the code must be written in such a way that it handles all sorts of unexpected errors rather terminating it unexpectedly.

Compile-time Errors - Compile- time errors occur at the time of compilation of the program.

Such errors can be further classified as follows:

Syntax Errors - Syntax errors are generated when rules of a programming language are violated.

Semantic Errors - Semantic errors are those errors which may comply with rules of the programming language but are not meaningful to the compiler.

Logical Errors - Logical errors are errors in the program code that result in unexpected and undesirable output which is obviously not correct. Such errors are not detected by the compiler, and programmers must check their code line by line or use a debugger to locate and rectify the errors.

Linker Errors - These errors occur when the linker is not able to find the function definition for a given prototype.

3.6 TESTING AND DEBUGGING APPROACHES

Testing is an activity that is performed to verify correct behavior of a program. It is specifically carried out with an intent to find errors. Ideally testing should be conducted at all stages of program development.

However, in the implementation stage, the following three types of tests can be conducted

Unit Tests Unit testing is applied only on a single unit or module to ensure whether it exhibits the expected behavior.

Integration Tests These tests are a logical extension of unit tests. In this test, two units that have already been tested are combined into a component and the interface between them is tested. This process is repeated until all the modules are tested together. The main focus of integration testing is to identify errors that occur when the units are combined.

System Tests System testing checks the entire system. For example, if our program code consists of three modules then each of the module is tested individually using unit tests and then system test is applied to test this entire system as one system.

Debugging - It is an activity that includes execution testing and code correction. The main aim of debugging is locating errors in the program code. Once the errors are located, they are then isolated and fixed to produce an error-free code.

Different approaches applied for debugging a code includes:

Brute-Force Method In this technique, a printout of CPU registers and relevant memory locations is taken, studied, and documented. It is the least efficient way of debugging a program and is generally done when all the other methods fail.

Backtracking Method It is a popular technique that is widely used to debug small applications. It works by locating the first symptom of error and then tracing backward across the entire source code until the real cause of error is detected. However, the main drawback of this approach is that with increase in number of source code lines, the possible backward paths become too large to manage.