Unit-II

Planning the Computer Program: Concept of problem solving, Problem definition, Program design, Debugging, Types of errors in programming, Documentation.

Techniques of Problem Solving: Flowcharting, decision table, algorithms, Structured programming concepts, Programming methodologies viz. top-down and bottom-up programming.

Concept of Problem solving

A computer is a very powerful and versatile machine capable of performing a multitude of different tasks, yet it has no intelligence or thinking power. The intelligence Quotient (I.Q) of a computer is zero. A computer performs many tasks exactly in the same manner as it is told to do. This places responsibility on the user to instruct the computer in a correct and precise manner, so that the machine is able to perform the required job in a proper way. A wrong or ambiguous instruction may sometimes prove disastrous. In order to instruct a computer correctly, the user must have clear understanding of the problem to be solved. A part from this he should be able to develop a method, in the form of series of sequential steps, to solve it. Once the problem is well-defined and a method of solving it is developed, then instructing the computer to solve the problem becomes relatively easier task. Thus, before attempt to write a computer program to solve a given problem. It is necessary to formulate or define the problem in a precise manner. Once the problem is defined, the steps required to solve it, must be stated clearly in the required order.

It is the software developer who has to write down the solution to the problem in terms of simple operations which the computer can understand and execute. In order to solve a problem by the computer, one has to pass though certain stages or steps. They are

1. Understanding the problem

2. Analyzing the problem

3. Developing the solution

4. Coding and implementation.

1.Understanding the problem: Here we try to understand the problem to be solved in totally. Before going to the next stage or step, we should be absolutely sure about the objectives of the given problem.

Understanding the problem can be easily understood with the help of following Example:

Calculate the average grade for all students in a class.

You need to know:

- What input data/information is available ?
- What does it represent ?
- What format is it in ?
- Is anything missing ?
- Do I have everything that I need ?
- What output information am I trying to produce ?
- What do I want the result to look like … text, a picture, a graph … ?
- What am I going to have to compute ?

In our example, we well understand that the input is grades of students. But we need to understand the **format** of the grades. Each grade might be a **number from 0 to 100** or it may be a **letter grade from A+ to F**. If it is a number, the grade might be a **whole integer** like 73 or it may be a **real number** like 73.42. We need to understand the format of the grades in order to solve the problem. We also need to consider missing grades. What if we do not have the grade for **every** student (e.g., some were away during the test) ? Do we want to be able to **include** that person in our average (i.e., they received 0) or **ignore** them when computing the average ? We also need to understand what the output should be. Again, there is a formatting issue. Should we output a **whole** or **real number** or a **letter grade** ? Maybe we want to display a pie chart with the average grade. It is our choice. Finally, we should understand the kind of processing that needs to be performed on the data

2. Analyzing the problem: After understanding thoroughly the problem to be solved, we look different ways of solving the problem and evaluate each solution. The idea here is to search an appropriate solution to the problem under consideration. The end result of this stage is a broad overview of the sequence of operations that are to be carries out to solve the given problem. This step can be understood with following example

In our example, we are going to compute the average of the incoming grades. So, we need to know the model (or formula) for computing the average of a bunch of numbers. If there is no such "formula", we need to develop one. Often, however, the problem breaks down into simple computations that we can understand easily.

In order to come up with a model, we need to fully understand the information available to us. Assuming that the input data is a bunch of integers or real numbers **x1,x2,…,xn** representing a grade percentage, we can use the following computational model: **Average1 = (x1 + x2 + x3 + … + xn) / n** where the result will be a number from 0 to 100.That is very straight forward (assuming that we knew the formula for computing the average of a bunch of numbers). However, this approach will not work if the input data is a set of letter grades like **B-, C, A+, F, D-**, etc.. because we cannot perform addition and division on the letters. This problem solving step must figure out a way to produce an average from such letters. Thinking is required. After some thought, we may decide to assign an integer number to the incoming letters as follows: **A+** = 12 **B+** = 9 **C+** = 6 **D+** = 3 **F** = 0 **A** = 11 **B** = 8 **C** = 5 **D** = 2 **A-** = 10 **B-** = 7 **C-** = 4 **D-** = 1
If we assume that these newly assigned grade numbers are **y1,y2,…,yn**, then we can use the following computational model: **Average2 = (y1 + y2 + y3 + … + yn) / n** where the result will be a number from 0 to 12. As for the output, if we want it as a percentage, then we can use either **Average1** directly or use **(Average2 / 12)**, depending on the input that we had originally. If we wanted a letter grade as output, then we would have to use **(Average1/100*12)** or **(Average1*0.12)** or **Average2** and then map that to some kind of "lookup table" that allows us to look up a grade letter according to a number from 0 to 12.Do you understand this step in the problems solving process ? It is all about figuring out how you will make use of the available data to compute an answer.

3. Developing the solution: Here the overview of the sequence of operations that was the result of analysis stage is expanded to form a detailed step by step solution to the problem under consideration. This is done with the help of algorithm, pseudo code and flowcharts. We have

discussed these techniques in later part of this chapter. Here we have shown the use of algorithm for problem designing. An algorithm is a precise sequence of instructions for solving a problem.

Example :

Algorithm: DisplayGrades

1. set the sum of the grade values to 0.
2. load all grades x1 … xn from file.
3. repeat n times {
4. get grade xi
5. add xi to the sum
}
6. compute the average to be sum / n.
7. print the average.

4. Coding and implementation: The last stage of the problem solving is the conversion of the detailed sequence of operations into a language that the computer can understand. Here each step is converted to its equivalent instruction or instructions in the computer language that has been chosen for the implantation.

Now that we have a precise set of steps for solving the problem, most of the hard work has been done. We now have to transform the algorithm from step 3 into a set of instructions that can be understood by the computer. Writing a program is often called "***writing code***" or"***implementing an algorithm***". So the ***code*** (or ***source code***) is actually the program itself. Without much of an explanation, below is a program (written in processing) that implements our algorithm for finding the average of a set of grades. Notice that the code looks quite similar in structure, however, the processing code is less readable and seems somewhat more mathematical:

```
int sum = 0;
byte[] x = loadBytes("numbers");
for (int i=0; i<x.length; i++)
sum = sum + x[i];
int avg = sum / x.length;
print(avg);

/* code to find the average */
```

After coding, the source code is converted into machine code with the help of specialized software : interpreters and compilers

Program Design

To produce an effective computer program, it is necessary to find the logical solution of a problem and write every instruction in the correct sequence. However, the instruction sequence or the logic of the program may be very complex. Hence, before writing its code we must design the program. We can use many tools for the designing of a computer program. Some of the tools which can be used to design a program are: Algorithm, flowchart, decision table, pseudo code etc. During the designing process firstly an algorithm is created then it is represented with the help of flow charts, decision tables, pseudo code etc for easy understanding of the logic used for solving the given problem and finally it is converted into a computer program.

Algorithm: The term algorithm refers to the logic of the program. It is a step by step description of how to arrive at a solution to a given problem.

The characteristics of an Algorithm are as follow:

1. Each instruction should be precise and unambiguous.
2. No instruction should repeat infinitely.
3. It should give the correct result at the end.

Advantages of an algorithm

1. It is a step-by-step representation of a solution to a given problem ,which is very easy to understand
2. It has got a definite procedure.
3. It is independent of programming language.
4. It is easy to debug as every step got its own logical sequence.

Disadvantages of an algorithm

1. It is time consuming process.
2. It is difficult to show branch and looping in the algorithm.

Some Examples of algorithms:

Type 1 (Simple formula based)

Example 1 : Write an algorithm to print „Good Morning".

Step 1: Start
Step 2: Print „Good Morning"
Step 3: Stop

Example 2 : Write an algorithm to find area of a rectangle.

Step 1: Start
Step 2: Take length and breadth and store them as L and B?
Step 3: Multiply by L and B and store it in area
Step 4: Print area
Step 5: Stop

Type 2 (based on conditional statements)

Example 3 : Write an algorithm to check whether he is eligible to vote? (More than or equal to 18 years old).

Step 1: Start
Step 2: Take age of the user and store it in age
Step 3: Check age value, if age >= 18 then go to step 4 else step 5
Step 4: Print "Eligible to vote" and go to step 6
Step 5: Print "Not eligible to vote"
Step 6: Stop

Example 4 : Write an algorithm to check whether given number is +ve, -ve or zero.

Step 1: Start

Step 2: Take any number and store it in n.
Step 3: Check n value, if n > 0 then go to step 5 else go to step 4
Step 4: Check n value, if n < 0 then go to step 6 else go to step 7
Step 5: Print "Given number is +ve" and go to step 8
Step 6: Print "Given number is -ve" and go to step 8
Step 7: Print "Given number is zero"
Step 8: Stop

Type 3 (Based on Looping statements)

Example : Write an algorithm to print all natural numbers up to n".
Step 1: Start
Step 2: Take any number and store it in n.
Step 3: Store 1 in I
Step 4: Check I value, if I<=n then go to step 5 else go to step 8
Step 5: Print I
Step 6: Increment I value by 1
Step 5: Go to step 4
Step 8: Stop
In the above example, steps 4, 5, 6 and 7 are executed more than one time.

Flow chart

Flowchart is a program designing tool in which standard graphical symbols are used that illustrates the sequence of operations to be performed to solve a given problem. In other words, it is a pictorial representation of an algorithm or a process. It is also known as the Flow Diagram.

In a Flowchart the commands or statements of algorithm are shown as a special type of shapes. Different types of shapes are used for different types of statements, and within those figures, that statement is written briefly. These shapes are connected with each other with the help of arrows, which show the flow of operation or processing.

A program's flow chart is like a roadmap for a programmer that guides him/her how to go from a starting point to a final point while writing a program. While writing a flowchart, one is not concerned with the details of a programming language so one can concentrate upon the logic of the program.

The basic Symbol used in making flowcharts are as follows

1. Terminal :- The **Oval** is used to show the **beginning** and **end** of the **Flowchart**. It is **necessary** that Flowchart has a **Start** and **Stop** terminal. A flowchart should have only **one** Start and one Stop terminal.

Example-

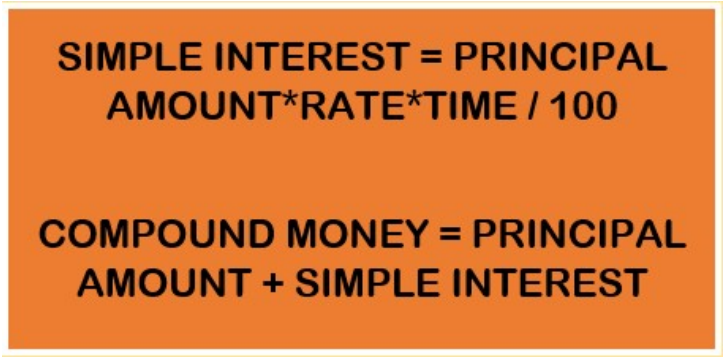
2. Input/Output :- Parallelogram shape is used to show the input and output operation in the algorithm or program.

For example, if there is a need to **get input** from the keyboard, disk, tape, or any other device in the program, then use the **Read** or **Input** inside this box. If you want to send output to magnetic tape, magnetic disk, VDU, in the program, then write the **Output** or **Write** within the Parallelogram.

Example-

3. Processing :- Rectangular shape is used to show the processing section of an algorithm. In it, the formulas and instructions are written, through which the data is being processed. When processing has more than one instruction, then they are written in the same box. The instructions that are written before, are executed first and which are written later, it is executed later.

Example-

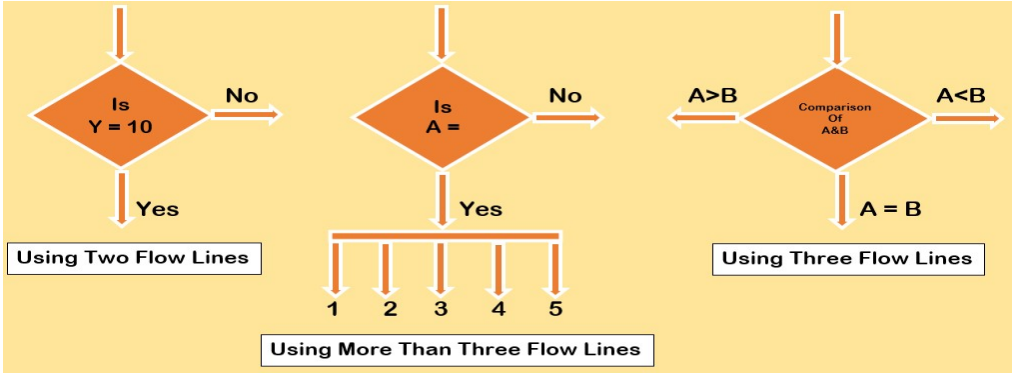


As shown in the box two statements are written simultaneously. In these statements, the simple interest will be calculated first and the compound money calculated later.

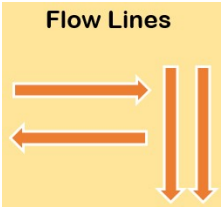
4. <u>Decision</u> :-The **Rhombus/Diamond is used to show the decision section**. It is used in the place where we want to perform certain task only when a particular condition holds true or false.

In this case for each possible situation arising, different flow-lines are used to represent the flow of execution.

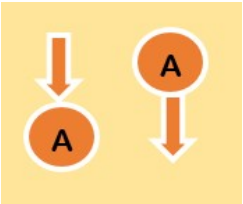
Example-



5. Flow-lines :- The arrow shape is used to show the flow lines, these are used to connect different shapes such as **input output box, terminal, decision box,** and **connector** etc. It is also used to show the **flow** of the execution of the program. The flow of execution of the program is determined by the direction of arrow mark on the flow line.



6. Connectors :- The **spherical / Circle** shapes are used to represent the connectors. These are used when the flowchart is so **large** that it cannot come to **one page** or the flow line of flowchart starts cutting each other. These shapes are , on which letters or numerals are written, so that they can know their links in between. Similar letters or numbers written in a program, display the continuous flow of the connector program.



Advantages / Benefits of Flowchart-

Simple explanation of program :- Flowchart is a pictorial description of any program. The program created by it can be easily understood.

Effective analysis :- Through the flowcharts, the program is divided into smaller parts, that making effective analysis of the program.

Designing of program :- Using a flowchart, your program can be designed correctly and easily .

Easily debugging :- Flow chart is very helpful in detecting, locating and removing the mistake in program.

Documentation : A flow chart provide documentation support which is very helpful in designing a good program.

Disadvantages / Limitations of Flowchart -

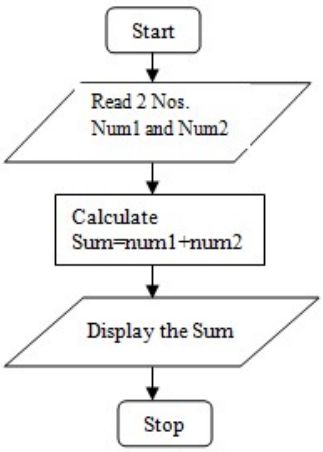
Time consuming :- Creating a flow chart takes more time. The flow chart is made especially for complex problems, which take more time.

Difficult to make changes :- If there are some complex modification in the flow chart, then the entire flowchart has to be recreated.

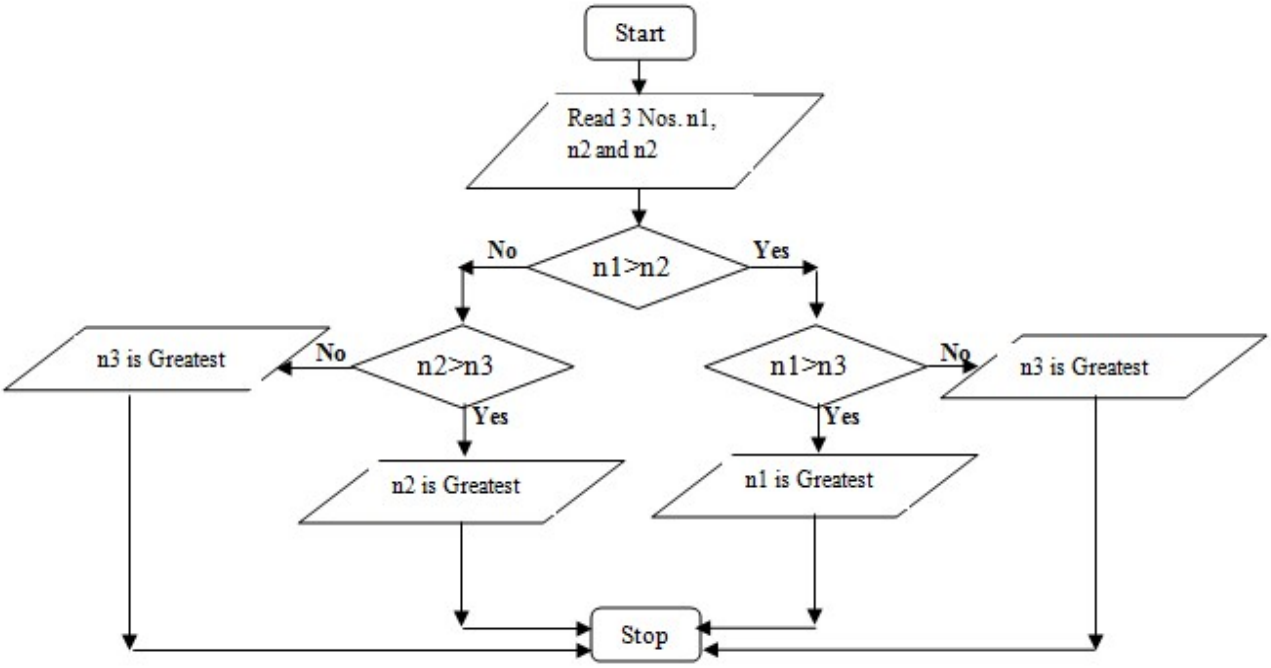
No standard :- Program flowcharts, although easy to understand, but can not be written as a standard, like others language. Each programmer resolves the problem in its own way and creates a flow chart accordingly.

Some Example of Flowchart Program-

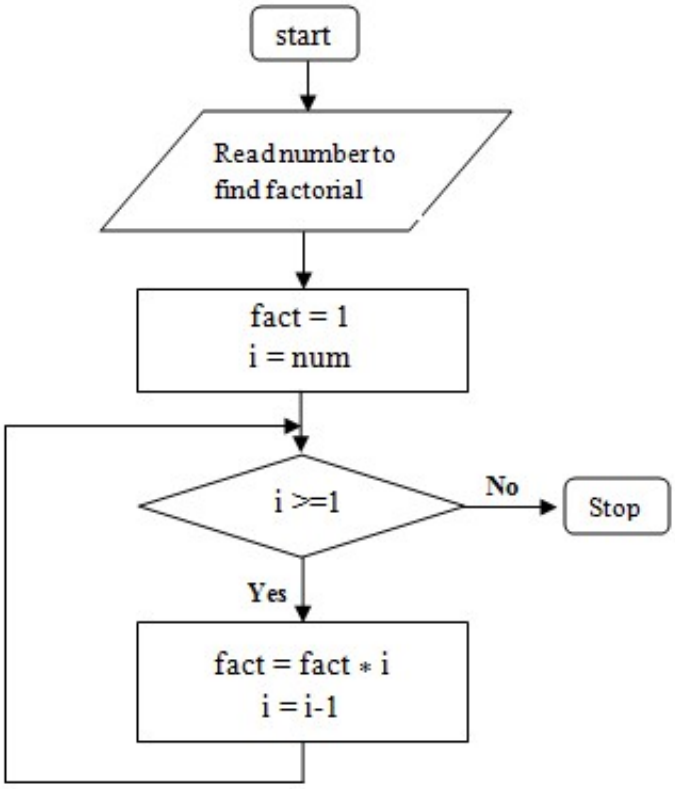
Example 1 : Create a flowchart to find the sum of two numbers



Example 2 : Create a flow chart to find the largest no. among 3 nos.



Example 3: Create a flow chart to find factorial on a number.



Decision Table

Decision table, also known as the logic table is a powerful graphical tool used to analyze and represent complex proceeding logic. It uses a tabular format to represent the various actions to be taken for various combination of conditions. The decision tables are extremely useful when a program involve a large number of conditions. Decision table have a standardized format and consists of four sections as shown below :

	Decision Rules				
	Rule 1	**Rule 2**	**Rule 3**	**……**	**Rule N**
Condition Stub 1 **.** **.** **.**	**Value for Stub 1** **(Y/N)**	**Value for Stub 1** **(Y/N)**	**Value for Stub 1** **(Y/N)**	**……**	**Value for Stub 1** **(Y/N)**
Condition Stub N	**Value for Stub N** **(Y/N)**	**Value for Stub N** **(Y/N)**	**Value for Stub N** **(Y/N)**	**……**	**Value for Stub N** **(Y/N)**
Action Stub 1 **.** **.** **.** **.**	**Action Option 1**	**Action Option 1**	**Action Option 1**	**…….**	**Action Option 1**
Action Stub N	**Action Option N**	**Action Option N**	**Action Option N**		**Action Option N**

Condition Stub : This section lists all the possible conditions which could apply to a particular problem.

Condition entry : This section of the decision table contains the different combination of the conditions given in the condition stub , each combination is referred as a "Rule" its value is generally represented as Y/N.

Action Stub : This section lists the actions resulted from a given set of conditions.

Action Entry : This section shows the actions to be taken for each combination of conditions.

Advantages of Decision Table :

1. Easier to draw and modify in comparison to algorithm and flowchart.
2. Decision rules are clearly structured.
3. It helps to better understand the decision logic of a problem.

Disadvantage

1. It is used only for the representation of actions performed based upon the various conditions. So, it is difficult to write a complete program on the basis of decision table.
2. Decision tables are larger in size.

Example : Create a decision table to give the rate of interest based upon the following conditions

1. Principle >=50000 Time> 5 year **Rate of Interest=8%**
2. Principle >=50000 Time = 3 to 5 year **Rate of Interest= 6.5 %**
3. Principle >=50000 Time = less than 3 years **Rate of Interest = 5%**
4. Principle >=25000 but less than 50000 Time> 5 year **Rate of Interest = 7%**
5. Principle >=25000 but less than 50000 Time = 3 to 5 year **Rate of Interest = 5.5 %**
6. Principle >=25000 but less than 50000 Time = less than 3 years **Rate of Interest = 4%**
7. In all other cases **Rate of Interest= 3.5 %**

Condition Stub	Rule1	Rule2	Rule3	Rule4	Rule5	Rule6	Rule7
Principal >=50000	Y	Y	Y	N	N	N	N
Principle >=25000	N	N	N	Y	Y	Y	N
Principle < 25000	N	N	N	N	N	N	Y
Time>5 Years	Y	N	N	Y	N	N	Y
Time 3 to 5 Years	N	Y	N	N	Y	N	Y
Time less than 3 years	N	N	Y	N	N	Y	Y
Rate of Interest=8%	√						
Rate of Interest=7%				√			
Rate of Interest=6.5%		√					
Rate of Interest=6%							
Rate of Interest=5.5%					√		
Rate of Interest=5%			√				
Rate of Interest=4%						√	
Rate of Interest=3.5%							√

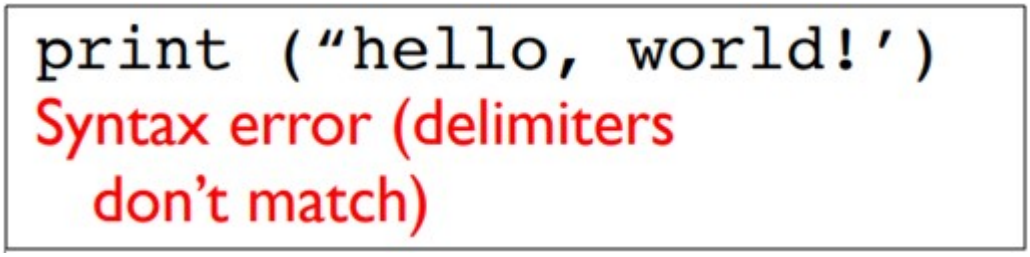
Debugging and Error Types

De-bugging a program is the process of finding and resolving errors. Debugging is an essential part of programming. Even for an experienced programmer most codes will not work in the first attempt due to syntax errors, mathematical errors and logical flaws. These errors or bugs have to be corrected before your program would yield desired results.

There are 3 kinds of errors (bugs) in the code:

1. Syntax errors: The code does not follow the rules of the language; for example, a single quote is used where a double quote is needed; a colon is missing; a keyword is used as a variable name. ¨

Example :



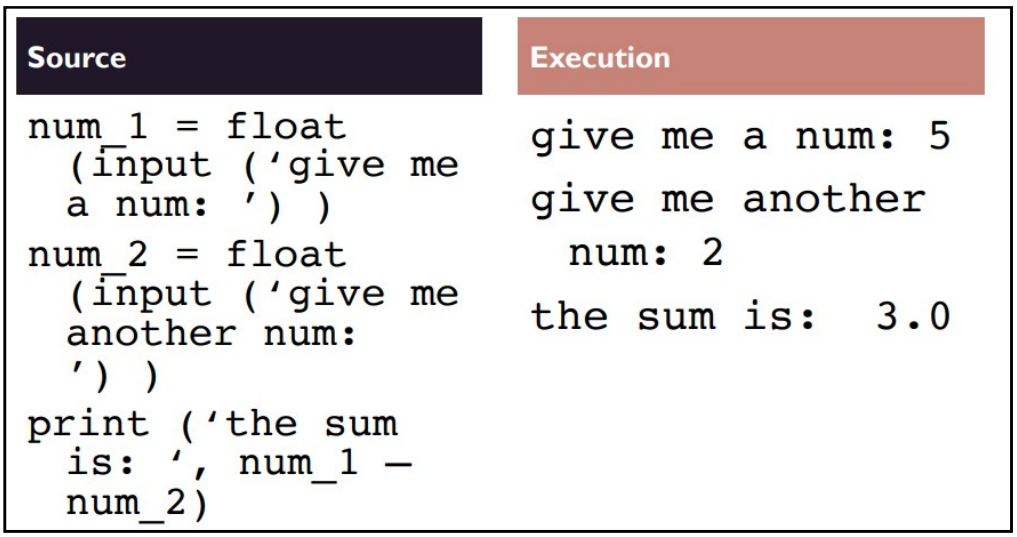
2. Runtime errors: In this case, your code is fine but the program does not run as expected (it "crashes"). For example, if your program is meant to divide two numbers, but does not test for a zero divisor, a run-time error would occur when the program attempts to divide by zero.

Example

```
>>> print(5/0)
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    print(5/0)
ZeroDivisionError: division by zero
```

3. Logic errors: These can be the hardest to find. In this case, the program is correct from a syntax perspective; and it runs; but the result is unanticipated or outright wrong. For example, if your program prints "5+2 = 3".

Example



Basic Debugging Techniques

Although there is no precise procedure for fixing all bugs, there are a number of useful strategies that can reduce the debugging effort. Below are several useful strategies to help with this.

- **Incremental program development.** One of the most effective ways to localize errors is to develop the program incrementally, and test it often, after adding each piece of code. It is highly likely that if there is an error, it occurs in the last piece of code that you wrote. With incremental program development, the last portion of code is small; the search for bugs is therefore limited to small code fragments.
- **Use debuggers.** Debugger is a tool that help to identify and remove bugs. Through this tool we can set breakpoints in the program to check the results of a particular section of the program.
- **Backtracking.** One option is to start from the point where to problem occurred and go back through the code to see how that might have happened.

Documentation

All large software development projects, irrespective of application, generate a large amount of associated documentation. The documents associated with a software project and the system being developed have a number of associated requirements:

1. They should act as a communication medium between members of the development team.

2. They should be a system information repository to be used by maintenance engineers.

3. They should provide information for management to help them plan, budget and schedule the software development process.

4. Some of the documents should tell users how to use and administer the system.

Generally we can divide the documentation produced into two classes:

1. Process documentation : These documents record the process of development and maintenance. Plans, schedules, process quality documents and organizational and project standards are process documentation.

Process documentation falls into a number of categories:

- Plans, estimates and schedules . These are documents produced by managers which are used to predict and to control the software process.
- Reports : These are documents which report how resources were used during the process of development.
- Standards : These are documents which set out how the process is to be implemented. These may be developed from organizational, national or international standards
- Working papers : These are often the principal technical communication documents in a project. They record the ideas and thoughts of the engineers working on the project, are interim versions of product documentation, describe implementation strategies and set out problems which have been identified.
- Memos and electronic mail messages : These record the details of everyday communications between managers and development engineers.

2. Product documentation: This documentation describes the product that is being developed. Product documentation includes user documentation which tells users how to use the software product and system documentation which is principally intended for maintenance engineers. The main documents included in product documentation are : installation guide, reference manual, system administration guide, functional description, validation document etc.

Programming Methodology

When programs are developed to solve real-life problems like inventory management, payroll processing, student admissions, examination result processing, etc. they tend to be huge and complex. The approach to planning for software development and controlling the development process is called programming methodology.

Types of Programming Methodologies

There are many types of programming methodologies prevalent among software developers, two of the most common programming methodologies are as follow :

Procedural Programming : Problem is broken down into procedures, or blocks of code that perform one task each. All procedures taken together form the whole program. It is suitable only for small programs that have low level of complexity. Example: For a calculator program that does addition, subtraction, multiplication, division, square root and comparison, each of these operations can be developed as separate procedures. In the main program each procedure would be invoked on the basis of user's choice. This methodology is suitable for solving small problems.

Object-oriented Programming : Here the solution revolves around entities or objects that are part of problem. The solution deals with how to store data related to the entities, how the entities behave and how they interact with each other. Example: If we have to develop a payroll

management system, we will have entities like employees, salary structure, leave rules, etc. around which the solution must be built.

Software developers may choose one or a combination of more than one of these methodologies to develop a software. Note that in each of the methodologies discussed, problem has to be broken down into smaller units. To do this, developers use any of the following two approaches:

Top-down approach and Bottom-up approach

Top-down or Modular Approach : In this approach the problem is broken down into smaller units, which may be further broken down into even smaller units. Each unit is called a module. Each module is a self-sufficient unit that has everything necessary to perform its task. The following

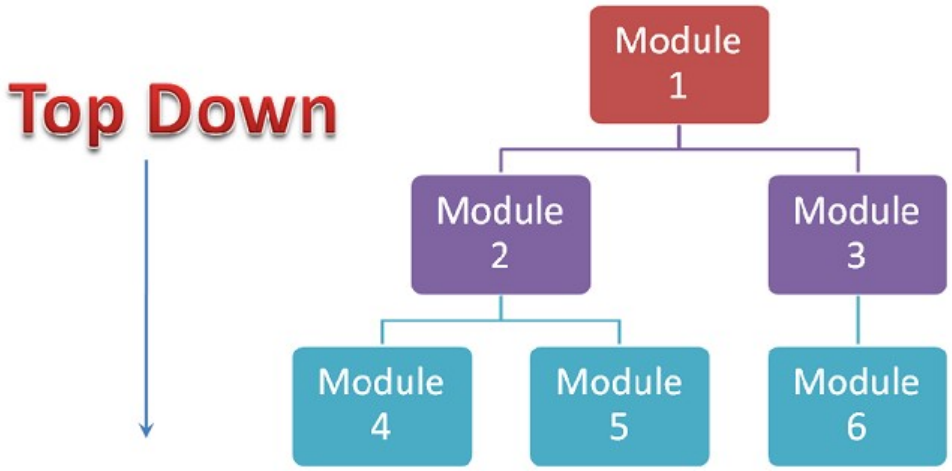


illustration shows an example of how you can follow modular approach to divide the problem into different modules while creating a program.

Advantages of top-down approach:

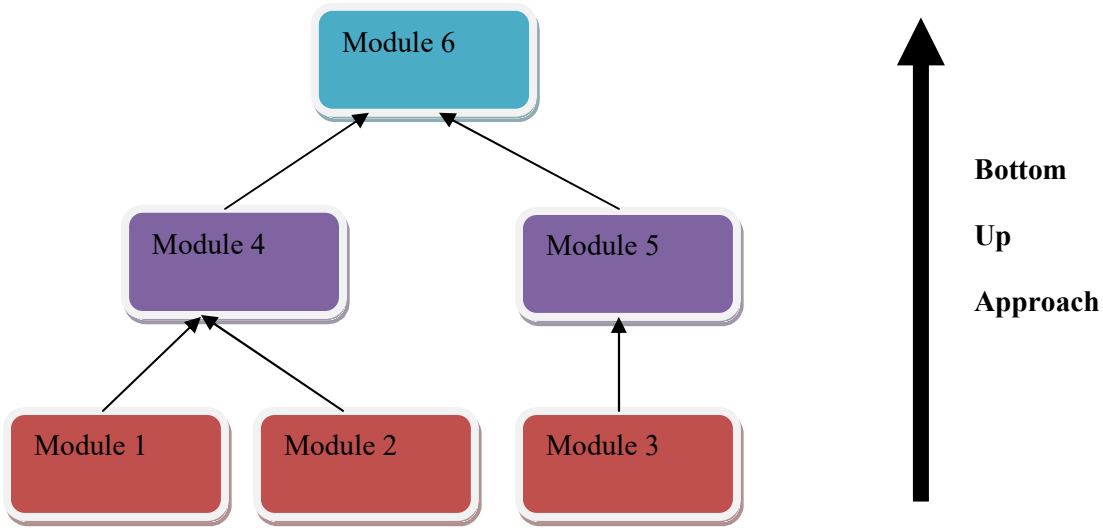
1. Using this approach, the problem solving process of big problem becomes easy.
2. This approach utilize the resources in a proper manner according to the project.
3. Testing and debugging is easier and efficient.
4. In this approach, project implementation is smoother and shorter.
5. This approach is good for detecting and correcting time delays.

Disadvantages of Top- Down approach

1. This approach is not suitable for solving highly complex problems.
2. This approach is less flexible (updation in the program) in comparison to bottom up approach.

Bottom-up Approach

In bottom-up approach, system design starts with the lowest level of components, which are then interconnected to get higher level components. This process continues till a hierarchy of all system components is generated. However, in real-life scenario it is very difficult to know all lowest level components at the beginning.



Advantages of Bottom up Approach

1. Flexible: This approach is very flexible. Addition of new features and modification of features is easier.
2. Reusability: As the parts of the program are created independent, therefore the possibility of reusing them increases.
3. Reliability: This approach is more reliable, because in this case each part of the program is independently created and tested first then it is integrated.
4. **Lower Cost:** In this approach the components are reused therefore the cost of the software development decreases.

Disadvantages of Bottom Up Approach

1. It is difficult to evaluate the progress of the project in this approach.

2. The idea about the final product is obtained at end only.

Structured Programming

Structured programming is concerned with the structures used in a computer program. Generally, structures of computer program comprise decisions, sequences, and loops. The decision structures are used for conditional execution of statements (for example, 'if statement'). The sequence structures are used for the sequentially executed statements. The loop structures are used for performing some repetitive tasks in the program.

Structured programming forces a logical structure in the program to be written in an efficient and understandable manner. The purpose of structured programming is to make the software code easy to modify when required. Primarily, the structured programming focuses on reducing the following statements from the program.

1. 'GOTO' statements.
2. 'Break' or 'Continue' outside the loops.
3. Multiple exit points to a function, procedure, or subroutine. For example, multiple 'Return' statements should not be used.
4. Multiple entry points to a function, procedure, or a subroutine.

Structured programming generally makes use of top-down design because program structure is divided into separate subsections. A defined function or set of similar functions is kept separately. Due to this separation of functions, they are easily loaded in the memory. In addition, these functions can be reused in one or more programs. Each module is tested individually. After testing, they are integrated with other modules to achieve an overall program structure. Note that a key characteristic of a structured statement is the presence of single entry and single exit point. This characteristic implies that during execution, a structured statement starts from one defined point and terminates at another defined point.

Advantages of structured programming

The following are the different advantages of structured programming

1. It is user friendly and easy to understand.
2. They require less time to write.
3. They are easier to maintain.
4. Program written in a higher level language can be translated into many machine languages and therefore can run on any computer for which there exists an appropriate translator.

Disadvantages of structured programming

1. Modification in structured programs is difficult.
2. This approach is not suitable for solving complex problems