****

**Basic Design Structures**

Lesson : Flowcharts, Algorithms & Pseudocodes



SO WHAT DO YOU NEED FOR THIS MODULE?

You are expected to have understood the various techniques used in software development methodologies.

So let us get started 

**INTRODUCTION**

Regardless of the area of study, computer science is all about solving problems with computers. The problems that we want to solve can come from any real-world problem or perhaps even from the abstract world. We need to have a standard systematic approach to solving problems.

So what is problem solving? Problem Solving is the sequential process of analyzing information related to a given situation and generating appropriate response options. In our daily lives, we make decisions geared towards solving problems that we face. Every decision you make determine your path. Let’s just take note of some random decisions we make.

* Must I go to university of must I join a coding bootcamp?
* Must I work hard at college or not?
* Must I go home during lecture time or not?

The answer to the above is part of problem solving. As you decide on which action to take, you are already making decisions. Similarly, when you start working as a developer, you are going to be presented with challenges in which you will be required to make decisions starting with choosing the programming language. Problem solving is a critical skill for success in business—in fact it’s often what you are hired and paid to do. This article explains the five problem solving steps and provides strategies on how to execute each one. There are common steps followed when solving a problem and these are presented in the subsequent section

**COMMON STEPS IN PROBLEM SOLVING**

**Step 1:** Understand the problem

**Step 2:** Devise a plan to obtain possible solutions.

**Step 3:** Evaluate the solutions and choose the best one.

**Step 4:** Test the solution, resolve errors and apply the changes

**Step 5:** Implement the solution.

**Understand the problem-**The first step to solving a problem involves understanding the problem and defining it. The best way to do this to break it into smaller manageable tasks. The first part involves writing down the problem and analyzing it. A good place to start your problem analysis is to investigate how the problem occurred. This is achieved through asking a lot of questions.

**Devise a plan to obtain possible solutions**. - Once you have completed your analysis of the problem you have to devise a solution. There are many techniques to help you to solve this. Such thinking processes include: Association (cause-effect-thinking). Analogy (comparisons), Brainstorming (group discussion), Intuition (knowledge and experience) Analytical thinking (dividing problem into smaller sub tasks).

**Evaluate the solutions and choose the best one** - Consider all possible solutions and identify weaknesses and strength of each problem. After eliminating all other solutions, go back to step 1 and see if your problem still exists. It is at this stage where you as a programmer you have to decide the modelling technique which you will use. Some of the techniques include flowchart, pseudocode, algorithms, Input Processing and Output tables and many others.

**Test the solution, resolve errors and apply the changes -Test** the solution and test it. Ideal is to start with prototypes.

**Implement the Solution-**This check for design flaws. A thorough test has to be implemented.

Fundamental to design is identifying input, process(es) or output. Consider a simple example of how the input/process/output works on a simple problem: Example: Calculate the average grade for all students in a class.

1. Input: get all the grades … perhaps by typing them in via the keyboard or by reading them from a USB flash drive or hard disk.

2. Process: add them all up and compute the average grade.

3. Output: output the answer to either the monitor, to the printer, to the USB flash drive or hard disk … or a combination of any of these devices.

As you can see, the problem is easily solved by simply getting the input, computing something and producing the output. Let us now examine the 6 steps to problems solving within the context of the above example.

**COMMON BASIC TECHNIQUES**

* Flowcharts
* Algorithms
* Pseudocodes
* Storyboards (mainly used in web development)
* Wireframes e.t.c

So let us start by discussing flowcharts.

**WHAT IS A FLOWCHART?**

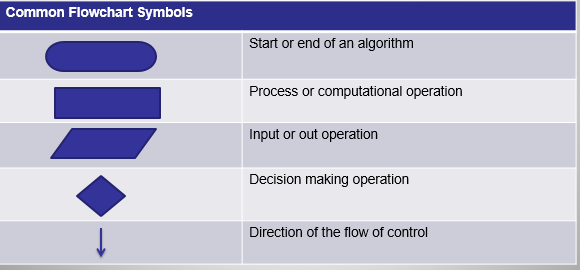
A flowchart is a method used to analyse a problem and then break it down into individual processes by representing the steps as boxes that are connected with arrows. This diagrammatic documentation gives a step by step solution to a problem. Frank Gilberth introduced flowcharts in 1921, and they were called “Process Flow Charts” at the beginning. Different flowchart symbols are used for different aspects of the process and these are discussed later.

Flowcharts can be modelled from the perspective of different user groups (such as managers, system analysts and clerks), and that there are four general types of flowcharts:

* **Document flowcharts**- showing controls over a document-flow through a system
* **Data flowcharts** - showing controls over a data-flow in a system
* **System flowcharts** - showing controls at a physical or resource level
* **Program flowchart** - showing the controls in a program within a system
* **Cross functional** flowchart - flow charts, also called **deployment** flow charts, are used when the programmer wants to clearly show the relationship to who or what process is responsible for each step of the process.
* **swim lane** **flowchart** - similar to the cross functional and deployment flow charts, uses the concept of a large pool divided into lanes for swimmers to stay on track.
* **Specification and Description Language Diagram (SDL)** is useful in real-time, stimulus response systems often found in telecommunications.

**FLOWCHART SYMBOLS.**

Figure 1 shows the common symbols that are used in depicting the problem in software development?



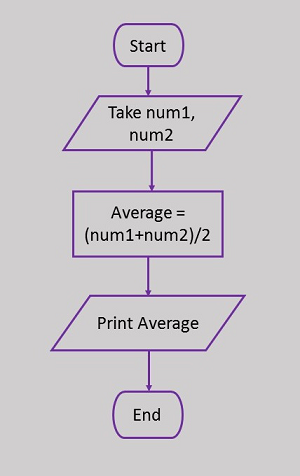
**Figure 1:Basic flowchart symbols**

**GUIDELINES FOR DEVELOPING FLOWCHARTS**

* Guidelines for Developing Flowcharts
* These are some points to keep in mind while developing a flowchart −
* Flowchart can have only one start and one stop symbol
* On-page connectors are referenced using numbers
* Off-page connectors are referenced using alphabets
* General flow of processes is top to bottom or left to right
* Arrows should not cross each other

**Try this out**

Let us look at the following diagram and identify the symbols used here.



Consider the following scenario and see the resulting flowchart.

You are required to write a program to capture two integer numbers which are not equal through the keyboard and compare them to display which one is greater. Design a flowchart to depict how the scenario.

**Hint:** As a developer, the problem solving techniques explained in this lecture will apply. First, you must understand the problem perhaps by asking yourself: what is it that I am expected to do. What are the inputs, processes and outputs? How are you going to implement the design structures?

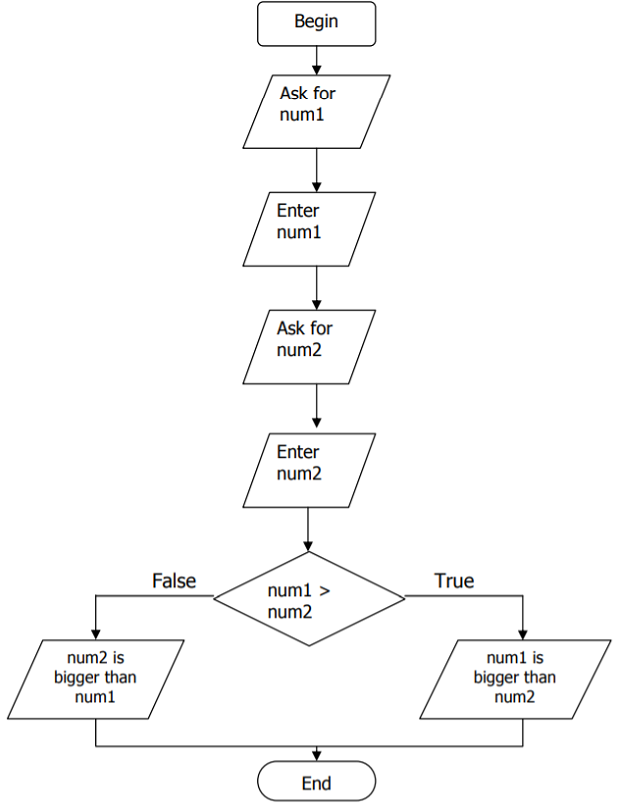
So let’s start by identifying the:

**INPUTS** - Num1, Num2

**PROCESSES** - Compare if Num1 is greater than Num2: is for

**OUTPUT**- Display Num1 is greater If Not Display Num2 is greater

So let us have a look at the resulting diagram.



**Figure 2:Flowchart to display greater number**

**WHY FLOWCHARTS FOR PROGRAMMING?**

We can see that flow charts use very few words to show the flow of data and how the data and processes are related. They mostly use arrows and symbols. Why would computer programmers want to use a picture to represent a program they are creating? Here are some of the advantages for using flow charts:

* Flow charts can be used to quickly communicate the ideas or plans that one programmer envisions to other people who will be involved in the process.
* Flow charts aid in the analysis of the process to make sure nothing is left out and that all possible inputs, processes, and outputs have been accounted for.
* Flow charts help programmers develop the most efficient coding because they can clearly see where the data is going to end up.
* Flow charts help programmers figure out where a potential problem area is and helps them with debugging or cleaning up code that is not working.

**ALGORITHMS**

In programming, an algorithm is a set of well-defined instructions in sequence to solve the problem. An algorithm is used to provide a solution to a particular problem in form of well-defined steps. Whenever you use a computer to solve a particular problem, the steps which lead to the solution should be properly communicated to the computer. While executing an algorithm on a computer, several operations such as additions and subtractions are combined to perform more complex mathematical operations. Algorithms can be expressed using natural language.

**WHAT IS AN ALGORITHM IN A PROGRAMMING PERSPECTIVE?**

You see, computers basically do a lot of math which means it has a lot of problems to solve. That’s exactly why algorithms [form the heart of computer science](https://www.educba.com/career-in-computer-science/). A computer algorithm is a computational procedure that takes in a set of finite input and transforms it into output by applying some math & logic.

An algorithm in programming will have several steps as follows:

1. **Problem definition** – What is to be done?
2. **Data collection** – What do we have to solve the problem? Or inputs.
3. **Data processing** – Understanding what we have or transforming them into a usable form.
4. **Logical approach** – Employing the collected & created data against logic to solve.
5. **Solution** – Present the solution in the way you want in a GUI or a terminal or a diagram or a chart.

One important aspect to know is that the algorithms are not strictly bound to any programming language. They are generic solutions as such. There are so many kinds of algorithms as shown in Table 1.

**Table 1: Types of Algorithms**

|  |  |
| --- | --- |
| Type of algorithm | Description |
| Brute force algorithms | Which are straight forward trial and error approach of solving problems? Just like you do repeat addition to find the result of a multiplication problem. |
| [Greedy algorithms](https://www.educba.com/what-is-a-greedy-algorithm/) | Which follow a problem-solving heuristic to reach the next best state to find the final best state as the result. Just like you find the less steep area which climbing a mountain for ease. |
| Divide and Conquer algorithms | Which break the problem into small sub problems and then combine the result of each sub problem to get the final result. |
| Dynamic programming | An approach which is same as divide and conquers but divides the problem into sub problems such that their results are reusable for other sub problems. |

**QUALITIES OF A GOOD ALGORITHM**

1. Input and output should be defined **precisely**.
2. Each steps in algorithm should be **clear and unambiguous**.
3. Algorithm should be most effective among many different ways to solve a problem.
4. An algorithm shouldn't have computer code. Instead, the algorithm should be written in such a way that, it can be used in similar programming languages **(generality).**in

Despite if someday we have a processor that is incredibly fast and a memory that is continuous, we still have to study algorithm, design them so as to see if the solution terminates and does so with a correct result. May it be commercial applications, scientific computing, engineering, operational research or artificial intelligence, in each field articulating problems, figuring out efficient algorithms to solve and data structures to deal with will remain inevitable forever.

Let us have some examples of algorithms

**PROBLEM**

Determine the weekly wage of an employee if the hours he has worked, and the hourly rate are entered by the user.

**SOLUTION**

1. Ask: How many hours did the employee work?

2. Enter number of hours

3. Ask: What is the hourly rate of pay?

4. Enter hourly rate of pay

5. Wage = number of hours X hourly rate of pay

6. Show the wage on the screen of the computer.

**Notes:**

• Every algorithm has a descriptive name. This algorithm is called CalcWage.

• The last statement (step) is always end.

• ~ indicates that a comment that explains the code, will follow.

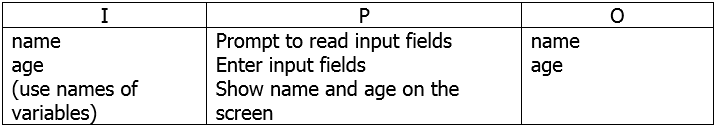
• Display means that a message or the contents of a variable is displayed on the screen. (Will be discussed later) • Enter indicates that data must be entered by using the keyboard.

• The calculation is done by using an equation. In this algorithm wage will accept a new value i.e. the product of the hours and the hourly rate of pay.

**Problem 2**

Enter the distance between two trees in kilometers. Display the kilometers and then calculate and display the distance in meters as well as the distance in centimeters.

To do better and more comprehensive planning to enable the programmer to write a complete algorithm without errors, an IPO-chart (Input, Processing, Output) is completed. The IPO chart includes an overview of the processing steps.



**SOLUTION**

1. Enter name, age
2. Display name in a new line
3. Display age in a new line

**Try this out**

You have been requested to write a program to check whether three given integer values are in the range 20..50 inclusive. Return true if 1 or more of them are in the said range otherwise return false. Design an algorithm to solve the problem.

**SUMMARY -ALGORITHM**

Despite if someday we have a processor that is incredibly fast and a memory that is continuous, we still have to study algorithm, design them so as to see if the solution terminates and does so with a correct result. May it be commercial applications, scientific computing, engineering, operational research or artificial intelligence, in each field articulating problems, figuring out efficient algorithms to solve and data structures to deal with will remain inevitable forever.

**PSEUDOCODES**

There are different ways of planning to solve a computer related problem. The solution for a problem can also be planned with flowcharts or algorithms as explained earlier in this handout. For now we are going to introduce pseudocode as another alternative.

Pseudocode is a specific way of writing the steps of the algorithm in English using specific words. Pseudocode gives a high-level description of an algorithm without the ambiguity associated with plain text but also without the need to know the syntax of a particular programming language. The running time can be estimated in a more general manner by using Pseudocode to represent the algorithm as a set of fundamental operations which can then be counted. So, what is the difference between a pseudocode and an algorithm.

Generally, the word "algorithm" can be used to describe any high level task in computer science. On the other hand, pseudocode is an informal and (often rudimentary) human readable description of an algorithm leaving many granular details of it. Writing a pseudocode has no restriction of styles and its only objective is to describe the high level steps of algorithm in a much realistic manner in natural language

The rules of Pseudocode are reasonably straightforward. All statements showing "dependency" are to be indented. These include while, do, for, if, switch. Examples below will illustrate this notion.

**Problem 1**

Write a program to determine whether a student has failed or passed. The program must display passed if mark is 60% or above else it displays failed.

Solution:

1.. *If student's grade is greater than or equal to 60*

*Print "passed"*

*else*

*Print "failed"*

**Problem 2**

Write a program to calculate an employee’s wage. If an employee has worked more than 45 hours during the past week, a bonus of R300 must be added to his wage. Design a pseudocode for the scenario.

*~ test if the employee worked overtime*

*if hours > 45 then*

*wage = wage + 300 ~ executed only if the condition is true*

*endif*

let us try to explain the pseudocode for problem 3

Explanation of the above 3 lines:

if hours > 45 then

This statement tests if the hours worked are more than 45

wage = wage + 300

If the test is true (the employee has worked more than 45 hours), the previous wage is replaced by an increased wage of R300 more

endif

This line ends the if-structure and the execution of statements will continue after the endif-statement regardless of the outcome of the if test

In the case where the employee has not worked for more than 45 hours, the execution of statements will continue after the endif . It would have skipped the statement to increase the wage.

**CONCLUSION - (ALGORITHMS, FLOWCHARTS AND PSEUDOCODES)**

An algorithm is a process or set of rules to be followed in calculations or other problem-solving operations, especially by a computer. A flowchart is a type of diagram that represents a workflow or process. A flowchart can also be defined as a diagrammatic representation of an algorithm, a step-by-step approach to solving a task. Pseudocode is an informal high-level description of the operating principle of a computer program or other algorithm. It uses the structural conventions of a normal programming language, but is intended for human reading rather than machine reading. Pseudo means false so pseudocode is false code, not real code.