# Introduction to Technical Programming

# Topic 2 Problem solving in computer programming

2.1 **Problem solving process and concepts**

Content:

* Problem solving
* Application of problem-solving constructs
* Developing solutions

*Learning Outcomes:*

*Students should be able to:*

2.1Define the term problem solving

2.1.2 Define the term computational thinking

2.1.3 Describe the phases of the PLDC (Program Development Life Cycle)

2.1.4 Describe the purpose of problem solving leading to solutions

2.1.5 Explain and apply various problem-solving steps. Polya, G., 1957) (Range:Understand the problem (task/problem description or scenario/user stories) State in own words Clarity on what needs to be done

What is known or given?

What is missing or needed?

Devise a plan/algorithm (storyboard – visual or textual)

Look for patterns

Look at related problems, known solutions

Examine simpler or special cases

Make a table, create diagram, use guess and check, work backwards, identify sub-goal

Carry out the plan/implement the algorithm (write the code)

Look back/test (see if it works)

Check results against original problem. Does it make sense? Is there another solution?)

2.1.6 Use appropriate tools and techniques to present a solution. Range:

User stories (written by the client and provide the requirements)

Noun-verb analysis of user stories

List of nouns provides identification of objects and state

List of verbs provides identification of behaviour

Acceptance tests (does the program meet the requirements?)

2.2 **Construct an algorithm and present a solution to a given problem**

Content:

* Problem solving
* Algorithm design
* Flowcharts

*Learning Outcomes:*

*Students should be able to:*

2.2.1 Define the term algorithm and its purpose in the problem-solving process. (Range: Basic concepts of an algorithm. What is an algorithm? Develop a clear understanding of the problem presented.)

2.2.2 Implement and understand the basic algorithmic constructs used to create a **flowchart.** Range: Input, Output, Processing and Calculations, Selection Iteration

2.2.3 Create a flowchart to present a particular algorithm and its associated tasks

2.2.4 Interpret a basic flow chart and describe its intended operation / function

# Topic 2 Problem solving in computer programming

After you have completed this module, you should be able to :

* Define the term problem solving
* Define the term computational thinking
* Describe the phases of the PLDC (Program Development Life Cycle)
* Describe the purpose of problem solving leading to solutions
* Explain and apply various problem-solving steps.
* State in own words Clarity on what needs to be done
* What is known or given?
* What is missing or needed?
* Devise a plan/algorithm (storyboard – visual or
* textual)
* Look for patterns
* Look at related problems, known solutions
* Examine simpler or special cases
* Make a table, create diagram, use guess and check,
* work backwards, identify sub-goal
* Carry out the plan/implement the algorithm (write
* the code)
* Look back/test (see if it works)
* Check results against original problem. Does it make sense? Is there another solution?)
* Use appropriate tools and techniques to present a solution. Range:
* User stories (written by the client and provide the requirements)
* Noun-verb analysis of user stories
* List of nouns provides identification of objects and state
* List of verbs provides identification of behaviour
* Acceptance tests (does the program meet the
* requirements?)
* Define the term algorithm and its purpose in the problem-solving process. (Range: Basic concepts of an algorithm.
* What is an algorithm? Develop a clear understanding of the problem presented.)
* Implement and understand the basic algorithmic constructs used to create a flowchart. Range: Input, Output, Processing and Calculations, Selection, Iteration
* Create a flowchart to present a particular algorithm and its associated tasks
* Interpret a basic flow chart and describe its intended operation / function

**Introduction**

Solving problems is computer science's primary objective. Any real-world issue, or perhaps even those from the abstract world, can give rise to the issues we are trying to solve. To solve problems, we must use a standardized, systematic approach. Programmers need to first understand how people approach problems, then they need to understand how to turn this "algorithm" into something that a machine can accomplish, and finally they need to understand how to represent the solution in the selected code of their choice. Problem solving is a key attribute a programmer must have. Understanding what needs to be solved in all instances is fundamental to the process of solving the problem. This chapter will highlight the different problem solving techniques applied in solving computer problems.

## 2.1 Problem solving process and concepts

### 2.1.1 Define the term problem solving

**VOCABULARY**

Problem solving is the sequential process of analysing information related to a given situation and generating appropriate response options.

Using a computer, issues are resolved by getting some sort of user input (such as keyboard/mouse data or gaming control motions), processing the input, and creating some sort of output (e.g., images, text, sound). Hard disks or network devices may be used to store and transmit data occasionally.

### 2.1.2 Define the term computational thinking

**VOCABULARY**

Computational thinking is It is a set of skills and practices that allow us to solve complex problems, learn topics across many disciplines, and fully participate in our digital world.

We can come up with potential answers to complex problems by using computational thinking. Our solutions can then be communicated to machines, people, or both in an understandable manner.

The four cornerstones of computational thinking

* **decomposition**
* **pattern recognition**
* **abstraction**
* **algorithms**

A complex problem is broken down into a succession of smaller, more manageable problems using computational thinking (decomposition). Then, each of these smaller issues can be examined separately, considering how comparable issues have been resolved in the past (pattern recognition), concentrating solely on crucial information, and eliminating irrelevant data (abstraction). Next, straightforward guidelines or procedures to address each of the lesser issues can be created (algorithms).

### 2.1.3 Describe the phases of the PLDC (Program Development Life Cycle)

**VOCABULARY**

Program development life cycles (PDLCs) refer to the systematic steps or phases involved in developing programs.

With the help of PDLC, software development tasks may be arranged into manageable smaller tasks, each of which must be finished successfully before going on to the next stage. Figure 2.1 illustrates the six phases of program development lifecycle.

Figure 2. 1: Program Development Lifecycle

Program Development Life Cycle consists of six stages, and these are:

1. Requirements Analysis
2. Designing the solution
3. Coding the program
4. Testing and Debugging
5. Implementation of the solution and support
6. Documentation

**Requirements Analysis**

Fundamental to any problem-solving process is understanding the problem. This forms the part of the process, and it guides how the subsequent stages will be executed. The programmer needs to understand what the inputs are, the processes involved and the expected output. This is only done through understanding the problem. All these are included in the first phase of program development life cycle.

**Designing the solution**

The first step in program design is to concentrate on the major objective that the program is attempting to accomplish, after which the program is divided into manageable parts. The sub objectives when combined will contribute to the overall objective. This approach to program design is known as top-bottom programming, also referred to as modular programming. Different design tools are used in this stage and some of these includes pseudocode, flowcharts, algorithms, Use case diagrams, data flow diagrams, decision tables, Input Processing -Output tables, hierarchy charts and many others. Systems architect and UX/UI designers are the key people in this stage of development.

**Coding the program**

The actual programming instructions for the actions specified in the previous phase are now written or implemented using a programming language. In this stage, we write the program. The programmers will make a choice on the programming language but obviously in consultation with the management of the organisation. Examples of programming languages are  C, C++, Java, Python, C++, VB.Net, C# and many others. The programmers might opt for a structured programming approach. Structured programming is the practice of solely employing well specified control structures when creating programs. The programmer must follow the language rules, violation of any rule causes an error. These errors must be eliminated before going to the next step. Developers, both front end and back end are the common people at this stage.

**Testing and Debugging**

*Software Testing* is a method to check for computer bugs. A bug is a problem which affects the intended flow of programs.

The application will run following the correction of syntax problems. However, the program's results might not be accurate. This is caused by logical errors in the program. A logical error occurs when the programmer makes a mistake when creating a solution to a problem. Therefore, by carefully reviewing the program output using test data, the programmer must identify and fix logical problems. The process of identifying errors and eliminating them is known as *debugging*.

**Testing levels**

Table 2.1shows the common testing techniques in software development process.

Table 2. 1:Testing levels

|  |  |
| --- | --- |
| **Testing level** | **Activities** |
| Integration testing | This brings together two or more application modules to make sure they work together. Defects in interface, communication, and data flow between modules are also revealed by this kind of testing. |
| Unit testing | Unit testing is typically done throughout the application development process and its goal is to make sure that every single unit or component works as planned. |
| Regression testing | This determines if adding additional features results in a decrease in an application's functionality. |
| Stress testing | This gauges a program's robustness by seeing how much stress it can withstand before failing. |
| White Box Testing | In white-box testing, test cases are created using programming knowledge and an internal viewpoint of the system. Typically, this testing is carried out at the unit level. |
| Black Box Testing | Software testers use the black box testing technique to assess the functionality of the program being tested without examining the internal code layout. |
| Acceptance testing | a testing technique performed to determine whether or not the software system has met the requirement specification |

**Implementation of the solution**

At this stage, all testing has been done and the developers are happy with the product. The program is deployed (installed) at the user’s site. It is important to include comments to the program so that other programmers would be able to understand the code.

**Documentation**

The software project is nearly finished after testing. During the design phase, the structure charts, pseudocodes, flowcharts, and decision tables created become documentation for others involved with the software project. The last step in this phase will be to create This phase concludes with the creation of a manual that includes a user manual with program functionality, tutorials to guide users and some developers and error messages and how to resolve them.

**Review and Maintenance**

The application is monitored in this instance as well until the user gives it the all-clear. Even after it is finished, the software still needs to be constantly maintained and assessed. The programming team upgrades the software and corrects program flaws during software maintenance.

I hope you have learnt how software is developed, starting with brainstorming to identify the problem up to the last stage of review and maintenance. You must remember that execution of these stages can be done only after completion of the previous phase.

### 2.1.4 Describe the purpose of problem solving leading to solutions

**VOCABULARY**

Problem solving is the process of detecting an existing issue, locating its underlying cause or causes, and selecting the best course of action.

There are 6 steps that you should follow to solve a problem:

1. Understand the Problem

2. Formulate a Model

3. Develop an Algorithm

4. Write the Program

5. Test the Program

6. Evaluate the Solution

As you will see, the above stages are just close to those of program development lifecycle.

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
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.

Let us try to make use of the problem-solving steps to solve the above problem.

**Step 1: Understand the problem**

Some of the key questions that should be answered in order to understand the problem are:

* What input data/information is available?
* What does it represent?
* What format is it in?
* Do I have everything that I need?
* What output information am I trying to produce and in what format?

We are aware that the input in our example is a collection of grades. But we must comprehend the structure of the grades. Each grade could be a letter grade from A+ to F or a number from 0 to 100. If the answer is a number, the grade may be a real number, such as 73, or a full integer.

**STEP 2: Formulate a Model**

We will calculate the average of the incoming grades in our example. Therefore, we must be aware of the procedure (or formula) for calculating the average of a collection of numbers.

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.

We can also draw up a chart of grade values and symbols.

**STEP 3: Develop an Algorithm**

To develop an algorithm, we need to represent the instructions in some way that is understandable to a person who is trying to figure out the steps involved. To allow even non-computer people to understand the development, pseudocode would be the best to depict the scenario.

*Pseudocode is described as a technique for expressing programming code and algorithms in a English language instead of the intended programming language.*

**EXAMPLE 2.1**

Input: Number of terms n

Output: Sum and average of those n terms

Procedure SumAverage

Sum=0

i=0

Repeat for each i <n:

Read a number x

i =i +1

Avg=sum /n

Sum =sum + x

Print sum and average

**STEP 4: Write the Program**

At this stage, the design structure is changed into a programming code of choice. In our case we used C++ programming language.

Here is the sample program

**EXAMPLE 2.2**

#include <iostream>

using namespace std.

 int main () {

    int i, count, sum, inputArray[500];

    float average.

     cout << "Enter number of elements\n";

    cin >> count.

    cout << "Enter " << count << " elements\n”.

    // Read "count" elements from user

    for (i = 0; i < count; i++) {

        cin >> inputArray[i];

    }

    sum = 0;

    // Find sum of all array elements

    for (i = 0; i < count; i++) {

        sum += inputArray[i];

    }

  average = (float)sum / count;

    cout << "Average = " << average;

     return 0;

}

After coding we will need to compile our program to check for errors. *Compiling* is the process of converting a program into instructions that can be understood by the computer.

**STEP 5: Test the Program**

Once a program has been written and has passed compilation, it must be checked to see if it solves the problem it was designed to and that the answers are accurate. If everything is in order, the output from your application should be correct after running the program. Your program should be bug-free as much as possible. Test your program with many test cases (called a test suite) to find bugs effectively.

**STEP 6: Evaluate the Solution**

You should re-consider the original problem and ensure that your answer is formatted into a proper solution after your program produces a correct result. Cross check with the objective of the problem and the results. Now you can deploy the solution and constantly maintain it. Sometimes, the program may misbehave days or months after deployment due various reasons. So, you will need to regularly test it with the acceptable test data.

So now we have illustrated how you can implement problem-solving steps. In order to find the right answer to a particular problem, you need make sure to use these steps as a guide.

2.1.5 Explain and apply various problem-solving steps.

Polya’s First Principle: Understand the problem. His techniques of problem solving became very common and are used in the modern days. "How to Solve It," which George Polya published in 1945, rapidly rose to the top on problem solving techniques. Polya identified four basic principles of problem solving.

First Principle: **Understand the problem**

* Some of the questions that can help elucidate the problem could be as follows:
* Do you understand all the words used in stating the problem?
* What are you asked to find or show?
* Can you restate the problem in your own words?
* Can you think of a picture or diagram that might help you understand the problem?
* Is there enough information to enable you to find a solution?

Second Principle: **Devise a plan**

There are numerous rational approaches to resolve issues, says Polya. The greatest way to learn how to choose an effective strategy is to solve lots of issues. Following the stages will make problem solving get easier and easier. Some of the strategies include:

* Guess and check
* Eliminating possibilities
* Use a model
* Use direct reasoning
* Look for a pattern
* Draw a picture

Third Principle: **Carry out the plan**

Typically, this stage is simpler than creating the strategy. Given that you have the essential abilities, all you really need is care and patience. Stick to the strategy you've picked. If it doesn't stop failing, throw it away and pick another. Don't be fooled; this is how math is done, even by experts.

Fourth Principle: **Look back**

The idea behind looking back is to allow the process of reflection on what worked well and what did not work. People learn more from reflections. You will be able to foresee what approach to take to address difficulties in the future by doing this.

**eLink**

https://youtu.be/zhL3EMFSm6o

### 2.1.6 Use appropriate tools and techniques to present a solution.

**VOCABULARY**

Agile software development is a term used to describe software development approaches that are based on the idea of iterative development. The methodology is characterised by self-organizing cross-functional teams that collaborate to establish program needs and solutions.

User stories are among some of the techniques to present a solution. User stories are mostly used in agile development methodologies.

In agile product management and software development a user story is a succinct, casual, and straightforward summary of software functionality that the system's end users need. Its main objective is to offer software features that will meet the needs of the consumer. User stories are simple, yet extremely powerful constructs: they describe pieces of functionality from a user’s point of view, expressed in a solid, compact way.

***Why User Stories?***

1. User stories provide an excellent way to [define your product with clarity](https://www.theinnovationmode.com/the-innovation-blog/the-mvp-recipe-what-to-include-in-your-product-when-and-why).
2. User stories help to achieve *cross-team clarity* on *what* to build, for *whom*, *why,* and when.
3. User stories encourage participation by non-technical members.
4. User stories help in [defining the entire product](https://www.theinnovationmode.com/the-innovation-blog/the-mvp-recipe-what-to-include-in-your-product-when-and-why) — as a set of solid, wisely-prioritized stories.

User stories are completely from the end-user perspective which follows the Role-Feature-Benefit pattern. The pattern is as follows:

As a [ type of user], I want [ an action], so that [ some reason]

Some points outlined which are taken into consideration during writing user stories like

1. Requirements
2. Tasks and their subtasks
3. Actual user
4. Importance to user words/feedback
5. Breaking user stories for larger requirements

A common principle when writing user stories is to consider 3C’s.

* Card-write stories on cards
* Conversation-conduct conversations with the people involved to get more information.
* Confirmation-meet the acceptance criteria of the software.

**Example of a user story**

As a customer I want ability to book a movie ticket that matches my preferences so that I get to quickly and easily book the movie of my choice.

User Story Description

This feature will involve user selecting a specific city, searching for the movie name, selecting a specific timeslot, and then completing the order booking formalities.

**Acceptance Criteria:**

1. User navigates to the search movie page
2. User selects city
3. User enters movie name
4. System searches for the matching movies and displays results
5. User selects specific movie timing and proceeds to book
6. User enters no. of guests and seats
7. User provides payment information
8. System validates the payment information and confirms the booking
9. System sends email/SMS to the user with booking confirmation details

**Key Terms**

* **user story** – one short sentence in everyday language of the end user that states what a user does as part of his or her work
* **acceptance criteria** – features that must be present in the final system for the user to be satisfied
* **use case** – an activity that the system performs, usually in response to a request by a user
* **user goal technique** – a technique used in use cases which targets on specifying goals or to be completed by a user
* **event** – something that occurs at a specific time and place, can be precisely identified, and must be remembered by the system

FORMATIVE ASSESSMENT 2.1 INDIVIDUAL TASK

2.1.1 Define the term problem solving as applied in programming. (2)

2.1.2 What is meant by the term computational thinking? (2)

2.1.3 Identify **FOUR** cornerstones of computational thinking. (4)

2.1.4 With the aid of a diagram, represent the SIX stages of program development life cycle. (6)

2.1.7 List SIX steps which must be followed in problem solving. (6)

2.1.8 Write an algorithm that reads three numbers and prints the value of the largest number. (10)

2.1.9 Define the term algorithm as used in programming. (2)

**Total: 32 marks**

## 2.2 Construct an algorithm and present a solution to a given problem

Earlier in this topic, we spoke about the program development lifecycle and the associated stages. We highlighted the design tools, but we want to go a bit deeper with examples. In this section we will discuss flowcharts, algorithms and pseudocodes.

### 2.2.1 Define the term algorithm and its purpose in the problem-solving process.

**VOCABULARY**

An algorithm is a finite sequence of steps expressed for solving a problem.

An algorithm can be defined as “a process that performs some sequence of operations in order to solve a given problem”. Algorithms are used for calculation, data processing, and many other fields. It is the programmers choice to select the type of modelling tool or technique to use when solving a problem. Further, it is important to note that one problem can have different algorithms from different programmers. There is no one size fits all in design.

Let’s say that you have a friend arriving at the railway station, and your friend needs to get from the railway station to your house. Here are three different ways (algorithms) that you might give your friend for getting to your home.

**The taxi algorithm:**

* Go to the taxi
* Get in a taxi
* Give the driver my address.

**The call-me algorithm:**

* When your train arrives, call my mobile phone.

**The bus algorithm: Outside the railway station, catch Golden Arrow Bus.**

* Transfer to taxi at Wynberg station.
* Get another taxi to HoutBay.
* Get off by the Shell Garage in Plumstead
* Walk two blocks west to my house in towards Victoria Hospital.
* My house is painted in yellow

Three reasons for using algorithms are:

* **Efficiency**- Certain types of problems, like sorting, occur often in computing. To solve such problems, efficient algorithms must be used, considering the time and cost factors involved in each algorithm.
* **Abstraction**- Algorithms provide a level of abstraction in problem solving. When we see a more complicated problem in a simplified light, we can consider the simpler problem to be an abstraction of the more complicated one.
* **Reusability**- Algorithms are frequently reusable.

Algorithms can be expressed in many different notations, including natural languages, pseudocode, flowcharts and programming languages.

### 2.2.2 Basic algorithmic constructs used to create a flowchart.

We have defined what an algorithm is in section 2.2.1. This section focuses on the basic constructs of algorithms used to create a flowchart. First, we will discuss the characteristics of an algorithm.

**Characteristics of an Algorithm**

* **Unambiguous**
* **Input-must have 0 or more inputs**
* **Output** − should have 1 or more well-defined outputs.
* **Finiteness**-must terminate after several steps
* **Feasibility**-should be feasible with available resources
* **Independent**-must have step by step directions independent from other programs

Design an algorithm to add two numbers and display the result.

**EXAMPLE 2.3**

Step 1 − START

Step 2 − declare three integers a, b & c

Step 3 − define values of a & b

Step 4 − add values of a & b

Step 5 − store output of step 4 to c

Step 6 − print c

Step 7 − STOP

**Types of Algorithms**

* **Brute Force Algorithm**- A brute force algorithm solves a problem through exhaustion: it goes through all possible choices until a solution is found.

**Example**: A company that uses the same types of vehicles for delivery has the same car keys. Let's say there are 10 keys, and you do not remember which key belongs to the car that one wants to drive, the most probable solution would be to test all of them one by one until you get the correct one. This technique is time consuming.

* **Recursive Algorithm**- In this instance, a problem is divided into multiple smaller components and repeatedly called by the same function.This is a bit complex but powerful and fast.

**Example**: Commonly used in solving problems such as factorial of a number or calculating Fibonacci series.

* [**Backtracking Algorithm**](https://www.geeksforgeeks.org/backtracking-algorithms/)**-** Every time a solution fails, we go back to the original problem, build on the new one, and repeat the process until the problem is solved or all potential solutions have been considered. A good example is solving puzzles. The puzzle is built incrementally by inserting pieces which match fit and those that do not fit are put aside.
* **Divide-and-conquer algorithm-**A a problem is repeatedly divided into two or more subproblems of the same or similar type, until these are sufficiently straightforward to be solved by themselves.

**Example : Mainly used in binary searching techniques where the list of elements is split into two and arranged in order and the middle element is compared with the search element.**

**PROBLEM**: Find the sum of 5 numbers. Use an algorithm to solve the problem

**Solution:** Algorithm (in simple English)

**EXAMPLE 2.4**

Step 1: Start

Step 2: Initialize sum = 0 and count = 0    (PROCESS)

Step3: Enter n    (I/O)

Step 4: Find sum + n and assign it to sum and then increment count by 1 (PROCESS)

Step 5: Is count < 5 (DECISION)

if YES go to step 2  
 else  
Step 6: Print sum (I/O)

Step 7: Stop

**Flowcharts**

**VOCABULARY**

A flowchart is a type of diagram (graphical or symbolic) that represents an algorithm or process.

A flowchart is a diagrammatic representation of a problem using agreed symbols. Each step in the process is represented by a different symbol and contains a short description of the process step. The flow chart symbols are linked together with arrows showing the process flow direction. A flowchart typically shows the flow of data in a process, detailing the operations/steps in a pictorial format which is easier to understand than reading it in a textual format.

**Flowchart symbols**

Table 2. 2:Flowchart symbols

|  |  |  |
| --- | --- | --- |
| **Symbol** | **Name** | **Function** |
|  | Start/End | An oval shape represents a start or end of a program |
|  | Arrows | A line is a connector that shows relationships between the representative shapes |
|  | Input/Output | A parallelogram represents an input or output to the program |
|  | Process | A rectangle represents a process |
|  | Decision | A diamond indicates a decision |
|  | Connector symbol | Indicates that the flow continues where a matching symbol has been placed. |

Consider that we need to find the sum, average and product of 3 numbers given by the user.

Algorithm for the given problem is as follows:

**EXAMPLE 2.5**

Step 1: Start

Step 2: Read num1, num2, num3

Step 3: Compute Sum (S) as num1 + num2 + num3

Step 4: Compute Average (A) as S / 3

Step 5: Compute Product (P) as num1 x num2 x num3

Step 6: Display Sum (S), Average (A), Product (P)

Step 7: Stop

Take note that it is not mandatory to start by developing an algorithm and then draw a flowchart. Design tools are selected based on what the programmer sees as best representing the given scenario. Let us now represent the above algorithm using the flowchart symbols discussed in Table 2.2. Figure 2.2 represents the same process depicted using an algorithm in example 2.5.

Text

Description automatically generated

Figure 2. 2: Example of flowcharting

Let us use the problem which we did earlier on sum of numbers.

**PROBLEM**: Find the sum of 5 numbers.

We already wrote the algorithm. Now we are going to use flowchart symbol to depict the scenario.

**Flowchart for summing up 5 numbers**

:A picture containing diagram

Description automatically generated

Figure 2. 3: Flowchart for sum of 5 numbers

**Basic guidelines for drawing a flowchart**

In drawing a proper flowchart,

* All necessary requirements should be listed out in logical order.
* The flowchart should be neat, clear, and easy to follow.
* There should not be any room for ambiguity in understanding the flowchart.
* The flowchart is to be read left to right or top to bottom.
* A process symbol can have only one flow line coming out of it.

### 2.2.3 Create a flowchart to present a particular algorithm and its associated tasks

In section 2.2.2 we identified the common shapes for making a flowchart. Let us consider the following challenge.

**Problem:**

Add two numbers from the keyboard and display the output. Design a flowchart to solve the problem.

To solve this problem, we will take a variable sum and set it to zero. In the previous example, we did not set values of num1, num2 and num3 to 0. It is always a good practice to set default values for variables. Then we will take the two numbers number1 and number2 as input. Next, we will add both the numbers and save the result in the variable sum i.e., sum = number1+number2. Finally, we will print the value stored in the variable sum.

Here is the algorithm for the above example.

Algorithm (in simple English)

**EXAMPLE 2.5**

* Initialize sum = 0 (PROCESS)
* Enter the numbers (I/O)
* Add them and store the result in sum (PROCESS)
* Print sum (I/O)

**Flowchart**

Text

Description automatically generated

Figure 2. 4: Adding two numbers

Figure 2.4 is a sequential flowchart since steps are followed one after the other in sequence without a choice of condition followed. If the program requires a choice of events based on specific conditions being met, conditional flowcharts will be used. A conditional flowchart is used when a condition is imposed on a problem. The condition will either be true or false. The course of the problem depends on the answer to the condition. Here is the challenge:

**Write a program to check if the given number is a multiple of 3 or not.**

Here are the steps:

1. Input the number 'n'.
2. Divide 'n' be 3.
3. If reminder equals 0, print 'n' is a multiple of 3'.
4. If reminder does not equal to 0, print 'n' is not a multiple of 3'.

Use a flowchart to depict the scenario in the algorithm.

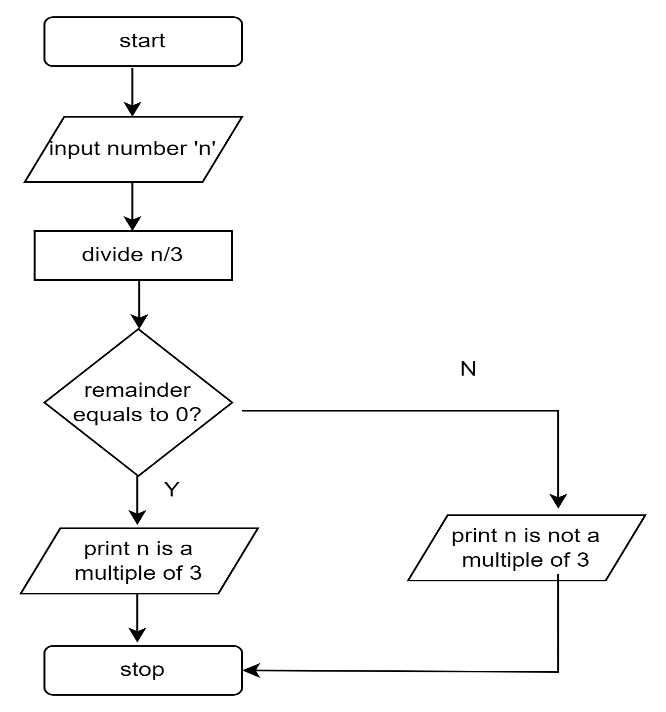


Figure 2. 5: conditional flowchart

**Repetition flowchart** is used when the program requires the act of repeating one or more steps in a process. It entails turning backwards from the regular progression of steps in order to reach an earlier phase. When the branching test is run, a condition determines the branching decision.

**Program:**

**Write a program to print hello five times.**

In this case, we take a variable count and set it to zero. Then we print "Hello World" and increment count by 1.

i.e., count = count + 1

Next, we check if count is less than 10. If this is TRUE, then we again print "Hello World" and increment the variable count. On the other hand, if the condition if FALSE then we will stop.

Here is the flowchart for the problem.

A picture containing diagram

Description automatically generated

Figure 2. 6:Repetition flowchart

Let us consider the following challenge.

**Challenge:** Draw a flowchart to log in to Facebook account

To log in to Facebook account we first enter the Facebook URL www.facebook.com in our browser. This request is sent to the Facebook server and it responds by sending us the home page of Facebook. Next, we enter our registered Email ID and Password and click the Login button. Then our login credential is checked. If it is correct, we are show our profile. On the other hand, if the login credential is wrong then an error occurs, and we are prompted to re-enter our Email ID and Password.

**Solution**

Figure 2.7 represents the logical steps of logging on to Facebook.

Diagram

Description automatically generated with medium confidence

Figure 2. 7: Logging on to Facebook

**PROBLEM:**

Write an algorithm to determine a student’s final grade from 4 tests scores and indicate whether it is passing or failing. The final grade is calculated as the average of four marks.

A picture containing polygon

Description automatically generated

Figure 2. 8:Average of four test scores

### 2.2.4 Interpreting a basic flow chart and its intended operation

How frequently have you found it difficult to comprehend a process after being given a thorough explanation? In these circumstances, flow charts are a helpful tool since they make a process simple to comprehend at a glance. They effectively illustrate what happens at each stage and how this influences other decisions and actions using only a few basic words and symbols.

**When to Use a Flow Chart**

* Defining a process.
* [Standardize](https://www.mindtools.com/pages/article/5s-system.htm)  a process.
* Communicating a process
* Identify bottlenecks or waste in a process

We have already discussed the key shapes symbols used in creating a flowchart. Interpreting your Flowchart will help you to :

* Determine who is involved in the process.
* Form theories about root causes.
* Identify ways to streamline the process.
* Determine how to implement changes to the process.
* Locate cost-added-only steps.
* Provide training on how the process works or should work.

**Benefits of Using Flowcharts**

The benefits of flowcharts are as follows:

* **Communication:** Flowcharts are a better way to communicate the logic of a system to all parties involved.
* **Effective analysis:** A flowchart can help you analyse a problem more effectively.
* **Proper documentation:** Program flowcharts are useful for program documentation, which is required for a variety of reasons.
* **Efficient Coding:** During the systems analysis and program development phases, the flowcharts serve as a guide or blueprint.
* **Proper Debugging:** The flowchart aids in the debugging process.
* **Efficient Program Maintenance:** The use of a flowchart simplifies program maintenance. It allows the programmer to focus his or her efforts more effectively on that aspect.

**Limitations of Using Flowcharts**

Although a flowchart is a very useful tool, there are a few limitations in using flowcharts which are listed below:

* **Complex logic:** The program logic can be quite complicated at times and in such instances flowchart become more complex and clumsy.
* **Alterations and Modifications:** If changes are required, the flowchart may need to be completely redrawn.
* **Reproduction:** Because flowchart symbols cannot be typed, reproduction of flowcharts is difficult.
* The essentials of what is done can easily be lost in the technical details of how it is done.

# FORMATIVE ASSESSMENT 2.2 INDIVIDUAL TASK

2.2.1 Discuss in brief **ANY FOUR** testing phases in program development lifecycle. (12)

2.2.2 Consider the following scenario.

Tickets are sold for a concert at $20 each, if 10 tickets are bought then the discount is 10%, if 20 tickets are bought the discount is 20%. No more than 25 tickets can be bought in a single transaction. Design a flowchart to depict the above scenario. (10)

2.2.3 List SIX characteristics of a good algorithm. (6)

2.2.4 Mr January’s class of programming has 25 students who sat for the test. The teacher asks you to design a flowchart to calculate the average from 25 exam scores. (10)

2.2.5 Write an algorithm to depict the given scenario.

Scenario: Find the area of a Circle of radius r.

**HINT**

Inputs to the algorithm: Radius r of the Circle.

Expected output: Area of the Circle (6)

2.2.6 List and explain SIX benefits of using flowcharts. (12)

**Total : 66 Marks**

# SUMMATIVE ACTIVITY 2.3 INDIVIDUAL TASK

2.3.1 Explain **FOUR** cornerstones of computational thinking. (8)

2.3.2 What is a conditional flowchart? (2)

2.3.3 You are requested to design a program to convert temperature recorded in Fahrenheit into degrees Celsius.

The formula for conversion is:

Celsius=5/9 \* (F-32) (5)

2.3.4 One of the uses of computer programs is mathematical calculation. Here is a problem Write an algorithm and draw a flowchart that will calculate the roots of a quadratic equation.

Formula: ax2 + bx+ c = 0

Hint: d = sqrt ( b2-4ac), and the roots are: x1 = (–b + d)/2a and

x2 = (–b – d)/2a

algorithm (7)

flowchart (7)

2.3.5 Consider the following statement.

Problem Statement

Calculate the interest of a bank deposit. You are to read the amount, years and interest rate from the keyboard and print the interest amount.

**Hint: I=Amount \* Years \*Rate/100**

Draw a flowchart to depict the scenario. (7)

2.2.6 List and explain THREE limitations of flowcharts. (6)

* + 1. The symbol denotes \_\_\_\_\_\_\_

1. I/O
2. Flow
3. Process
4. Decision (1)

2.2.8 A box that can represent two different conditions.  
a) Rectangle  
b) Diamond  
c) Circle  
d) Parallelogram (1)

2.2.9 In computer science, algorithm refers to a pictorial representation of a flowchart.  
a) True  
b) False (1)

2.2.10 The operation represented by parallelograms is \_\_\_\_\_\_\_\_.  
a) Input/Output  
b) Assignment  
c) Comparison  
d) Conditions (1)

**Total :46 Marks**

## References

Polya, G., 2004. *How to solve it: A new aspect of mathematical method* (Vol. 85). Princeton university press.

<https://www.geeksforgeeks.org/most-important-type-of-algorithms/>