University of Maiduguri

Department of Computer Science

Course Code: CSC 102

Course Title: Introduction to Problem Solving

Introduction

Computer Science is the study of problems, problem-solving, and the solutions that come out of this problem-solving process.

- Given a problem, the goal is to develop an algorithm to solve the problem.
- An algorithm is a step-by-step list of instructions to solve the problem.

Problem Solving is the sequential process of analysing information related to a given situation and generating appropriate response options. In solving a problem, there are some well-defined steps to be followed. Problem Solving is the sequential process of analyzing information related to a given situation and generating appropriate response options. Computer based problem solving is a systematic process of designing, implementing and using programming tools during the problem solving stage. Computer-Based Problem Solving Process is a work intended to offer a systematic treatment to the theory and practice of designing, implementing, and using software tools during the problem solving process.

Problem Solving Steps:

Basically, there are six (6) steps that you should follow in order 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

STEP 1: Understand the Problem:

First step to solving any problem is to make sure that you understand the problem that you are trying to solve.

STEP 2: Formulate a Model:

Now we need to understand the processing part of the problem. Many problems break down into smaller problems that require some kind of simple mathematical computations in order to process the data.

STEP 3: Develop an Algorithm:

Now that we understand the problem and have formulated a model, it is time to come up with a precise plan of what we want the computer to do.

An algorithm is a precise sequence of instructions for solving a problem.

Two commonly used representations for an algorithm are by using:

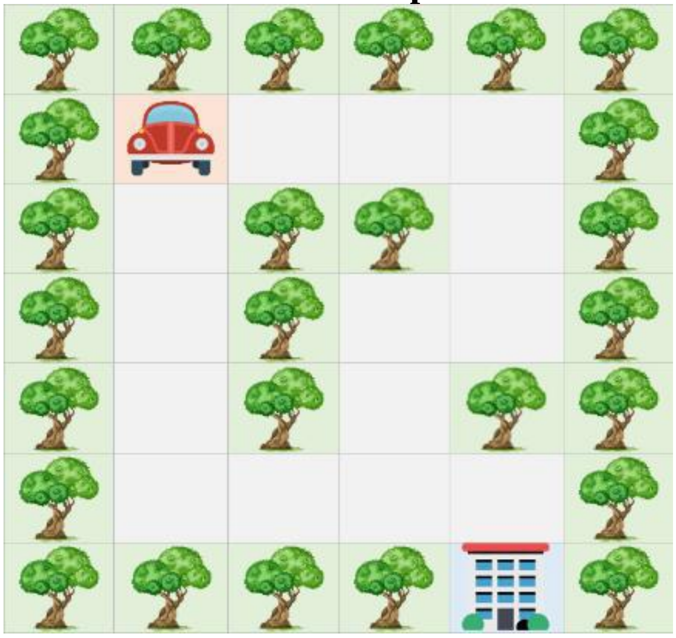
- Pseudocode: this is a simple and concise sequence of English-like instructions to solve a problem.
- Flow chart: This is a diagram that represents a set of instructions. Flowcharts normally use standard symbols to represent the different types of instructions.

In other word, a flowchart is a graphical representation of various logical steps of a program. These expressions use several shapes, including the geometric ones, to show the step-by-step process with arrows while establishing a data flow.

STEP 4: Write the Program:

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.
STEP 5: Test the Program:
Once you have a program written that compiles, you need to make sure that it solves the problem that it was intended to solve and that the solutions are correct. Running a program is the process of telling the computer to evaluate the compiled instructions.
STEP 6: Evaluate the Solution:
Once your program produces a result that seems correct, you need to re-consider the original problem and make sure that the answer is formatted into a proper solution to the problem. It is often the case that you realize that your program solution does not solve the problem the way that you wanted it to. You may realize that more steps are involved.

Road Example



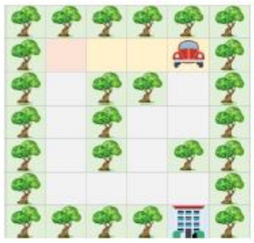
Imagine that you have the following image above, which is a map of a road leading to the building shown in the picture.
There is a car and trees.

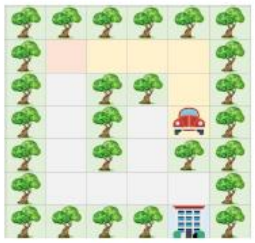
- The car cannot cross the trees.
- The road is divided into squares to calculate the steps of the car.
- Each square is considered as one step.

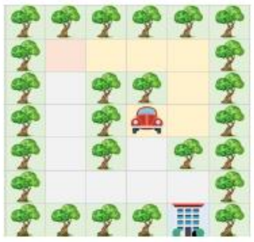
How can the car arrive at the building?

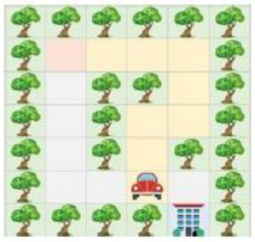
Road Example
Solution A

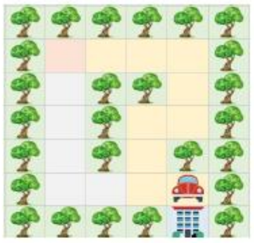
- Step 1: Move to the right three steps.
- Step 2: Move to down two steps.
- Step 3: Move to the left one step.
- Step 4: Move to down two steps.
- Step 5: Move to the right one step.
- Step 6: Move to down one step.

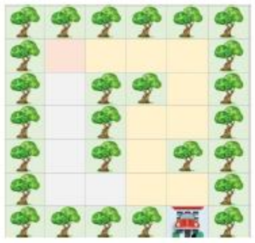

Step 1


Step 2


Step 3

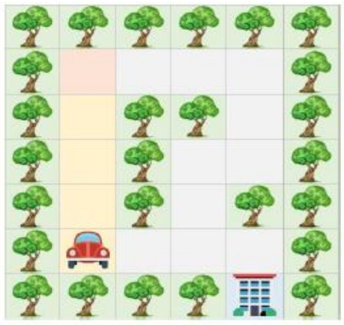

Step 4

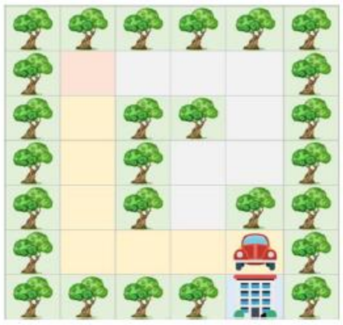

Step 5

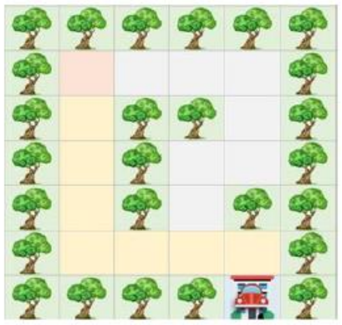

Step 6

Road Example
Solution B

- Step 1: Move to down four steps.
- Step 2: Move to the right three steps.
- Step 3: Move to down one step.


Step 1


Step 2


Step 3

Different Solutions

- As we can see that Solution A and Solution B are both correct solutions to the same problem, but there are differences in the complexity and efficiency of the solutions.
- The cost of Solution A is 10 steps while Solution B is 8 steps, so we consider Solution B as a better solution based on the number of steps.
- Reducing the number of steps in the previous example means reducing the amount of fuel needed by the vehicle and speeding up the arrival time.

Algorithm

The term algorithm originally referred to any computation performed via a set of rules applied to numbers written in decimal form. The word is derived from the phonetic pronunciation of the last name of Abu Ja'far Mohammed ibn Musa al- Khowarizmi, who was an Arabic mathematician who invented a set of rules for performing the four basic arithmetic operations (addition, subtraction, multiplication, and division) on decimal numbers.

An **algorithm** is a representation of a solution to a problem. If a problem can be defined as a difference between a desired situation and the current situation in which one is, then a problem solution is a procedure, or method, for transforming the current situation to the desired one. We solve many such trivial problems every day without even thinking about it, for example making breakfast, travelling to the workplace etc. An algorithm is an effective step-by-step procedure for solving a problem in a finite number of steps. In other words, it is a finite set of well-defined instructions or step-by-step description of the procedure written in human readable language for solving a given problem. An algorithm itself is division of a problem into small steps which are ordered in sequence and easily understandable. Algorithms are very important to the way computers process information because a computer program is basically an algorithm that tells computer what specific tasks to perform in what specific order to accomplish a specific task. The same problem can be solved with different methods. So, for solving the same problem, different algorithms can be designed. In these algorithms, number of steps, time and efforts may vary.

Algorithms

- An algorithm is a set of obvious, logical, and sequential steps that solve a specific problem.
- To put it simply, the algorithm is like a recipe for preparing a specific food.
- Following the steps of the algorithm will end up solving the problem.

An **algorithm** must:

- Be lucid (clear), precise and unambiguous.
- Give the correct solution in all cases, and eventually end.

What is **pseudocode**?

- It is English that looks like code
- But it is not actual code (only looks a little similar).
- Think of pseudocode as a way of expressing your algorithm.

What is a **flowchart**?

- A graphical representation of the sequence of operations in an information system or program.

Now, we express our algorithms in many ways:

- **Pseudocode**: this is not "real code", but a slightly more formal way of writing the algorithmic steps as an example, maybe the programmer does not know the language he/she will use. Therefore, they just write pseudocode during Problem-Solving Phase.
- **Flowchart**: this is a graphical representation of the algorithm
- **Actual code**: this is during the Implementation Phase Python, Java, C++, C, etc

Importance of Algorithm

1. Algorithms Define Efficiency: At their core, algorithms are step-by-step sets of instructions designed to solve specific problems. They are like recipes for computers, telling them how to perform tasks efficiently and accurately. Without algorithms, computers would be lost in a sea of data, unable to make sense of it.
2. Universal Problem Solvers: Algorithms are universal problem solvers. From search engines finding relevant web pages to GPS systems finding the fastest route, algorithms work tirelessly behind the scenes, tackling a staggering variety of problems.
3. Complexity Management: In a world flooded with data, algorithms help us manage complexity. They sift through vast datasets, making sense of information and distilling it into actionable insights. This ability is invaluable in fields like data science, where algorithms analyze data to make predictions and decisions.
4. Time and Resource Savings: Algorithms save time and resources. They enable automation, reducing the need for manual labor. Think of algorithms in manufacturing, optimizing production processes and saving both time and money.
5. Innovation Catalysts: Algorithms drive innovation, they power everything from recommendation systems on streaming platforms to the development of new drugs in healthcare. By solving problems more efficiently, algorithms open doors to new possibilities and discoveries.

Exploring Real-World Applications of Algorithms

1. Web Search Engines: Search engines like Google utilize complex algorithms to retrieve and rank web pages, ensuring that you find the most relevant results for your queries quickly. These algorithms consider factors like keywords, user behavior, and page authority.
2. GPS Navigation: GPS algorithms calculate the fastest or shortest route to your destination, taking into account traffic conditions, road closures, and real-time

data. They optimize your travel experience by minimizing travel time and fuel consumption.

3. Social Media Feeds: Algorithms power social media platforms by curating your feed to show you content you're likely to engage with. They analyze your preferences, interactions, and user data to personalize your social media experience.
4. E-commerce Recommendations: When you shop online, algorithms suggest products based on your past purchases, browsing history, and the behavior of other shoppers with similar preferences. These recommendations algorithms enhance user experience and boost sales.
5. Medical Diagnosis: In healthcare, algorithms assist doctors in diagnosing diseases and predicting patient outcomes. Machine learning algorithms analyze medical data to identify patterns and make predictions about patient health.
6. Finance and Stock Trading: Algorithmic trading uses mathematical models and algorithms to make rapid, data-driven decisions in financial markets. These algorithms execute trades at optimal times and prices, often faster than human traders.
7. Manufacturing and Supply Chain: Algorithms optimize production schedules, manage inventory, and streamline supply chain logistics. They ensure products are manufactured efficiently and delivered to consumers on time.
8. Natural Language Processing: Algorithms for natural language processing enable chatbots, virtual assistants, and automatic translation services. They help bridge language barriers and improve communication.
9. Image and Video Processing: Image recognition algorithms are used in security systems, autonomous vehicles, and even in sorting and quality control in manufacturing. Video compression algorithms enable efficient streaming and storage of multimedia content.
10. Weather Forecasting: Meteorologists rely on sophisticated algorithms to process vast amounts of atmospheric data and generate accurate weather forecasts. These forecasts are crucial for disaster preparedness and everyday planning.

Analyzing the Time and Space Complexity of Algorithms

When it comes to evaluating the efficiency and performance of algorithms, analyzing their time and space complexity is paramount. These analyses provide crucial insights into how algorithms behave under different conditions and help in making informed decisions about algorithm selection. Here's why analyzing time and space complexity is essential:

1. Performance Prediction: Time complexity analysis allows us to predict how long an algorithm will take to run as the input size grows. This prediction is

invaluable in choosing the most efficient algorithm for a particular task. It helps avoid unexpected slowdowns when dealing with larger datasets.

2. Resource Management: Space complexity analysis helps us understand how much memory an algorithm requires. In memory-constrained environments or when dealing with massive datasets, this analysis is crucial to ensure an algorithm doesn't cause memory overflow or excessive resource usage.
3. Comparative Evaluation: Time and space complexity analyses enable us to compare different algorithms objectively. By quantifying their performance characteristics, we can select the algorithm that best suits the specific requirements of a problem.
4. Algorithm Optimization: Understanding the bottlenecks in an algorithm's time or space complexity allows for optimization. We can identify which parts of the algorithm consume the most resources and focus on improving those areas for efficiency gains.
5. Scalability Assessment: Time and space complexity analyses are essential for assessing how well an algorithm scales with input size. This information is critical when designing systems that need to handle increasing amounts of data or users.
6. Algorithm Design: Time and space complexity considerations often guide algorithm design decisions. They influence the choice of data structures, algorithms, and optimization techniques used in implementation.
7. Resource Allocation: In real-world applications, resources like processing power and memory are finite. Analyzing complexity helps allocate these resources optimally, ensuring that an algorithm can run efficiently on the available hardware.
8. Troubleshooting: When an algorithm exhibits unexpected behavior or performance issues, analyzing its time and space complexity can pinpoint the source of the problem. This aids in debugging and fine-tuning the algorithm.

Programming Constructs: are used to describe the control flow of the algorithm.
SEQUENCE represents linear tasks sequentially performed one after the other.
WHILE a loop with a condition at its beginning.
DO-WHILE a loop with a condition at the bottom.
FOR another way of looping.
IF-THEN-ELSE a conditional statement changing the flow of the algorithm.
CASE the generalization form of IF-THEN-ELSE.

Reasons for Algorithm

A programmer writes a program to instruct the computer to do certain tasks as desired. The computer then follows the steps written in the program code. Therefore, the programmer first prepares a roadmap of the program to be written, before writing the code. Without a roadmap, the programmer may not be able to clearly visualise the instructions to be written and may end up developing a program which may not work as expected. Such a roadmap is nothing but the **algorithm** which is the building block of a computer program.

Steps Involved in Algorithm Development

An algorithm can be defined as “a complete, unambiguous, finite number of logical steps for solving a specific problem “

Step 1. **Identification of input:** For an algorithm, there are quantities to be supplied called input and these are fed externally. The input is to be identified first for any specified problem.

Step 2: **Identification of output:** From an algorithm, at least one quantity is produced, called for any specified problem.

Step 3: **Identify the processing operations:** All the calculations to be performed to lead to output from the input are to be identified in an orderly manner.

Step 4: **Processing Definiteness:** The instructions composing the algorithm must be clear and there should not be any ambiguity in them.

Step 5: **Processing Finiteness:** If we go through the algorithm, then for all cases, the algorithm should terminate after a finite number of steps.

Step 6: **Possessing Effectiveness:** The instructions in the algorithm must be sufficiently basic and in practice they can be carries out easily.

Characteristics of Algorithm

An algorithm must possess following characteristics:

1. Precision: the steps are precisely stated or defined.
2. Uniqueness: results of each step are uniquely defined and only depend on the input and the result of the preceding steps.
3. Finiteness: the algorithm always stops after a finite number of steps.
4. Input: the algorithm receives some input.
5. Output — the algorithm produces some output

Exercise

Area of a Rectangle Calculator Solution A

Solution A: Good:

1.Ask the user to enter Width
2.Ask the user to enter Height
3.Set Areato (Width×Height)
4.Display Area for the user

As you can see in this solution, we have described the steps that are going to solve the problem. You can describe the steps in your own way, but your description of the steps should be obvious, logical, and sequential.

Area of a Rectangle Calculator Solution B

Solution B: Bad:

1.Ask the user to enter Width
2.Ask the user to enter Height
3.Calculate Area
4.Display Areafor the user

The reason for considering Solution B as a bad solution:

- Step 3 is not clear because it does not explain how we can calculate Area.

- So, this algorithm is bad because its steps are not obvious.

Area of a Rectangle Calculator Solution C
Solution C -Bad:

```
1.Set Areato (Width×Height)
2.Ask the user to enter Width
3.Ask the user to enter Height
4.Display Areafor the user
```

The reasons for considering Solution C as a bad solution:

- We don't know what Width and Height at the Step1 are. In other words, Width and Height have not been defined before Step1, so we cannot use them because they do not exist yet.
- What about Step2 and Step3? Width and Height are defined there! After Step2, Width does exist, but Height does not. After Step3, Height does exist. Both Width and Height are available to be used at or after step4.
- So, this algorithm is bad because its steps are not correctly sequential.

Algorithms calculate Simple Inter set

```
Step 1: Read the three input quantities' P, N and R.
Step 2: Calculate simple interest as Simple interest = P* N* R/100
Step 3: Print simple interest.
Step 4: Stop.
```

Algorithm find Area of Triangle

```
Step 1: Input the given elements of the triangle namely sides $b$, $c$ and angle between the sides $A$
Step 2: Area = $12*b*c*sin(A)$
Step 3: Output the Area
Step 4: Stop.
```

3. Write an algorithm to find the largest of three numbers X,Y,Z.

Algorithm find Largest of Three Numbers

```
Step 1 Read the numbers $X,Y,Z$.
Step 2 If (X > Y)
big = X
Else big = Y
Step 3 If (big < Z)
big = Z
Step 4 Print big
Step 5 Stop.
```

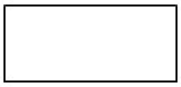
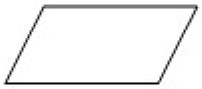
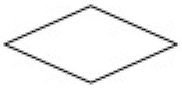
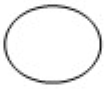
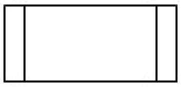
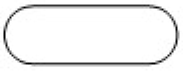
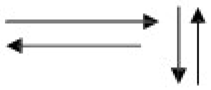
Write an algorithm to add two numbers entered by user.

```
Step 1: Start
Step 2: Declare variables num1, num2 and sum.
Step 3: Read values num1 and num2.
Step 4: Add num1 and num2 and assign the result to sum.
sum←num1+num2
Step 5: Display sum
Step 6: Stop
```

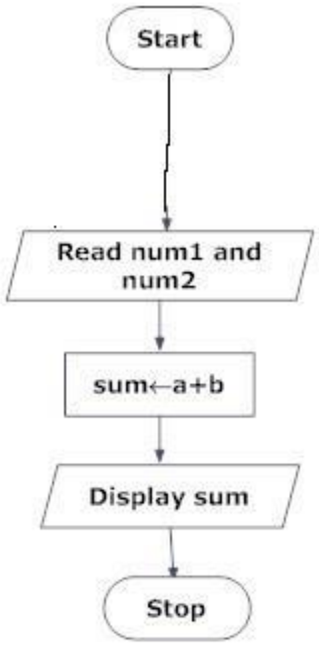
Flowcharts

Flowcharting is a tool developed in the computer industry, for showing the steps involved in a process. A flowchart is a diagram made up of boxes, diamonds, and other shapes, connected by arrows - each shape represents a step in the process, and the arrows show the order in which they occur. Flowcharting combines symbols and flowlines, to show figuratively the operation of an algorithm.

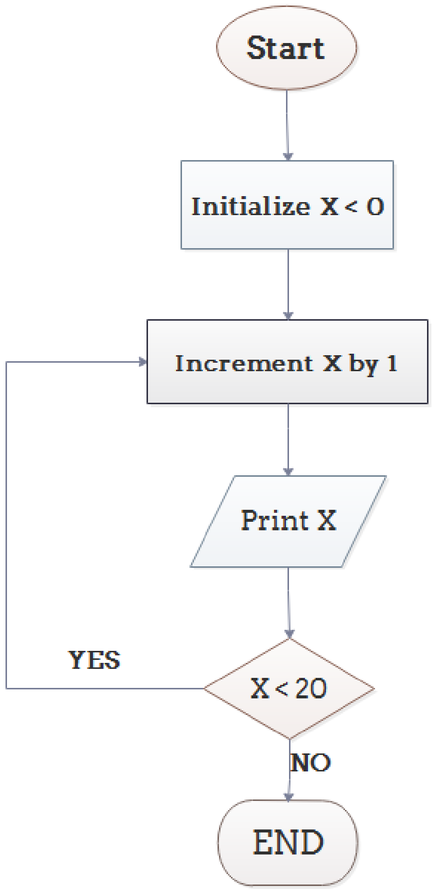
A flowchart is a graphical representation of an algorithm or of a portion of an algorithm. Flowcharts are drawn using symbols. The main purpose of a flowchart is to analyze different processes. The main symbols used to draw a flowchart are shown in following figures.

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)
	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.
	Terminal	Indicates the starting or ending of the program, process, or interrupt program.
	Flow Lines	Shows direction of flow.

Example 1: Flowchart that sum two input numbers



Example 2: Flowchart that Print numbers from 1 to 20



Pseudocode

Pseudocode (pronounced SOO-doh-kohd) is a detailed yet readable description of what a computer program or algorithm must do, expressed in a formally styled natural language rather than in a programming language. Pseudocode is sometimes used as a detailed step in the process of developing a program. It allows designers or lead programmers to express the design in detail and provides programmers a detailed template for the next step of writing code in a specific programming language.

Pseudo code is a term which is often used in programming and algorithm-based fields. It is a methodology that allows the programmer to represent the implementation of an algorithm. Simply, we can say that it's the cooked up representation of an algorithm. Often at times, algorithms are represented with the help of pseudo codes as they can be interpreted by programmers no matter what their programming background or knowledge is.

Reasons for using Pseudocode

1. **Better readability**. Often, programmers work alongside people from other domains, such as mathematicians, business partners, managers, and so on. Using pseudocode to explain the mechanics of the code will make the communication between the different backgrounds easier and more efficient.
2. **Ease up code construction.** When the programmer goes through the process of developing and generating pseudocode, the process of converting that into real code written in any programming language will become much easier and faster as well.
3. **A good middle point between flowchart and code.** Moving directly from the idea to the flowchart to the code is not always a smooth ride. That's where pseudocode presents a way to make the transition between the different stages somewhat smoother.
4. **Act as a start point for documentation.** Documentation is an essential aspect of building a good project. Often, starting documentation is the most difficult part. However, pseudocode can represent a good starting point for what the documentation should include. Sometimes, programmers include the pseudocode as a docstring at the beginning of the code file.
5. **Easier bug detection and fixing.** Since pseudocode is written in a human-readable format, it is easier to edit and discover bugs before writing a single line of code. Editing pseudocode can be done more efficiently than testing, debugging, and fixing actual code.

The main constructs of pseudocode

The core of pseudocode is the ability to represent 6 programming constructs (always written in uppercase): *SEQUENCE, CASE, WHILE, REPEAT-UNTIL, FOR, and IF-THEN-ELSE*. These constructs — also called keywords —are used to describe the control flow of the algorithm.

1. **SEQUENCE** represents linear tasks sequentially performed one after the other.

2. **WHILE** a loop with a condition at its beginning.

3. **DO-WHILE** a loop with a condition at the bottom.

4. **FOR** another way of looping.

5. **IF-THEN-ELSE** a conditional statement changing the flow of the algorithm.

6. **CASE** the generalization form of IF-THEN-ELSE.

Exercises

1. Write pseudocode that reads two numbers and multiplies them together and print out their product.

```
READ $num1$, $num2$
SET product to $num1$* $num2$
Write product
```

2. Write pseudocode that tells a user that the number they entered is not a 5 or a 6.

```
READ isfive
IF (isfive = 5)
WRITE "your number is 5"
ELSE IF (isfive = 6)
WRITE "your number is 6"
ELSE
WRITE "your number is not 5 or 6"
END IF
```

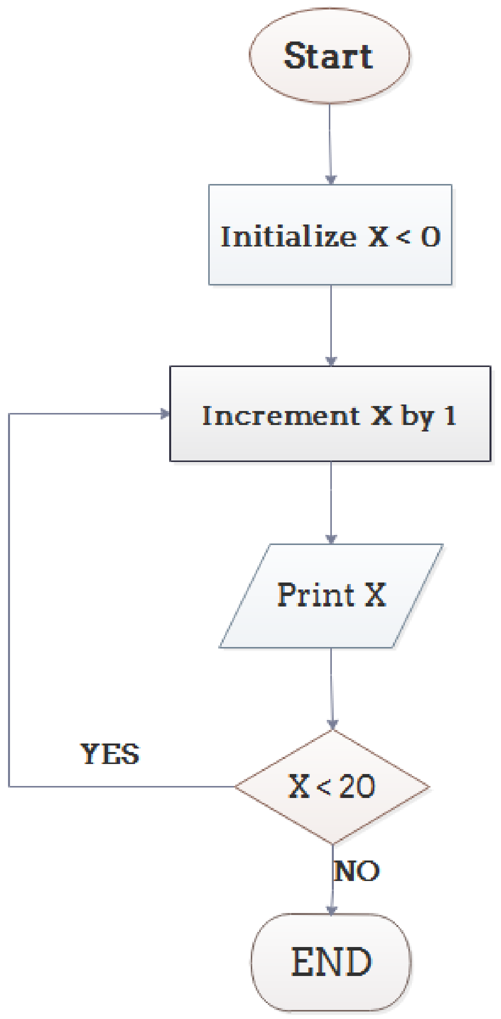
3. Write pseudocode to print all multiples of 5 between 1 and 100 (including both 1 and 100).

```
SET x to 1
WHILE (x < 20)
WRITE x
WRITE x = x*5
ADD 1 to x
END WHILE
```

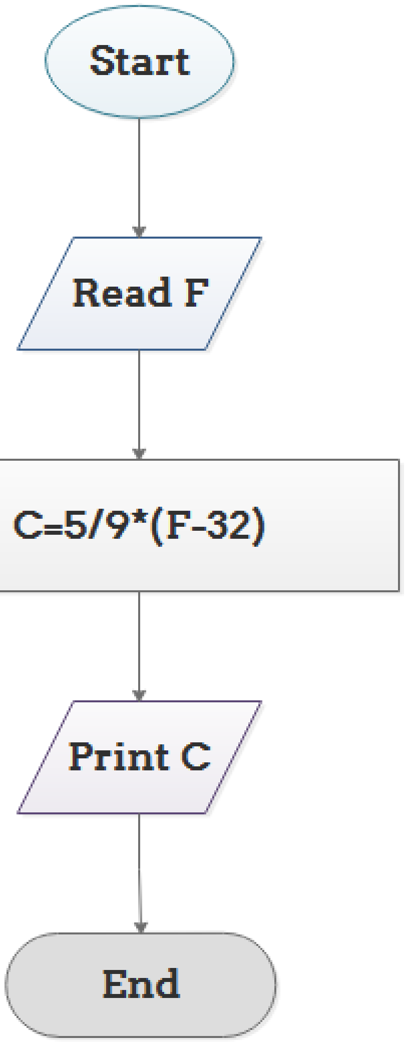
Algorithm (Pseudocode and Flowchart)

Algorithm: Print numbers from 1 to 20:

```
Step 1: Start
Step 2: Initialize X as 0
Step 3: Increment X by 1
Step 4: Print X
Step 5: If X is less than 20 then go back to step 3
Step 6: Stop
```



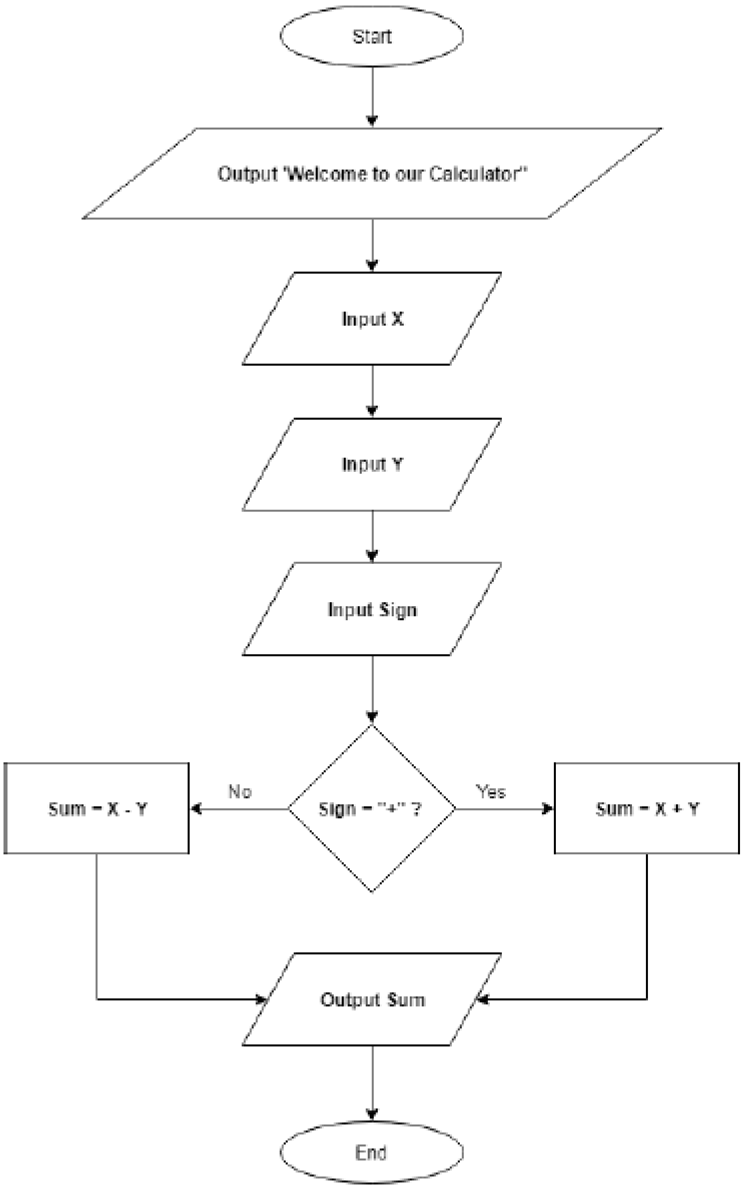
Convert Temperature from Fahrenheit (°F) to Celsius (°C)
Step 1: Start
Step 2: Read temperature in Fahrenheit
Step 2: Calculate temperature with formula C=5/9*(F-32)
Step 3: Print C
Step 4: Stop



Simple Calculator
The Pseudocode and Flowchart
Suppose that you are asked to write a calculator program that can sum and subtract two integer numbers. Write the program requirements, specifications, and algorithm.
The algorithm (pseudocode and flowchart):

```
1.print “Welcome to our Calculator”
2.X= input “Enter the first number:“
3.Y= input“Enter the second number:”
4.Sign= input "Select –or +"
5.if Signis equal to "+" then:
6.Sum= X+ Y
```

```
7.else:
8.Sum= X–Y
9.End if
10.print Sum
```



Verifying The Algorithm

1.Display a welcome message that says, “Welcome to our Calculator".

2.Ask the user to enter the first number and save it to X.

3.Ask the user to enter the second number and save it to Y.

4.Ask the user to select the sign (-,+) and save it in Sign.

5.if Signis equal to “+”, make Sum= X+ Y

6.Otherwise, make Sum= X-Y

7.Display Sum

Determining a Student’s Final Grade Algorithm

Write an algorithm and pseudocode to determine a student’s final grade and indicate whether it is passing or failing. The final grade is calculated as the average off our marks.

Algorithm:

1.Ask the user to enter 4 marks (Mark1, Mark2, Mark3, Mark4)

2.Calculate the marks average (Avg) by summing marks and it dividing by 4

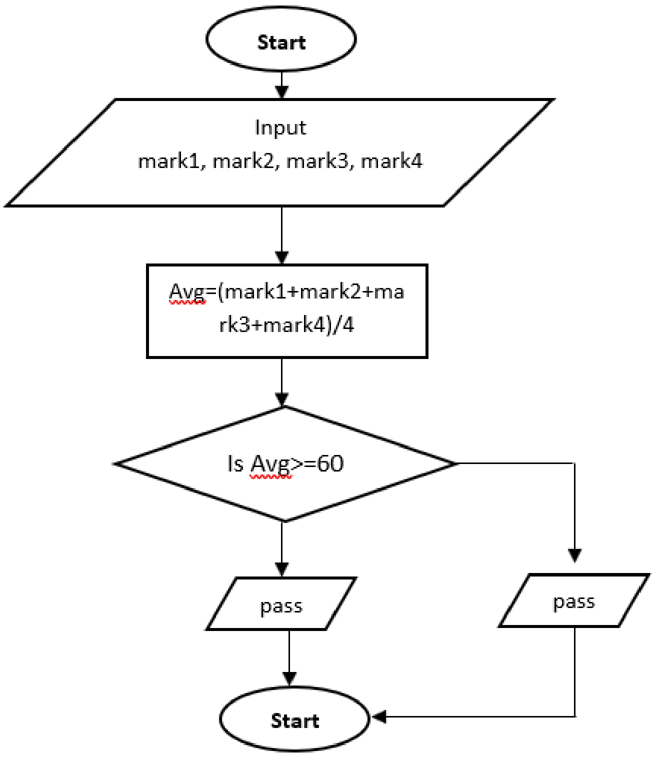
3.If average (Avg) is greater than or equal 60

4.Print “Pass”

5.Else

6.Print “Fail”

7.End if



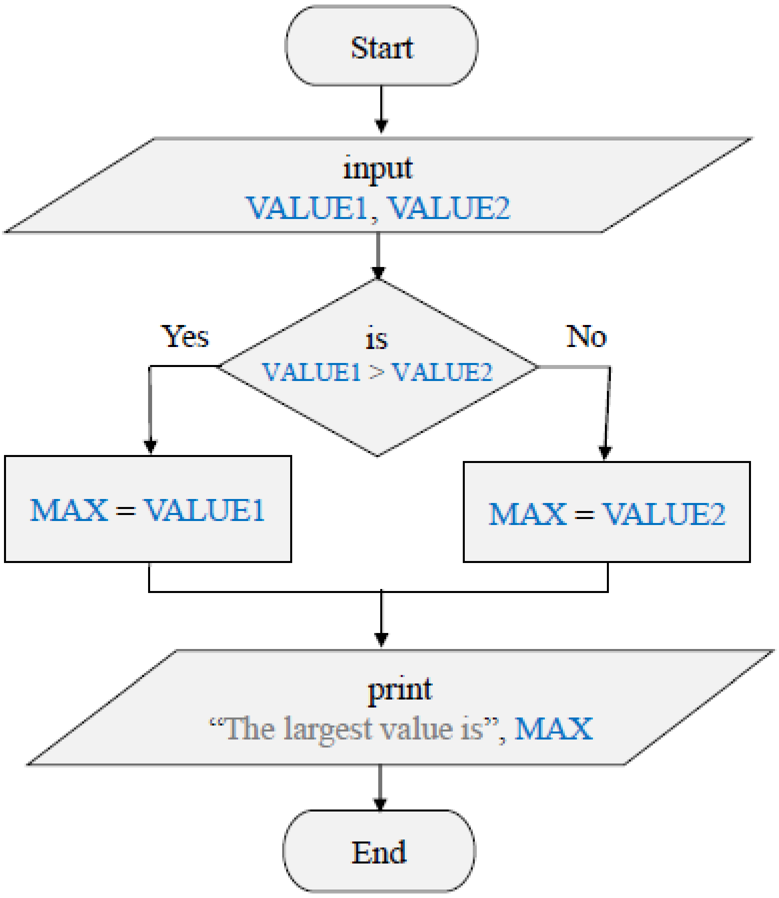
s

Determining The Largest Value Example

Write a Pseudocode that reads two values, determines the largest value and prints the largest value with an identifying message.

Pseudocode:

```
1.Input VALUE1, VALUE2
2.if (VALUE1> VALUE2) then
3.MAX=VALUE1
4.else
5.MAX=VALUE2
6.endif
7.print "The largest value is", MAX
```



End

Play & Learn
•Be familiar with basic logic and problem-solving techniques through practicing at Code.org.

•Visit https://studio.code.org/hoc/1and play.
How Do We Write a Program?

- A Computer is not intelligent.
- It cannot analyze a problem and come up with a solution.

A human (the programmer) must analyze the problem, develop the instructions for solving the problem, and then have the computer carry out the instructions.
To write a program for a computer to follow, we must go through a two-phase process: problem solving and implementation.

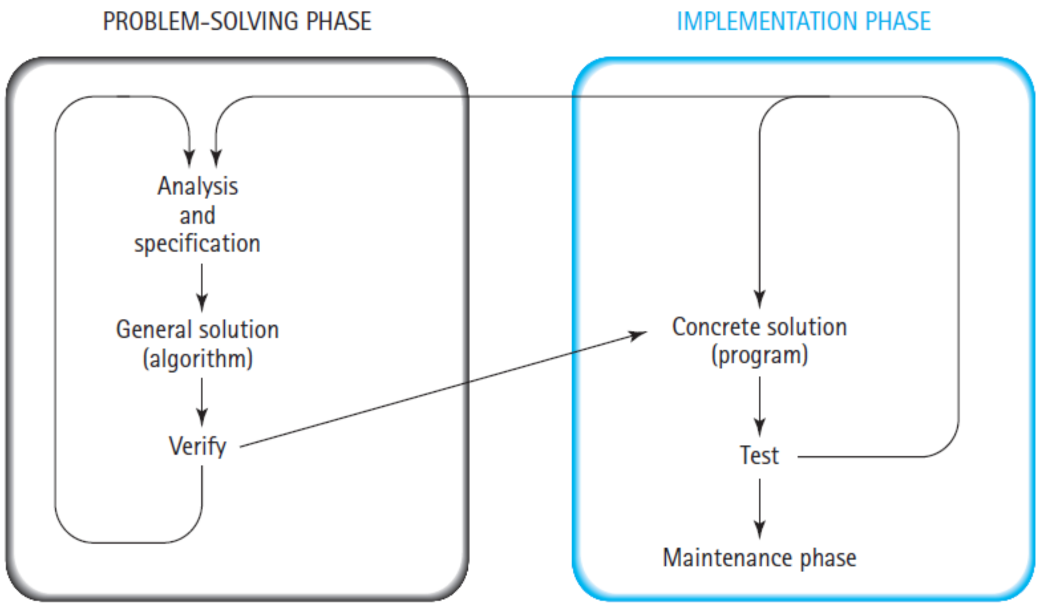

PROBLEM-SOLVING PHASE
IMPLEMENTATION PHASE
Analysis and specification
General solution (algorithm)
Verify
Concrete solution (program)
Test
Maintenance phase

Figure *Programming process*

Problem-Solving Phase
1. Analysis and Specification-Understand (define) the problem and what the solution must do.
2. General Solution (Algorithm)-Specify the required datatypes and the logical sequences of steps that solve the problem.
3. Verify-Follow the steps exactly to see if the solution really does solve the problem.
Implementation Phase

- **Concrete Solution (Program)** -Translate the algorithm (the general solution) into a programming language.
- **Test**-Have the computer followed the instructions.
- Then manually check the results.
- If you find errors, analyze the program and the algorithm to determine the source of the errors, and then make corrections.
- Once a program is tested, it enters into next phase (**Maintenance**).
- Maintenance requires modification of the program to meet changing requirements or to correct any errors that show up while using it.