

# Algorithm Development and Programming Fundamentals

---

# What is Programming?

---

# What is Programming?

It is a set of instructions given in specific manner to perform a specific task.

It can be of any type...

- Making Tea
- Cooking Rice
- Driving Car

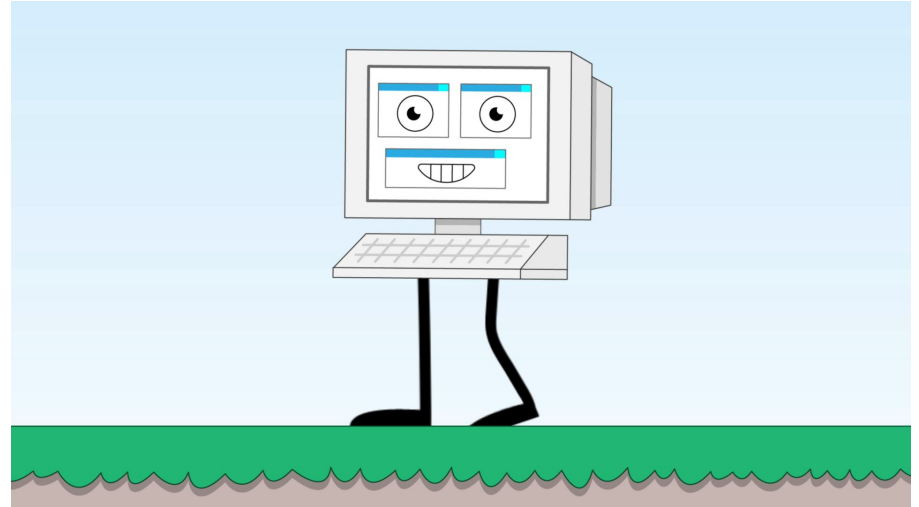
Instructions can also be given to computers.



# Computer Programming

Computer programming is the act of writing computer programs, which are a sequence of instructions written using a Computer Programming Language to perform a specified task by the computer.

- a. left foot forward
- b. right foot forward
- c. go back to instruction a



# What is Programming?

Programming is the process of taking an algorithm and encoding it into a notation, a programming language, so that it can be executed by a computer.

Although many programming languages and many different types of computers exist, the important first step is the need to have the solution.

Without an algorithm there can be no program.

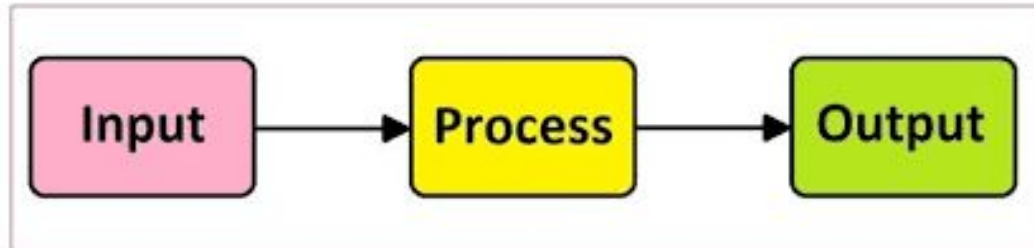
# Processing of Data

---

# What is Information Processing

The input–process–output (IPO) model is a widely used approach in systems analysis and software engineering for describing the structure of an information processing program or another process.

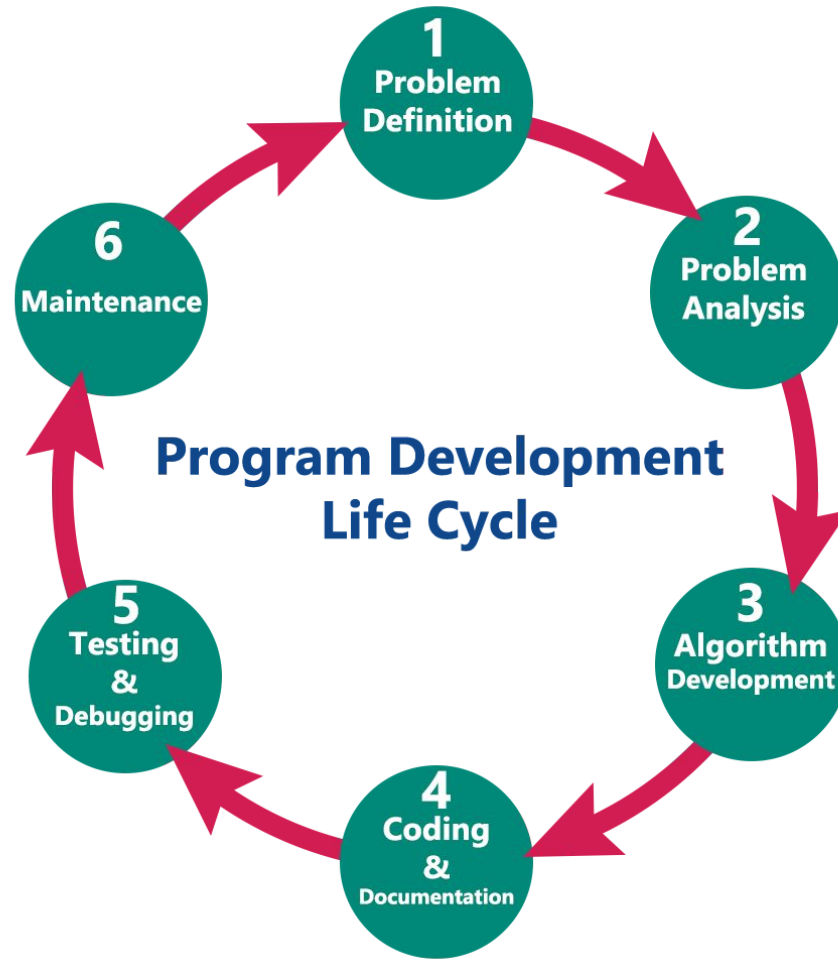
1. A requirement from the environment (input)
2. A computation based on the requirement (process)
3. A provision for the environment (output)



# Program Development Life Cycle

---





# PDLC

1. Analyze the problem.
2. Design the program.
3. Code the program.
4. Debug the program.

# Defining the Problem

---

# Problem Definition

In this phase, we define the problem statement and we decide the boundaries of the problem.

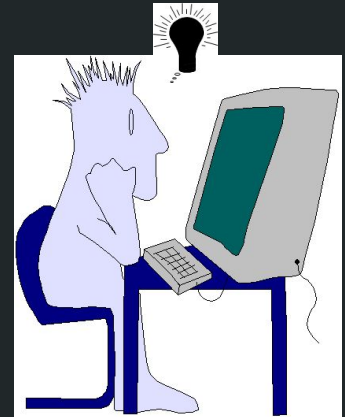
In this phase we need to understand the problem statement, what is our requirement, what should be the output of the problem solution.

These are defined in this first phase of the program development life cycle.

The program specification defines the data used in program, the processing that should take place while finding a solution, the format of the output and the user interface.

# Analyze the Problem

---



# Problem Analysis

In this phase we determine the requirements like variables, functions, etc. to solve the problem.

That means we gather the required resources to solve the problem defined in the problem definition phase.

We also determine the bounds of the solution.

# Algorithm Development

---

# Develop Algorithm

During this phase, we develop a step by step procedure to solve the problem using the specification given in the previous phase.

This phase is very important for program development.

That means we write the solution in step by step statements.



Program design starts by focusing on the main goal that the program is trying to achieve and then breaking the program into manageable components, each of which contributes to this goal.

This approach of program design is called top-bottom program design or modular programming.

The first step involve identifying main routine, which is the one of program's major activity. From that point, programmers try to divide the various components of the main routine into smaller parts called modules.

For each module, programmer draws a conceptual plan using an appropriate program design tool to visualize how the module will do its assign job.

# Tools

Structure Charts – A structure chart, also called Hierarchy chart, show top-down design of program. Each box in the structure chart indicates a task that program must accomplish. The Top module, called the Main module or Control module.

Algorithms – An algorithm is a step-by-step description of how to arrive at a solution in the most easiest way. Algorithms are not restricted to computer world only. In fact, we use them in everyday life.

Flowcharts – A flowchart is a diagram that shows the logic of the program.

# Tools

Decision tables – A Decision table is a special kind of table, which is divided into four parts by a pair of horizontal and vertical lines.

Pseudocode – A pseudocode is another tool to describe the way to arrive at a solution. They are different from algorithm by the fact that they are expressed in program language like constructs.

# Coding and Documentation

---

# Coding the Solution

This phase uses a programming language to write or implement the actual programming instructions for the steps defined in the previous phase.

In this phase, we construct the actual program.

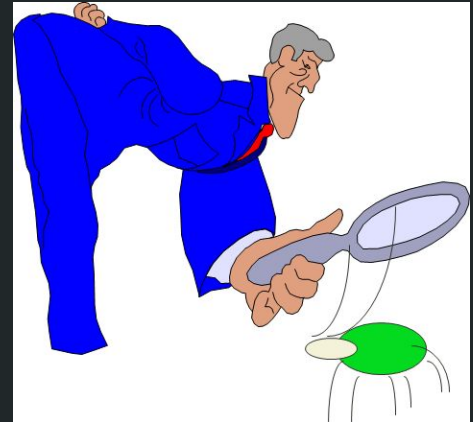
That means we write the program to solve the given problem using programming languages like C, C++, Java, etc.,

Coding the program means translating an algorithm into specific programming language.

The technique of programming using only well defined control structures is known as Structured programming.

# Test the Solution

---



# Testing

During this phase, we check whether the code written in the previous step is solving the specified problem or not.

That means we test the program whether it is solving the problem for various input data values or not.

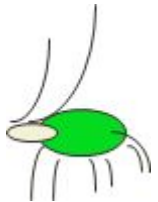
We also test whether it is providing the desired output or not.

After removal of syntax errors, the program will execute.

However, the output of the program may not be correct. This is because of logical error in the program.

A logical error is a mistake that the programmer made while designing the solution to a problem. So the programmer must find and correct logical errors by carefully examining the program output using Test data.

Syntax error and Logical error are collectively known as Bugs. The process of identifying errors and eliminating them is known as Debugging.





# Maintenance

---

# Maintenance

During this phase, the program is actively used by the users.

If any enhancements found in this phase, all the phases are to be repeated to make the enhancements.

That means in this phase, the solution (program) is used by the end-user. If the user encounters any problem or wants any enhancement, then we need to repeat all the phases from the starting, so that the encountered problem is solved or enhancement is added.

# Algorithms

---

# Introduction

The set of sequential steps developed to solve a problem is called its Algorithm.

And the implementation of algorithm in high computer programming language is called a Program.

Generally every algorithm is converted into a graphical format called Flow Chart before it is implemented as program.

The word algorithm comes from the word Al-khowarizmi, an Arabian inventor. It means recipe, method technique or procedure.

# Properties

An algorithm is a step-by-step problem solving procedure that can be carried out by a computer.

- It should be simple
- It should be clear with no ambiguity
- It should lead to unique solution of the problem
- It should involve a finite number of steps to arrive at a solution
- It should have the capability to handle some unexpected situations which may arise during the solution of a problem

# Characteristics/ Features

1. Input
  2. Process Method
  3. Finiteness
  4. Effective basic instructions
  5. Output
- 
- Adaptability on a computer
  - Simplicity of logic
  - Elegance
  - Efficiency with which it can be exacted

# Steps to develop algorithm

1. Clearly understand the problem statement.
2. Study the outputs to be generated
3. Design the process
4. Refine the process
5. Test the algorithm

# Advantages

1. Easy to understand as it is a step by step process
2. It has got a definite procedure
3. Convenient to develop an algorithm and then convert it a flow chart and eventually an program.
4. It has a start and end which simplifies the procedure
5. Easy to debug
6. Independent of programming languages



# Disadvantage




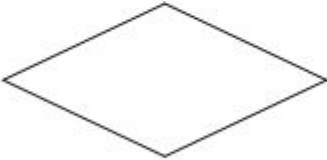


It is time consuming and cumbersome.

# Flow Charts

---

- A flowchart is a blueprint that pictorially represents the algorithm and its steps.
- A flowchart is a diagram that depicts a process, system or computer algorithm.
- Flowcharts use rectangles, ovals, diamonds and potentially numerous other shapes to define the type of step, along with connecting arrows to define flow and sequence.
- They can range from simple, hand-drawn charts to comprehensive computer-drawn diagrams depicting multiple steps and routes.
- Benefits of Flowchart are: Simplify the Logic, Better Communication, Effective Analysis, Useful in Coding, Testing, Documentation

# Symbols

Name	Symbol	Use in Flowchart
Oval		Denotes the beginning or end of the program
Parallelogram		Denotes an input operation
Rectangle		Denotes a process to be carried out e.g. addition, subtraction, division etc.
Diamond		Denotes a decision (or branch) to be made. The program should continue along one of two routes. (e.g. IF/THEN/ELSE)
Hybrid		Denotes an output operation
Flow line		Denotes the direction of logic flow in the program



Process symbol

Also known as an “Action Symbol,” this shape represents a process, action, or function. It’s the most widely-used symbol in flowcharting.



Start/End symbol

Also known as the “Terminator Symbol,” this symbol represents the start points, end points, and potential outcomes of a path. Often contains “Start” or “End” within the shape.



Document symbol

Represents the input or output of a document, specifically. Examples of an input are receiving a report, email, or order. Examples of an output using a document symbol include generating a presentation, memo, or letter.



Decision symbol

Indicates a question to be answered — usually yes/no or true/false. The flowchart path may then split off into different branches depending on the answer or consequences thereafter.



Connector symbol

Usually used within more complex charts, this symbol connects separate elements across one page.



Off-Page  
Connector/Link  
symbol

Frequently used within complex charts, this symbol connects separate elements across multiple pages with the page number usually placed on or within the shape for easy reference.



Input/Output  
symbol

Also referred to as the “Data Symbol,” this shape represents data that is available for input or output as well as representing resources used or generated. While the paper tape symbol also represents input/output, it is outdated and no longer in common use for flowchart diagramming.



Comment/Note  
symbol

Placed along with context, this symbol adds needed explanation or comments within the specified range. It may be connected by a dashed line to the relevant section of the flowchart as well.

# Pseudo code

---



- Pseudo-code is an informal way to express the design of a computer program or an algorithm.
- The aim is to get the idea quickly and also easy to read without syntactic details.
- Pseudocode is not a real language of programming.
- It can not thus be compiled into a software executable.
- It is used to construct code for programmes in brief phrases or basic English syntaxes prior to it becoming a certain programming language.
- This is done to identify the flow errors in the highest possible level and comprehend the data flows that will be used by the final software.

- This helps undoubtedly to save time when conceptual problems are already fixed during real programming.
- To get the desired results for a program, you first gather the program description and functionality, then use pseudo code for creating statements.
- Detailed pseudocode is evaluated and validated according to design standards by the design team or programmers.
- Catching faults or incorrect program flows during the phase of pseudo code creation is useful since it is cheaper than catching them afterwards.
- Once the team accepts the pseudo code, it is written in a programming language using vocabulary and grammar.

# Examples

---

# Find area of a rectangle - Algorithm

Step 1: Start

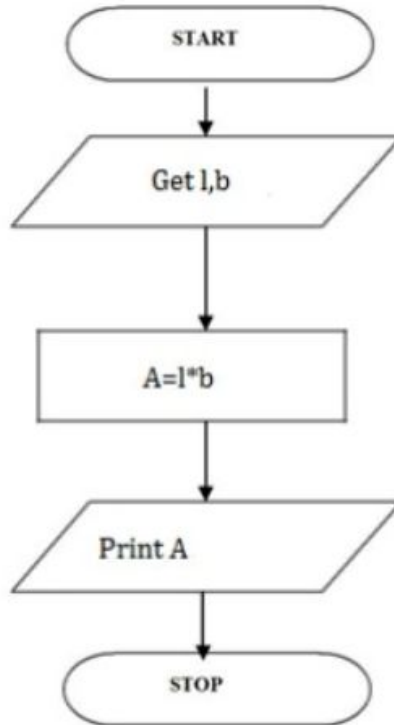
Step 2: get l,b values

Step 3: Calculate  $A=l*b$

Step 4: Display A

Step 5: Stop

# Find area of a rectangle - Flowchart



# Find area of a rectangle - Psuedocode

BEGIN

READ  $l, b$

CALCULATE  $A = l * b$

DISPLAY  $A$

END

# Greatest of two numbers - Algorithm

Step 1: Start

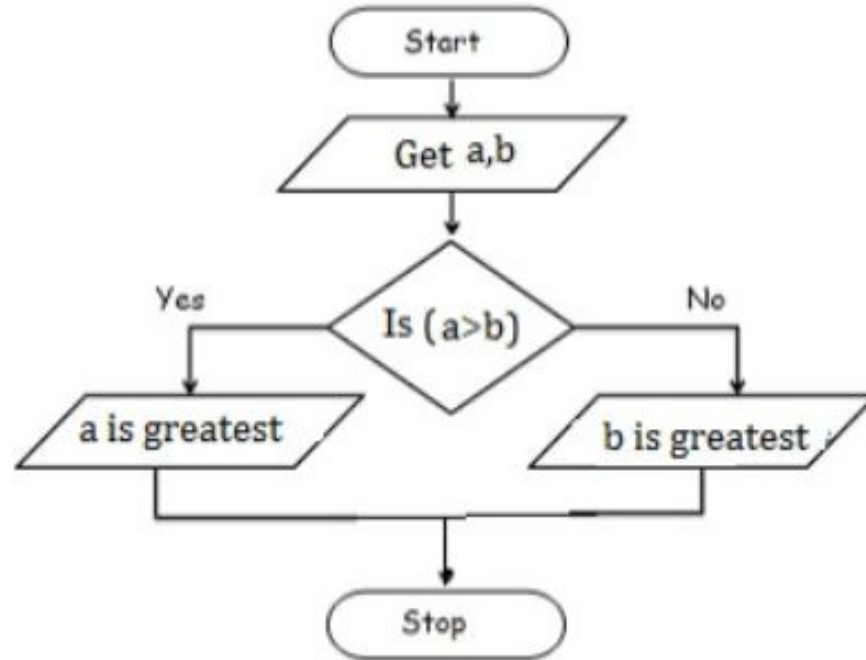
Step 2: get a,b value

Step 3: check if( $a > b$ ) print a is greater

Step 4: else b is greater

Step 5: Stop

# Greatest of two numbers - Flowchart





# Greatest of two numbers - Psuedocode

```
BEGIN  
  READ a,b  
  IF (a>b) THEN  
    DISPLAY a is greater  
  ELSE  
    DISPLAY b is greater  
  END IF  
END
```

# An Algorithm Development Process

---

# Introduction

- Every problem solution starts with a plan. That plan is called an algorithm.
- An algorithm is a plan for solving a problem.
- There are many ways to write an algorithm. Some are very informal, some are quite formal and mathematical in nature, and some are quite graphical.
  - The instructions for connecting a DVD player to a television are an algorithm.
  - A mathematical formula such as  $\pi R^2$  is a special case of an algorithm.
- The form is not particularly important as long as it provides a good way to describe and check the logic of the plan

# Process

- The development of an algorithm (a plan) is a key step in solving a problem.
- Once we have an algorithm, we can translate it into a computer program in some programming language.
- Our algorithm development process consists of five major steps.

Step 1: Obtain a description of the problem.

Step 2: Analyze the problem.

Step 3: Develop a high-level algorithm.

Step 4: Refine the algorithm by adding more detail.

Step 5: Review the algorithm.

Obtain a description of the problem.

- This step is much more difficult than it appears.
- The developer must create an algorithm that will solve the client's problem.
- The client is responsible for creating a description of the problem, but this is often the weakest part of the process.
- It's quite common for a problem description to suffer from one or more of the following types of defects:

(1) the description relies on unstated assumptions,

(2) the description is ambiguous,

(3) the description is incomplete, or

(4) the description has internal contradictions.

- These defects are seldom due to carelessness by the client. Instead, they are due to the fact that natural languages (English, French, etc.) are rather imprecise.
- Part of the developer's responsibility is to identify defects in the description of a problem, and to work with the client to remedy those defects.

Analyze the problem

- The purpose of this step is to determine both the starting and ending points for solving the problem. This process is analogous to a mathematician determining what is given and what must be proven. A good problem description makes it easier to perform this step.
- When determining the starting point, we should start by seeking answers to the following questions:
  - What data are available?
  - Where is that data?
  - What formulas pertain to the problem?
  - What rules exist for working with the data?
  - What relationships exist among the data values?



- When determining the ending point, we need to describe the characteristics of a solution. In other words, how will we know when we're done? Asking the following questions often helps to determine the ending point.
  - What new facts will we have?
  - What items will have changed?
  - What changes will have been made to those items?
  - What things will no longer exist?

Develop a high-level  
algorithm

- An algorithm is a plan for solving a problem, but plans come in several levels of detail.
- It's usually better to start with a high-level algorithm that includes the major part of a solution, but leaves the details until later.
- We can use an everyday example to demonstrate a high-level algorithm.

**Problem:** I need a send a birthday card to my brother, Harry.

**Analysis:** I don't have a card. I prefer to buy a card rather than make one myself.

**High-level algorithm:**

- Go to a store that sells greeting cards
- Select a card
- Purchase a card
- Mail the card

- This algorithm is satisfactory for daily use, but it lacks details that would have to be added were a computer to carry out the solution.
- These details include answers to questions such as the following.
  - "Which store will I visit?"
  - "How will I get there: walk, drive, ride my bicycle, take the bus?"
  - "What kind of card does Harry like: humorous, sentimental, risqué?"

These kinds of details are considered in the next step of our process.

Refine the algorithm  
by adding more  
detail

- A high-level algorithm shows the major steps that need to be followed to solve a problem.
- Now we need to add details to these steps, but how much detail should we add? Unfortunately, the answer to this question depends on the situation.
- We have to consider who (or what) is going to implement the algorithm and how much that person (or thing) already knows how to do.
- If someone is going to purchase Harry's birthday card on my behalf, my instructions have to be adapted to whether or not that person is familiar with the stores in the community and how well the purchaser known my brother's taste in greeting cards.

- When our goal is to develop algorithms that will lead to computer programs, we need to consider the capabilities of the computer and provide enough detail so that someone else could use our algorithm to write a computer program that follows the steps in our algorithm.
- As with the birthday card problem, we need to adjust the level of detail to match the ability of the programmer.
- When in doubt, or when you are learning, it is better to have too much detail than to have too little.

- Most examples will move from a high-level to a detailed algorithm in a single step, but this is not always reasonable.
- For larger, more complex problems, it is common to go through this process several times, developing intermediate level algorithms as we go.
- Each time, we add more detail to the previous algorithm, stopping when we see no benefit to further refinement.
- This technique of gradually working from a high-level to a detailed algorithm is often called stepwise refinement.
- **Stepwise refinement** is a process for developing a detailed algorithm by gradually adding detail to a high-level algorithm.



Review the  
algorithm.

- The final step is to review the algorithm.
  - What are we looking for?
- First, we need to work through the algorithm step by step to determine whether or not it will solve the original problem.
- Once we are satisfied that the algorithm does provide a solution to the problem, we start to look for other things.
- The following questions are typical of ones that should be asked whenever we review an algorithm.
- Asking these questions and seeking their answers is a good way to develop skills that can be applied to the next problem.

# Questions

- Does this algorithm solve a very specific problem or does it solve a more general problem? If it solves a very specific problem, should it be generalized?
  - For example, an algorithm that computes the area of a circle having radius 5.2 meters (formula  $\pi * 5.2^2$ ) solves a very specific problem, but an algorithm that computes the area of any circle (formula  $\pi * R^2$ ) solves a more general problem.
- Can this algorithm be simplified?
  - One formula for computing the perimeter of a rectangle is:
    - length + width + length + width
  - A simpler formula would be:
    - $2.0 * (\text{length} + \text{width})$

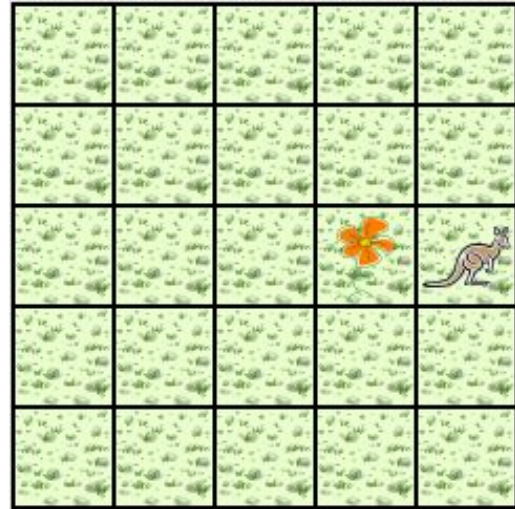
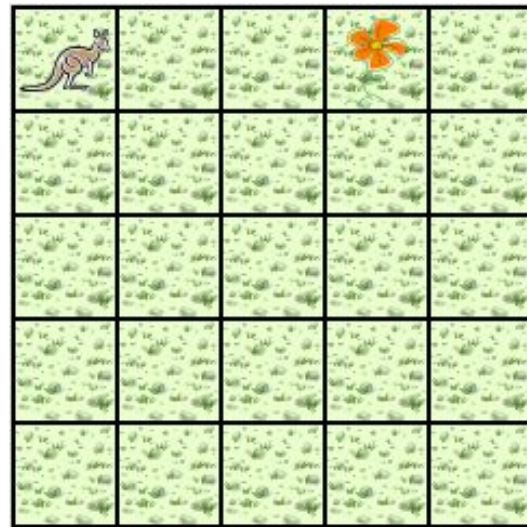
# Questions

- Is this solution similar to the solution to another problem? How are they alike? How are they different?
  - For example, consider the following two formulae:
    - Rectangle area = length \* width
    - Triangle area = 0.5 \* base \* height
  - **Similarities:** Each computes an area. Each multiplies two measurements.
  - **Differences:** Different measurements are used. The triangle formula contains 0.5.
  - **Hypothesis:** Every area formula involves multiplying two measurements.

Example :  
Pick and Plant

# Problem Statement (Step 1)

- A Jeroo starts at (0, 0) facing East with no flowers in its pouch. There is a flower at location (3, 0). Write a program that directs the Jeroo to pick the flower and plant it at location (3, 2). After planting the flower, the Jeroo should hop one space East and stop. There are no other nets, flowers, or Jeroos on the island.



## Analysis of the Problem (Step 2)

- The flower is exactly three spaces ahead of the jeroo.
- The flower is to be planted exactly two spaces South of its current location.
- The Jeroo is to finish facing East one space East of the planted flower.
- There are no nets to worry about.

## High-level Algorithm (Step 3)

Jeroo should do the following:

- Get the flower
- Put the flower
- Hop East



# Detailed Algorithm (Step 4)

Jeroo should do the following:

- Get the flower
  - Hop 3 times
  - Pick the flower
- Put the flower
  - Turn right Hop 2 times Plant a flower
- Hop East
  - Turn left Hop once

## Review the Algorithm (Step 5)

- The high-level algorithm partitioned the problem into three rather easy subproblems. This seems like a good technique.
- This algorithm solves a very specific problem because the Jeroo and the flower are in very specific locations.
- This algorithm is actually a solution to a slightly more general problem in which the Jeroo starts anywhere, and the flower is 3 spaces directly ahead of the Jeroo.

# Programming the solution

- A good programmer doesn't write a program all at once.
- Instead, the programmer will write and test the program in a series of builds.
- Each build adds to the previous one.
- The high-level algorithm will guide us in this process.

# Programming the solution

- The recommended first build contains three things:
- The main method
- Declaration and instantiation of Jeroo that will be used.
- The high-level algorithm in the form of comments.

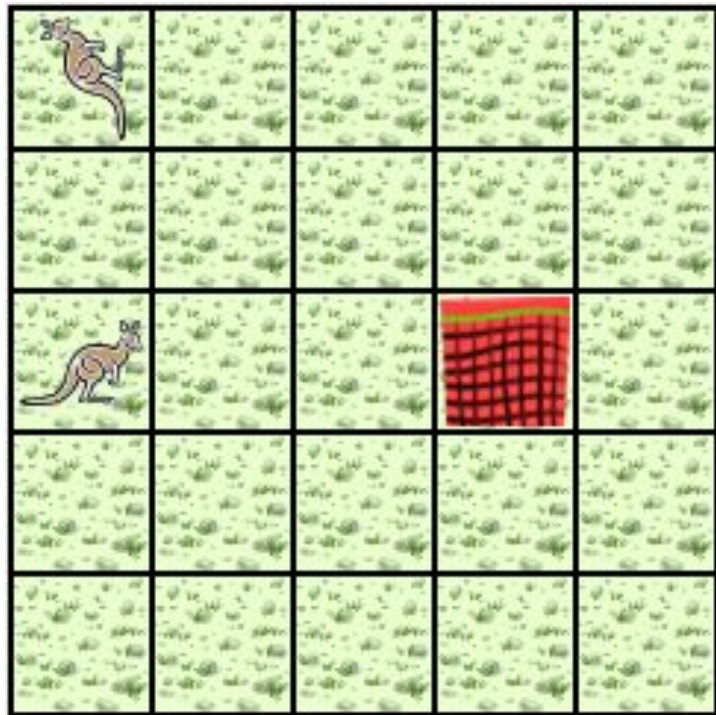
```
int main()
{
    //Jeroo initialization
    // --- Get the flower ---
    // --- Put the flower ---
    // --- Hop East ---
}    // ===== end of method =====
```

Example:  
Replace Net with  
Flower

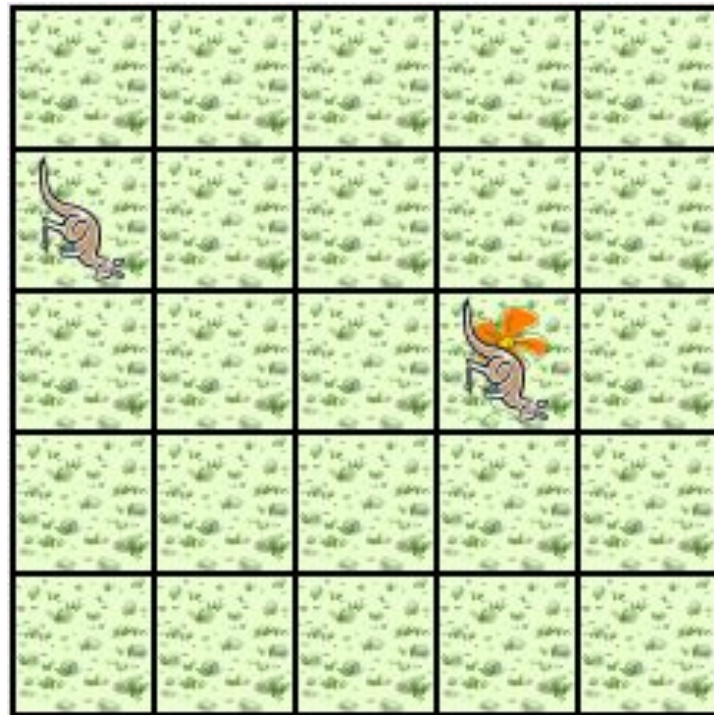
# Problem Statement (Step 1)

- There are two Jeroos. One Jeroo starts at  $(0, 0)$  facing North with one flower in its pouch.
- The second starts at  $(0, 2)$  facing East with one flower in its pouch.
- There is a net at location  $(3, 2)$ .
- Write a program that directs the first Jeroo to give its flower to the second one. After receiving the flower, the second Jeroo must disable the net, and plant a flower in its place. After planting the flower, the Jeroo must turn and face South. There are no other nets, flowers, or Jeroos on the island.

Start



Finish



## Analysis of the Problem (Step 2)

- Jeroo\_2 is exactly two spaces behind Jeroo\_1.
- The only net is exactly three spaces ahead of Jeroo\_2.
- Each Jeroo has exactly one flower.
- Jeroo\_2 will have two flowers after receiving one from Jeroo\_1.
  - One flower must be used to disable the net.
  - The other flower must be planted at the location of the net, i.e. (3, 2).
- Jeroo\_1 will finish at (0, 1) facing South.
- Jeroo\_2 is to finish at (3, 2) facing South.
- Each Jeroo will finish with 0 flowers in its pouch. One flower was used to disable the net, and the other was planted.



## High-level Algorithm (Step 3)

Let's name the first Jeroo Ann and the second one Andy.

- Ann should do the following:
  - Find Andy (but don't collide with him)
  - Give a flower to Andy (he will be straight ahead)
- After receiving the flower, Andy should do the following:
  - Find the net (but don't hop onto it)
  - Disable the net
  - Plant a flower at the location of the net
  - Face South

# Detailed Algorithm (Step 4)

- Let's name the first Jeroo Ann and the second one Andy.
- Ann should do the following:
  - Find Andy
    - Turn around (either left or right twice)
    - Hop (to location (0, 1))
  - Give a flower to Andy
    - Give ahead
- Now Andy should do the following:
  - Find the net
    - Hop twice (to location (2, 2))
  - Disable the net
    - Toss
  - Plant a flower at the location of the net
    - Hop (to location (3, 2))
    - Plant a flower
  - Face South
    - Turn right

## Review the Algorithm (Step 5)

- The high-level algorithm helps manage the details.
- This algorithm solves a very specific problem, but the specific locations are not important.
- The only thing that is important is the starting location of the Jeroos relative to one another and the location of the net relative to the second Jeroo's location and direction.