Middle School
Unpacking

Algorithms

2-AP-10: Use flowcharts and/or pseudocode to address complex problems as algorithms.

Authors:
Satabdi Basu, Daisy Rutstein, Arif Rachmatullah, Hui Yang & Carol Tate

Suggested Citation

Basu, S., Rutstein, D., Rachmatullah, A., Yang, H., & Tate, C. (2025). *Unpacking 2-AP-10: Use flowcharts and/or pseudocode to address complex problems as algorithms*. SRI International.

Acknowledgements

We sincerely thank the teachers and students who participated in the study to test and refine this material. We also extend our gratitude to Andrea Beesley, Christopher Ortiz, Cris Jimenez, Eliese Rulifson, and Nonye Alozie as well as the Learning Partnerships team for their invaluable support and feedback on the earlier version of this material. This material is based upon work supported by the National Science Foundation under Grant No. DRL-2010591. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Unpacking 2-AP-10: Algorithms

Table of Contents

Language from the CSTA Standard

2-AP-10: Use flowcharts and/or pseudocode to address complex problems as algorithms.

Complex problems are problems that would be difficult for students to solve computationally. Students should use pseudocode and/or flowcharts to organize and sequence an algorithm that addresses a complex problem, even though they may not actually program the solutions. For example, students might express an algorithm that produces a recommendation for purchasing sneakers based on inputs such as size, colors, brand, comfort, and cost. Testing the algorithm with a wide range of inputs and users allows students to refine their recommendation algorithm and to identify other inputs they may have initially excluded.

Practice(s): Developing and Using Abstractions

Concept: Algorithms & Programming

Subconcept: Algorithms

I. Defining the Scope of the Standard

2-AP-10 is a middle school standard in the "Algorithms and Programming (AP)" strand that requires students to be able to represent complex problems as algorithms using flowcharts and/or pseudocode. Complex problems are hard for students to program and solve computationally, so an initial step towards solving them involves articulating a plan or a set of steps to solve the problem.

An algorithm is a set of well-defined, precise, ordered instructions to solve a problem or perform a calculation. This standard includes expressing an algorithm using informal representations, such as flowcharts and pseudocode (plain language description of the steps in an algorithm), identifying relevant information in a problem to use in formulating an algorithm (e.g., inputs, goals, and decision points), and testing and debugging algorithms. Additional details about flowcharts and pseudocode are provided in Section III of this document.

This standard transitions students from being able to interpret and compare algorithms at the upper elementary level to being able to create and test algorithms that address problems at the middle school level. At this level, the standard does not require students to be able to transform an algorithm to a program or a prototype, which is the focus in high school.

Prerequisite computer science (CS) standard for 2-AP-10:

1B-AP-08 (Grade 3–5): Compare and refine multiple algorithms for the same task and determine which is the most appropriate.

This upper-elementary standard comes before 2-AP-10 and requires students to be able to look at different ways to solve the same task and evaluate the best solution. For example, students could compare two sets of directions for getting ready for school and determine which is the most efficient.

Upper boundary for 2-AP-10:

3A-AP-13 (Grade 9–10): Create prototypes that use algorithms to solve computational problems by leveraging prior student knowledge and personal interests.

This standard comes after 2-AP-10 and emphasizes the importance of students' creating a prototype, which is a computational artifact that represents the focal functionality of a product. Designing a prototype during the early stage of the creation of computational artifacts can help students get early feedback to reflect on the feasibility of a product.

II. Learning Targets for 2-AP-10

The broad middle school 2-AP-10 standard can be decomposed into the following fine-grained learning targets (LTs):

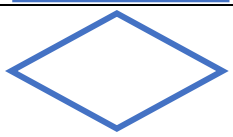
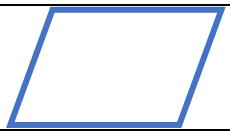
1. Knowledge that an algorithm is a step-by-step, ordered set of instructions for solving a problem, and in order to be computer-understandable, the instructions must be precise and unambiguous. (Note: This learning target reflects the upper-elementary standard for algorithms but may need to be revisited in middle school as well.)
2. Knowledge that pseudocode is an informal way to describe code without strictly following any specific programming language and can be used to plan out code before programming.
3. Knowledge that a flowchart is a diagrammatic representation of an algorithm that specifies a step-by-step way to complete a task.
4. Ability to trace an algorithm (in the form of pseudocode or a flowchart) and describe its behaviour or output when given a specific set of inputs.
5. Ability to recognize relevant information from a problem to identify possible inputs and goals of an algorithm, decision points that may require branching, test cases, stopping conditions, and constraints such as cost, time, or delivery platform.
6. Ability to select and/or create appropriate representations (pseudocode or flowchart) to plan a problem solution that handles the desired range of inputs and is able to deal with edge cases.
7. Ability to generate multiple algorithms (flowchart or pseudocode) to solve a problem.
8. Ability to compare the trade-offs between different algorithms or approaches to problem solving based on certain evaluation criteria or constraints.
9. Ability to identify meaningful test cases (including edge cases) for testing an algorithm.
10. Ability to test and debug algorithms using a systematic and iterative process to ensure the algorithms function appropriately.

III. Flowchart and Pseudocode Components

1. Flowchart Components

Flowcharts are a way for a user to visualize how a program works. Using some basic shapes that indicate the type of action the program takes, users can create a path that goes through the entirety of the program. With lines connecting each process, it is simple to follow the flow of the program from start to finish, even when the program has dynamic pieces that can alter the output.

The following information depicting flowchart shapes and their uses has been taken from "2.3 Flowcharts. *Fundamentals of Engineering Programming with C and Fortran*. Cambridge University Press. pp. 32–36. ISBN 978-0-521-62950-8."

Flowchart Shape	Name	When to Use
	Terminal	To denote the start and end of a program or process. Usually just contains the words "start" or "end."
	Process	Represents a process that changes or updates data.
	Decision	This is equivalent to a conditional statement. It always leads to multiple flowlines running from it based on the data entered.
	Input/Output	This indicates that data is being entered or displayed.
	Flowline	Acts as the path between each node. It indicates where the program will go next at each step.

2. Pseudocode Components

Pseudocode helps bridge the gap between an idea and its implementation in code. It includes the concepts of programming but is written in a general enough way to be easily implemented in whatever coding language is needed. The fact that it resembles actual code makes planning a program much simpler. However, because it is a planning tool, there is no single "correct" way to write it. As long as it is not overly abstract and follows common control structures, pseudocode can be a useful tool for any coder. Below is a table of common keywords that one may run into when looking at pseudocode. While it is not exhaustive, it includes the most common terms and provides good context for pseudocode in general. Note that many of the terms can be interchanged depending on preference.

Code Component	Pseudocode Wording	When to Use	Examples
A section of code	Program	When denoting the start of the program, not just a piece of it.	Begin **Program** … End **Program**
	Procedure, Function, Process	When denoting a subroutine or a piece of a program	Begin **Procedure** … End **Procedure**

Code component	Pseudocode wording	When to use	Examples

Loop	For	When running a section of code multiple times.	Set i to 0
			Begin **for** i less than 5
			  Print i
			  Set i to i + 1
			End **for**

	For each	When iterating through items in a list/array.	Begin **for each** item in bag Print item End **for each**

Conditional	If – Then – Else	When basing an action on a condition or decision.	Begin **If** x is less than 5
			  Begin **Then**
			  Print “x < 5”
			  End **Then**
			  Begin **Else**
			  Print “x >= 5”
			  End **Else**
			End **If**

	AND, OR	When basing an action on a combination of conditions (all conditions need to be met or any one of the conditions need to be met).	Begin If x is less than 10 **AND** x is greater than 5
			  Begin Then
			  Print “5 < x < 10”
			  End Then
			End If

	While	This denotes a loop but requires a condition to be evaluated to specify how many times the loop will iterate.	Begin **While** x is less than 5 … End **While**
Values	Set	When you want to clarify that a variable is being intentionally set to a value.	**Set** student to “Lauren” **Set** grade to 90 **Set** passed to True

	Return	To show when another process or procedure results in a new value passed back to the main program.	Begin Procedure
			  Set j to 10
			  Return j
			End Procedure

Code component	Pseudocode wording	When to use	Examples
Comments	A # sign or a // sign is used to denote comments.	To describe the function of one or more lines of pseudocode.	// gameWon is a variable that starts with a value of 0 and turns 1 when the user wins the game.
Other miscellaneous	Begin	Can be used before any of the previous keywords to give more clarity on what is happening.	**Begin** Program **Begin** If **Begin** For
	End	Used after another keyword process has ended. This also is also used to denote the end of the program or process.	**End** Program **End** If **End** For
	Using tabs or spaces	In many coding languages, code within certain structures is indented as a way to visually organize code. The same process is used when writing pseudocode.	Set k to 0 Begin Procedure _Begin For k less than 5 ___Print k _End For _Return k End Procedure

IV. Examples of Algorithms for Different Scenarios

Below, we illustrate what proficiency on the 2-AP-10 standard looks like by presenting examples of algorithms in the form of pseudocode and flowcharts for different scenarios. First, we present a generic template to help design algorithms and then demonstrate how to apply the template to come up with algorithms for the different scenarios. This process helps students who already understand the definition of algorithms transition to creating and generating algorithms specified in middle school.

Template for designing algorithms

Given: problem description or scenario

1. Establish the desired outcome for the program/algorithm.
2. Consider whether the problem scenario, as presented, contains sufficient information to understand the problem. If not, find ways to fill in missing information.
3. Determine what inputs are provided in the problem statement.
4. Determine what inputs need to come from the user.
5. Find a path from the inputs to desired outcomes.
    a. Identify characters and/or objects that need to be programmed.
    b. Identify data structures and variables that need to be programmed (including properties of characters that can change in the program).
    c. Identify behaviors or actions of characters and/or objects that need to be programmed.

d. Identify interactions among characters and/or objects.
e. Identify sequence of actions—which actions happen in order, which happen in parallel, determine repeating actions or action sequences (loops), determine decision points where branching occurs.

Example 1 – Scenario: Create a sneaker recommendation app

Given problem description: You are asked to create an app that produces a recommendation for purchasing sneakers based on inputs such as size, color, brand, comfort, and cost.

Applying the template to design the algorithm:

1. Desired outcome: User will receive a recommendation for a pair of sneakers based on their inputs about preferred size, color, brand, comfort, and maximum affordable cost.
2. Additional information needed to solve the problem: Acceptable values for user inputs needed if input will be validated.
3. Input variable values provided by the problem: None.
4. Input variable values provided by the user:
   User_size, *User_color*, *User_brand*, *User_comfort*, *User_cost*
5. Path from inputs to desired outcome:
   a. Characters/Objects to be programmed: A single character needs to be programmed – the sneaker selector.
   b. Required data structures: A list of available sneakers, each with an assigned size, color, brand, cost, and comfort level.
      Required variables: Input variables described above in 4.
      Properties of characters that can change: None.
   c. Character behaviors/actions: Sneaker selector recommends sneakers based on whether sneaker properties match user preferences.
   d. Character interactions: Only 1 character, therefore no interactions.
      Sequence of actions including repeating actions and decision points: The sneaker selector selects a sneaker from the list of available sneakers. If the size, color, brand, comfort, and cost of the selected sneaker matches the user's desired input values of *User_size*, *User_color*, *User_brand*, *User_comfort*, and *User_cost*, the sneaker is recommended. Otherwise, the next sneaker in the list is selected, and this process is repeated until there are no more sneakers available.

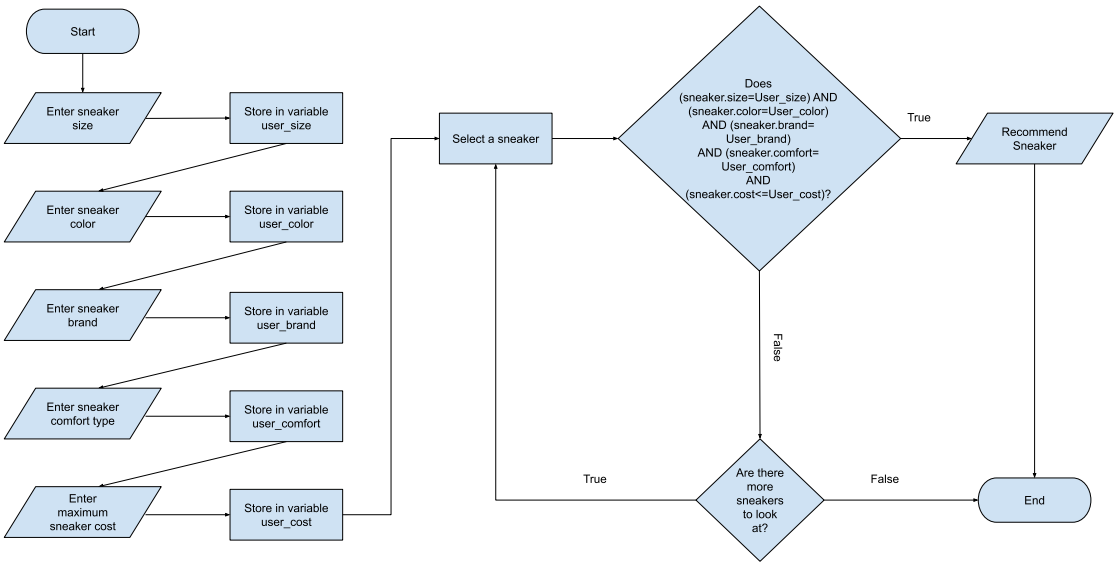
Representing the algorithm as pseudocode and as a flowchart:

Pseudocode

1. Ask user to enter preferred sneaker size and store in variable User_size
2. Ask user to enter preferred sneaker color and store in variable User_color
3. Ask user to enter preferred sneaker brand and store in variable User_brand
4. Ask user to enter preferred sneaker comfort type and store in variable User_comfort
5. Ask user to enter maximum affordable sneaker cost and store in variable User_cost
6. While (There are more sneakers to look at)
    a. Select a sneaker
    b. If `(sneaker.size = User_size) AND (sneaker.color = User_color) AND (sneaker.brand = User_brand) AND (sneaker.comfort = User_comfort) AND ((sneaker.cost < User_cost) OR (sneaker.cost = User_cost))`
    Then (Recommend sneaker)
    Otherwise (Move on to look at next sneaker)
    End If

End While

Flowchart



Example 2 – Scenario: Create a cat and mouse chase game

Given problem description: You are given a 30x30 grid (x=0–30, y=0–30) and asked to program a game with two characters – a cat and a mouse. If the cat moves into the same space as the mouse, the cat wins and the game ends. If the mouse gets to the door on the upper right corner of the grid (x=30, y=30), the mouse wins and the game ends. The mouse will start the game at the lower left corner (x=0, y=0) of the grid, while the cat will start at a random location on the grid. The computer generates the cat's location randomly at regular intervals. The mouse's location is controlled via user inputs. For example, pressing the left arrow key moves the mouse to the square on the left in the grid, while pressing the down arrow key moves the mouse to the square below in the grid.

Applying the template to design the algorithm:

1. Desired outcome: The user will be able to play a game where the user can control the movement of a mouse through a grid structure using keyboard arrow keys to make the mouse reach a given door without encountering the cat.
2. Additional information needed to solve the problem: No additional information is required, but additional information can be provided about how the program should behave if the mouse's x or y position becomes less than 0 or more than 30.
3. Input variable values provided by the problem:
   Mouse_Xposition, *Mouse_Yposition*, *Door_Xposition*, *Door_Yposition*
4. Input variable values provided by the user:
   Direction of mouse's movement (keyboard input).
5. Path from inputs to desired outcome:
   a. Characters/Agents to be programmed: Mouse, Cat.
   b. Required variables: Input variables described above in Step 3, *Cat_Xposition*, *Cat_Yposition*, a Boolean variable (value = TRUE or FALSE) called *Game_Running*.
      Required data structures: None.
      Properties of characters that can change: *Mouse_Xposition*, *Mouse_Yposition*, *Cat_Xposition*, *Cat_Yposition*.
   c. Character behaviors/actions:
      - Mouse moves in the direction of the arrow key being pressed.
      - Cat moves based on random location generated by the program.
   d. Character interactions:
      - If Mouse and Cat are at the same position, the cat wins and the game is over.

      Sequence of actions including repeating actions and decision points: The initial positions of the cat, mouse, and door are set up. Then, the following set of actions are repeated until either the cat or the mouse wins the game – the positions of the cat and the mouse are updated, and checks are made to see if the cat or mouse has won the game. If the cat and mouse are in the same location, the cat wins and the game ends. If the mouse reaches the door, the mouse wins and the game ends. Otherwise, the game continues with the position of the cat and mouse getting updated.

Representing the algorithm as a pseudocode and as a flowchart:

Pseudocode

```
#Setup the game
Show 30x30 grid;
Set Mouse_Xposition to 0
Set Mouse_Yposition to 0
Set Cat_Xposition to random (0, 30)
Set Cat_Yposition to random (0, 30)
Set Door_Xposition to 30
Set Door_Yposition to 30
Set Game_Running to TRUE

While (Game_Running = TRUE) {
  # Move the mouse and the cat to new location
  If(userPressed [left_arrow]}{
     Mouse_Xposition = Mouse_Xposition −1
  }
  If(userPressed [right_arrow]}{
      Mouse_Xposition = Mouse_Xposition +1
  }
  If(userPressed [up_arrow]}{
      MouseYposition = MouseYposition +1
  }
  If(userPressed [down_arrow]}{
      MouseYposition = MouseYposition −1
  }
  Cat_Xposition = random (0, 30)
  Cat_Yposition = random (0, 30)

  #Determine whether the cat and mouse landed in the same position
  If(Cat_Xposition = Mouse_Xposition  AND Cat_Yposition = Mouse_Yposition) {
      Say (“Cat Won!”)
      Set Game_Running to FALSE
  }

  #Determine if the mouse made it to the door
  If (Door_Xposition = Mouse_Xposition  AND Door_Yposition = Mouse_Yposition){
      Say (“Mouse Won!”)
      Set Game_Running to FALSE
  }
}
```

Flowchart