

MATLAB Marina: Algorithms 1

Student Learning Objectives

After completing this module, one should:

1. Be able to write algorithms to solve sequential problems.

Terms

algorithm, top down design (stepwise refinement), rubber duck debugging, pair programming

MATLAB Functions, Keywords, and Operators

NA

Algorithms

Algorithms are the logical recipes that underlie computer programs. Virtually all disciplines now rely on some form of algorithmic model: medicine, economic theory, social theory, etc. Computer programs express algorithms in a particular programming language. The goal is to take something that can be done in a human mind and use a computer to perform the mind's work especially for tasks that are tedious or time consuming. Algorithms must be precise and unambiguous.

Consider creating an algorithm to brew coffee. Assumptions: will make cowboy coffee (in a pot on the stove). Figure 1a shows an imprecise sequence of steps to make coffee and Figure 1b shows a more precise sequence of steps to make coffee.

Get a pot
Fill with water
Place on stove
Heat water
Add coffee
Turn off heat
Let sit
Pour into cup

Figure 1a. Algorithm (not precise)

Get a two quart saucepan
Fill with one quart of tap water
Place on stove
Heat water to 210 degrees F
Add 8 Tablespoons coffee (2 Tbsp per cup of water)
Turn off heat
Let sit for 4 minutes
Pour coffee through a strainer into cup

Figure 1b. Algorithm (more precise)

There may be a need for additional algorithms that precisely describe how to get the pot (where it is, how to recognize a saucepan), how the coffee should be ground (is it pre-ground or does it need to be ground, where the coffee grinder is located and how to grind if not pre-ground), how to heat things on a stove, etc.

Algorithms for computer programs need to be precise, more like the sequence of steps of Figure 1b.

Top Down Design (Stepwise Refinement)

Larger more complex problems are typically solved by breaking the problem up into smaller problems, each of which is relatively easily solved or has been solved before. This is called stepwise refinement. The solutions to the smaller problems are integrated into a solution for the larger problem.

A general procedure for solving large complex problems is:

1. Decompose the larger problem into smaller subproblems.
2. Continue decomposing the subproblems until you obtain subproblems that are manageable (easily solvable or already solved).
3. Solve each subproblem and use these solutions to solve the more complex subproblems and finally the original problem.

Keep your initial problems general and add details as necessary. Do not worry about program syntax until you are ready to solve the subproblems. Using programming syntax in your algorithms can lock you into a particular solution which may not be best. Try to keep your algorithms in plain language that a non-technical person would understand.

Algorithm Development and Implementation Example

Develop an algorithm for swapping the contents of two containers. Implement the developed algorithm in MATLAB to swap the contents of two numeric variables.

Switching the contents of two containers is called a swap. Swap operations are used when sorting a collection of data.

Initial Algorithm:

- Create two containers and put the appropriate contents for each container in them.
- Move the contents of container 1 to container 2 and the contents of container 2 to container 1.

The contents of one of the containers cannot be put into the other container until its contents are removed. A temporary place to store one of the containers' contents is needed so that one of the containers is empty.

Refined Algorithm:

- Create two containers and put the appropriate contents for each container in them.
- Create a temporary container.
- Move the contents of container 1 to the temporary container.
- Move the contents of container 2 to container 1.
- Move the contents of the temporary container to container 2.

After the swap operations is complete, the extra temporary container still exists. You might want to delete this. A comment skeleton corresponding to the swap algorithm including

deleting the temporary container is shown in Figure 2a. The corresponding MATLAB code segment is shown in Figure 2b.

```
% create two containers and give them contents
% create temporary container

% transfer the contents of container 1 to temporary container
% transfer the contents of container 2 to container 1
% transfer the contents of temporary container to container 2

% delete the temporary container
```

Figure 2a. Comment Skeleton for Swap Algorithm

```
% create two containers and give them contents
container1 = 10;
container2 = 25;
% create temporary container
tempContainer = 0;

% transfer the contents of container 1 to temporary container
tempContainer = container1;
% transfer the contents of container 2 to container 1
container1 = container2;
% transfer the contents of temporary container to container 2
container2 = tempContainer;

% delete the temporary container
clear tempContainer;
```

Figure 2b. MATLAB Swap Code Segment

Some additional tips:

- Solve the sample problem using a set of data by hand. If you cannot solve the problem, you will not be able to write an algorithm to solve the problem.
- It is often helpful to describe the problem from an input – output perspective, i.e. what information does the program need to solve the problem and what must be calculated.
- Pay attention to the MATLAB warnings and errors; the MATLAB IDE is generally good at catching typos and syntax errors.
- Logic errors (errors in the algorithm) will generally not be caught until the program is tested. Just because a program compiles successfully does not mean it works. Compilers catch syntax errors, not logic errors. Test your solution with some fairly simple data that you can verify by hand and then with a reasonable number of sets of additional data (both valid and invalid). It is here where you will catch any logic errors.
- Having a second set of eyes (pair programming) or using the rubber ducky method is helpful for catching logic errors and debugging code.

Pair Programming

Pair programming is a problem-solving technique that is particularly useful for lengthy or complicated problems. Essentially, pair programming means having two people working together on one computer. One person types, focusing on the code being written, while the other person watches, catching mistakes and logic errors as they happen and anticipating problems. The person typing and the person observing switch places from time to time to keep both participants engaged. Such role division leads to two different perspectives on the code: one that focuses on the details and one that focuses on the overall problem at hand.

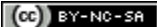
Pair programming is not just useful for catching logic errors while programming; it can also be used when planning a solution or developing an algorithm. Before coding starts, the pair must discuss the problem together and plan their approach. Having two viewpoints will increase the chances of developing a robust solution, as each person brings their own unique ideas and experiences to the table. Also, a fresh perspective on a difficult problem can be invaluable, particularly when one has been stuck for some time.

Rubber Ducky Method of Debugging

Another method of debugging is known as the rubber ducky method. For this method of debugging, one says/reads their code out loud in an explanatory way to a “rubber ducky” or another inanimate object (like a stuffed animal) that can be personified. The code is read line by line and each line is explained to the rubber ducky. The idea being that when you are explaining what each line of code does out loud, your brain takes a break from constantly reading and instead listens. This can make it easier to find bugs or logic errors in the program.

While programming, people are often constantly reading and not talking. This method gives the programmer a break and a chance to debug an issue they could not have found by just reading.

Last modified Friday, September 18, 2020

 [MATLAB Marina](#) is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.