

## Lab 2.2

### Algorithms

Before we proceed to programming computers, we need to prepare ourselves by learning how to think about programming. At the heart of programming is the *algorithm*, basically a **very carefully expressed problem-solving strategy**, usually in the form of a structured set of instructions. We say we “use” or “apply” an algorithm to solve the problem it was designed for. In everyday life, when we think of instructions, we mean instructions for other people to follow, rather than a machine, and we can conveniently make many unstated assumptions and get away with being ambiguous. Unlike instructions for people, instructions for computers can leave no room for misinterpretation, so algorithms require an extreme level of precision that will take some getting used to. Following the guidelines discussed in Chapter 10, in this lab, we will exercise our algorithmic thinking on a wide variety of problems, ranging from everyday tasks to mathematical problems.

### Vocabulary

All key vocabulary used in this lab are listed below, with closely related words listed together:

algorithm, program  
control flow  
conditional, condition  
iteration, looping, loop condition

### Post-Lab Questions

Write your answers after completing the main part of the lab:

1. Identify at least one example of iteration in the algorithms you wrote for this lab. Explain which steps are being repeated and what determines when the repetition stops.
2. Identify at least one example of a conditional in the algorithms you wrote for this lab.

### Discussion and Procedure

You will be writing algorithms for three different kinds of problems in this lab. Before we plunge into writing algorithms for common computing problems, we will work with problems that are probably more familiar to you. In Part 1, you will be writing algorithms for completing everyday tasks like doing laundry or making a sandwich. In Part 2, we will move on to special everyday tasks carefully chosen to have strong

similarities to mathematical or computing problems. For example, locating a word's definition in a dictionary is an example of a "search problem," using computer science terminology. (Chapter 10's CD rack sorting example also falls into this category.) Finally, in Part 3, we will work with problems that are explicitly mathematical or computational in nature (e.g., finding the average of a large set of numbers). You will be instructed to choose one or two problems from each of these three groups and write an algorithm for each. For each algorithm you write, make sure to follow the following instructions:

### *Instructions for Writing Algorithms*

For each problem stated, write a carefully worded and structured set of instructions for solving the problem. Just as in the example algorithm in Chapter 10 (for sorting a CD rack), include the following sections, described in detail:

- inputs or start state, including a list of required materials, if applicable (This might seem obvious for physical tasks like preparing a sandwich. However, even for mathematical problems, you might want to note which quantities you want to keep track of.)
- outputs or end result(s)
- numbered steps

Always keep conscious of the assumptions you are making by writing them down explicitly. You should adhere to the guidelines discussed in Chapter 10 and follow the format shown in the sample solution below. (See the CD rack sorting example in Chapter 10 for a more extended example.)

Remember, there is no one right answer. Many different algorithms might be acceptable for each problem. Due to the flexibility of the English language, the same algorithm can often be expressed in more than one way. In addition, there is almost always more than one way to solve a problem. They might differ in their complexity, time required, and other aspects, which are important differences to consider, but they can all be valid algorithms for the problem.

Finally, both as you write and after you are finished, check your algorithm to make sure it satisfies all of the properties discussed at the beginning of Chapter 10. We have summarized them here in the form of questions you can ask yourself to check your algorithm:

1. **Inputs specified.** Do you precisely describe all of the data (or materials) needed for the algorithm?
2. **Outputs specified.** Do you precisely describe the results of your algorithm? Your outputs description should clearly state what the algorithm is supposed to do and solve the problem the algorithm is designed for.

3. **Definiteness.** As discussed earlier, an algorithm must be expressed very precisely. An ambiguous algorithm, if misinterpreted, might be ineffective at solving the problem it was designed for. It is easy to get carried away, however, and include details that are not essential to the algorithm's correctness. Ask yourself the following as you write: Is this detail essential to the algorithm's correctness? Does it add important information that is not evident from what you have already written? Include the detail if you answer, "yes," to these questions.
4. **Effectiveness.** Are the required resources (whether they be information or the ability to do certain things) realistic and reasonable? For instance, an algorithm for converting a temperature from degrees Fahrenheit to Celsius that relies on looking up the temperature in a Fahrenheit-Celsius conversion table is hardly interesting or useful.
5. **Finiteness.** Are you sure that the algorithm actually finishes? For instance, one algorithm for finding a specific playing card in a deck is to repeat the following: Shuffle the deck and check the top card. Strictly speaking, the card you are looking for might never end up at the top of the deck, no matter how many times you reshuffle. (That would be pretty rotten luck, but it *is* possible.) In contrast, checking the whole deck, one card at a time, while time-consuming, is an algorithm that is guaranteed to finish. At worst, your card would end up being the last one you check.

Before we begin, we will go through a simple example. Notice that each problem statement in this lab is very carefully stated, with as many details and as few implicit assumptions as possible. This is no accident. It is only reasonable that a detailed and precise algorithm requires an equally carefully stated problem.

EXAMPLE PROBLEM: Given a VCR, TV, and videotape of twelve episodes of your favorite television show, each 30 minutes long, find a specified episode and cue the tape to the start of the episode. For instance, suppose you have already seen all but one of the episodes, but you are not sure where the one episode is on the tape. Your objective is to find this episode on the tape. Assume that the TV and VCR are both on, and that the tape is in the VCR, cued to an unknown location (i.e., not necessarily rewind).

EXAMPLE ALGORITHM.

START STATE: videotape of twelve 30 min. episodes, VCR, TV with assumptions as stated in problem above; ability to recognize one specific episode

END RESULT: videotape cued to beginning of specified episode

ASSUMPTIONS:

- VCR has tape counter that shows tape position in hours and minutes.

- Episodes begin at 30 min. multiples, i.e., the first episode is at time 0:00, the second at 0:30, the third at 1:00, etc. There is no other video recorded on the tape between episodes.

**PROCEDURE:**

- 1) Rewind tape to start.
- 2) Reset tape counter.
- 3) Play tape for a few minutes to check whether the episode is the one you are looking for. If so, proceed to Step 5; otherwise, proceed to Step 4.
- 4) Stop play, and fast forward the tape. Watch the counter for the next 30 min. multiple, and press stop when the tape reaches that point, which should be the start of the next episode on the tape. For example, if you stopped the tape at 2:04 (two hours, four minutes), fast forward until the counter reads 2:30. Go to Step 3.
- 5) Stop play, and rewind to the previous 30 min. multiple using the tape counter. For example, if you recognize the episode you are looking for and the counter reads 3:32 (three hours, 32 minutes), rewind to 3:30.

**Part 1. Algorithms for Everyday Problems**

**IMPORTANT ADVICE:** Do not assume that problems in this part will be easy just because the procedures for these tasks might be second nature to you. In fact, the more familiar you are with a task, the more likely it is that you will make implicit assumptions and skip details in your algorithm.

Choose one of the problems below and write an algorithm to solve it. Assume a reasonably equipped kitchen as the context.

- 1a-1. Prepare a bowl of cold cereal. You may choose to include fruit or other ingredients beyond cereal and milk, but make sure to state them explicitly. Assume that a carton of milk is in the refrigerator.
- 1a-2. Prepare a sandwich. Make sure to state what kind of sandwich you are making and what ingredients and utensils are required. Again, assume all required ingredients are in the kitchen, refrigerated if needed.
- 1a-3. Prepare a cup of tea using a tea bag. You may describe adding cream, sugar, lemon, honey, etc., but mention these ingredients in your write-up.

Each of the following everyday problems involves repeating some steps in the procedure (just as Step 4 in the above example can be repeated). Choose one and write an algorithm to solve it:

- 1b-1. Given a pile of pens, find and discard the ones that do not write any more.

- 1b-2. Suppose you are told that your favorite movie is on television right now, but you are not sure which channel it is on. Without the use of a program guide (that is, without using things like TV Guide or the newspaper program listings), find the channel broadcasting the movie. Assume, of course, that you could recognize any part of the movie if you saw it.
- 1b-3. Brush your teeth.
- 1b-4. Wash and dry a load of laundry using coin operated washing machine and dryer.

## Part 2. Algorithms for Computational Everyday Problems

The problems in this section are real-life versions of computational problems. Choose one of the below:

- 2-1. Given two words, determine which would come first in alphabetical order. Do not use a dictionary or other reference book.
- 2-2. Given a bag full of cherries, determine whether you could split them evenly among seven people (so that each person has the same number of cherries and none are left over). The result is a yes or no answer.
- 2-3. Given a restaurant check subtotal (before tax) and total (after tax), calculate how much you should leave for a 15% tip.
- 2-4. Given fifty apples and a two-pan balance (which can determine whether two items have the same weight), determine which two (if any) have the same weight (have weight close enough that the balance measures them as being equal).

## Part 3. Algorithms for Computational Problems

Problems in this section are obviously computational, involving computing mathematically interesting results. They are fundamentally different from the problems in the previous two parts of the lab in that they involve no physical process. That is, these problems involve manipulating numbers rather than physical objects like foods, pens, or laundry detergent. Choose one of these problems:

- 3-1. Given a set of 20 numbers, compute the average of the numbers.
- 3-2. Given a set of 100 numbers, find the largest and smallest numbers in the set.
- 3-3. Find the  $k$ th power of two (without using a calculator).

## Part 4. Recognizing Control Flow Patterns

Some algorithms are very simple, and applying them just involves proceeding from one step to the next (sequential execution). We have already seen examples of algorithms, however, which involve skipping ahead to a later step or repeating an earlier step, depending on certain conditions. *Control flow* is the programming term used to describe the order in which the steps are executed, and we will discuss two common ways control flow diverges from simple, sequential execution. In the Post-Lab Questions, you are

asked to examine the algorithms you wrote and identify at least one example of each of these control flow patterns.

**Conditionals: The keyword is “if.”** A *conditional* (also known as a *branch*) is when control flow reaches a point where it can proceed to one of two (or more) alternatives. The *condition* is what determines which step to proceed to. For example, to round off a number to its nearest integer, you first need to check the following condition: whether the fractional part is less than 0.5. If so, the result is just the integer part; otherwise, you add one to the integer part. Use of the phrase structure “if...then...” and instructions to skip ahead/back to other steps in an algorithm are hallmarks of the conditional.

**Iteration: Controlled repetition.** *Iteration* is basically the repetition of one or more steps in an algorithm. Iteration is related to the conditional in that the number of times you do the repetition is usually controlled by a condition. Iteration is also called *looping* and the condition is accordingly called the *loop condition*. The video searching example algorithm at the start of this lab has a loop in Steps 3-4, where the conditional in Step 3 (checking whether the episode is the one you are looking for) determines whether you repeat Step 4’s fast-forwarding.

Understanding the concepts of control flow and the two standard patterns of the conditional and iteration will help you a great deal as you proceed to expressing algorithms in a programming language.

### Optional Additional Activities

- This activity is *much* more difficult than it sounds, so don’t be quick to dismiss it. It should help you gain an appreciation for precise language and instructions (a topic the textbook returns to in Chapter 7). You will need to work with a friend for this activity. You will be an “instruction writer,” and your friend, a “builder.” Using a building toy kit such as Tinker Toys, Legos, or Construx, build some simple structure using four to six parts. Do not show your friend this structure. Place the structure on a table and, without touching it, write a set of instructions for constructing the structure, starting from the individual parts all separated. You are *not* allowed to draw diagrams or use anything other than English to describe the steps. Next, either find a set of parts identical to those used in the structure or dismantle it after make sure you remember it exactly (with sketches, digital or Polaroid photographs). Provide your friend with the parts and the instructions you wrote and see if they can reproduce the structure from just your instructions. (No coaching from you!)
- Of course, you can always try more than one problem per part in the lab above.