6.5 Expressing Algorithms

In the previous sections, we have written programs to write out a greeting, read characters in and write them out in reverse order, add numbers together and print an error message if the sum is negative, and enter a value and read and sum that many numbers. We expressed the solution to each problem in paragraph form and then wrote the code. In computing, the plan for a solution is called an algorithm. As you saw, going from the problem in paragraph form to the code is not always a clear-cut process. Pseudocode is a language that allows us to express algorithms in a clearer form.

Pseudocode Functionality

Pseudocode is not a real computer language, but rather a shorthand-like language that people use to express actions. There are no special grammar rules for pseudocode, but to express actions we must be able to represent the following concepts.

Variables

Names that appear in pseudocode algorithms refer to places in memory where values are stored. The name should reflect the role of the content in the algorithm. For example, the variable sum could be used to represent the summation of a set of other values.

Assignment

If we have variables, we must have a way to put a value into one. We use the statement

Set sum to 0

to store a value into the variable sum. Another way of expressing the same concept uses a back arrow (<—):

sum <— 1

When we want to access the value of the variable later, we just use the name. For example, we access the values in sum and num in the following statement:

Set sum to sum + num

or

sum <— sum + num

The value stored in sum is added to the value in num and the result is stored back in sum. Thus, when a variable is used on the right side of the to or <—, the value of the variable is accessed. When a variable is used following Set or on the left side of <—, a value is stored into the variable.

The value being stored into a variable can be a single value (as in 0) or an *expression* made up of variables and operators (as in sum + num).

Input/Output

Most computer programs simply process data of some sort, so we must be able to input data values from the outside world and output the result on the screen. We can use the word write for output and the word read for input.

Write "Enter the number of values to read and sum"

Read num

The characters between the double quotation marks are called *strings* and

tell the user what to enter or to describe what is written. It doesn't matter which exact words you use: Display or Print would be equivalent to Write; Get or Input would be synonyms for Read. Remember, pseudocode algorithms are written for a human to translate into some programming language at a later stage, so there are no set rules. However, being consistent within a project is a good idea—both for you as you are working and for the person following you, who may have to interpret what you have written.

The last two output statements demonstrate an important point:

Write "Err"

Write sum

The first writes the characters between the double quotation marks on the screen. The second writes the *contents* of the variable sum on the screen. The value in sum is not changed.

Selection

The selection construct allows a choice between performing an action or skipping it. Selection also allows a choice between two actions. The condition in parentheses determines which course of action to follow. For example, the following pseudocode segment prints the sum or an error message.

- // Read and sum three numbers
- IF (sum < 0)
    - Print error message

- ELSE
    - Print sum
- // Whatever comes next

We use indention to group statements (only one in this case). Control goes back to the statement that is not indented. The // introduces a comment to the reader, which is not part of the algorithm.

This version of the selection construct is called the *if-then-else* version because the algorithm chooses between two actions. The *if-then* version is the case where an action is executed or skipped. If we wanted to print the sum in any event, we could show the algorithm this way.

- // Read and sum three numbers
- IF(sum < 0)
    - Print error message
- Print sum
- // Whatever comes next

Repetition

The repetition construct allows instructions to be repeated. In the summing problem, for example, a counter is initialized, tested, and incremented. Pseudocode allows us to outline the algorithm so that the pattern becomes clear. Like the selection construct, the expression in parentheses beside the WHILE is a test. If the test is true, the indented code is executed. If the test is false, execution skips to the next non-indented statement.

- Set limit to number of values to sum
- WHILE (counter < limit)
    - Read num
    - Set sum to sum + num
    - Set counter to counter + 1
- // Rest of program

The expression in parentheses beside the WHILE and the IF is a Boolean expression, which evaluates to either true or false. In the IF, if the expression is true, the indented block is executed. If the expression is false, the indented block is skipped and the block below the ELSE is executed if it exists. In the WHILE, if the expression is true, the indented code is executed. If the expression is false, execution skips to the next non-indented statement. We are putting WHILE, IF, and ELSE in all capital letters because they are often used directly in various programming languages. They have special meanings in computing.

TABLE 6.1 summarizes the pseudocode elements we've covered

TABLE

6.1 Pseudocode Elements

Construct	What It Means	Example
Variables	Named places into which values are stored and from which values are retrieved.	sum, total, counter, name
Assignment	Storing a value into a variable.	Set number to 1 number <— 1
Input/output	Input: reading in a value, probably from the keyboard. Output: displaying the contents of a variable or a string, probably on the screen.	Read number Get number Write number Display number

		Write "Have a good day"
Repetition (iteration, looping)	Repeat one or more statements as long as a condition is true.	While (condition) // Execute indented statement(s)

Selection: *if-then*	If a condition is true, execute the indented statements; if a condition is not true, skip the indented statements.	IF (newBase = 10)
		Write "You are converting "
		Write "to the same base."
		// Rest of code

Selection: *if-then-else*	If a condition is true, execute the indented statements; if a condition is not true, execute the indented statements below ELSE.	IF (newBase = 10)
		Write "You are converting "
		Write " to the same base."
		ELSE
		Write "This base is not the "
		Write "same."
		// Rest of code

Here is the pseudocode algorithm for the program that read and summed two values and printed an error message if the total was negative:

- Set sum to 0
- Read num1
- Set sum to sum + num1
- Read num2
- Set sum to sum + num2
- If (sum < 0)

    - Write “Error”
- ELSE
    - Write sum

Here is the pseudocode algorithm for the program that input the number of values to read, read them, and printed the sum:

- Set counter to 0
- Set sum to 0
- Read limit
- While (counter < limit)
    - Read num
    - Set sum to sum + num
    - Set counter to counter + 1
- Print sum

Note that expressing an idea in pseudocode might translate to many instructions in low-level languages like assembly language.