**Lab 5: Repetition Structures**

This lab accompanies Chapter 5 of *Starting Out with Programming Logic & Design*.

Name: Ulric Aird

# Lab 5.1 –Repetition Structures Pseudocode: Condition Controlled Loops

|  |
| --- |
| Critical Review    A repetition structure causes a statement or set of statements to execute repeatedly.    Repetition structures are used to perform the same task over and over.    Repetition structures are commonly called loops    A condition-controlled loop uses a true/false condition to control the number of times that it repeats.    The general structure of a While loop with a condition-controlled statement is:    //Declare loop control variable  While condition  Statement  Statement  Etc.  //Ask Question that changes the loop control variable  End While    The general structure of a Do While loop with a condition-controlled statement is:    //Declare loop control variable  Do  Statement  Statement  Etc.  //Ask Question that changes the loop control variable  While Condition |

This lab requires you to implement a condition controlled loop.

**Step 1:** Examine the following main module from Lab 4.2. Loops are commonly used to call modules multiple times. The best design is to use a loop around the module calls in Main.

Module main ()

//Declare local variables

Declare Real monthlySales

Declare Real storeAmount

Declare Real empAmount

Declare Real salesIncrease

//Function calls

Call getSales(monthlySales)

Call getIncrease(salesIncrease)

Call storeBonus(monthlySales, storeAmount)

Call empBonus(salesIncrease, empAmount)

Call printBonus(storeAmount, empAmount)

End Module

**Step 2:** In the space provided, create a loop control variable named keepGoing of the data type String. Initialize this variable to "y". (Reference: Modularizing the Code in the Body of a Loop, page 226).

**Step 3:** In the space provided, write a While statement.

Module main ()

// Declare local variables

Declare Real monthlySales

Declare Real storeAmount

Declare Real empAmount

Declare Real salesIncrease

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// Function calls

While \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Call getSales(monthlySales)

Call getIncrease(salesIncrease)

Call storeBonus(monthlySales, storeAmount)

Call empBonus(salesIncrease, empAmount)

Call printBonus(storeAmount, empAmount)

Display "Do you want to run the program again? (Enter y for yes)."

Input \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

End While

End Module

**Step 4:** In the space provided, create a loop control variable named keepGoing of the data type string. Initialize this variable to "y". (Reference: Writing a Do-While Loop in Pseudocode, page 229).

**Step 5:** In the space provided, write a Do-While statement.

Module main ()

// Declare local variables

Declare Real monthlySales

Declare Real storeAmount

Declare Real empAmount

Declare Real salesIncrease

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// Function calls

Do

Call getSales(monthlySales)

Call getIncrease(salesIncrease)

Call storeBonus(monthlySales, storeAmount)

Call empBonus(salesIncrease, empAmount)

Call printBonus(storeAmount, empAmount)

Display "Do you want to run the program again? (Enter y for yes)."

Input \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

While \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

End Module

# Lab 5.2 –Repetition Structures Pseudocode: Counter Controlled Loops

|  |
| --- |
| Critical Review    A count-controlled loop repeats a specific number of times.    The loop keeps a count of the number of times that it iterates, and when the count reaches a specified amount the loop stops.    A variable, known as a counter variable, is used to store the number of iterations that it has performed.    The three actions that take place are initialization, test, and increment.   * Initialization: Before the loop begins, the counter variable is initialized to a starting value. * Test: The loop tests the counter variable by comparing it to a maximum value. * Increment: To increment a variable means to increase its value. This is done by adding one to the loop control variable.     Any loop can be used with a count-controlled loop.    A running total is a sum of numbers that accumulates with each iteration of a loop. The variable used to keep the running total is called an accumulator. |

This lab requires you to write a complete program using a condition controlled loop, a counter controlled loop, and an accumulator. The program is as follows:

Write a program that will allow a grocery store to keep track of the total number of bottles collected for seven days. The program should allow the user to enter the total number of bottles returned for seven days. The program will calculate the total number of bottles returned for the week and the amount paid out (the total returned times .10 cents). The output of the program should include the total number of bottles returned and the total paid out.

**Step 1:** In the pseudocode below, declare the following variables under the documentation for Step 1.

* A variable called totalBottles that is initialized to 0 o This variable will store the accumulated bottle values
* A variable called counter and that is initialized to 1 o This variable will control the loop
* A variable called todayBottles that is initialized to 0 o This variable will store the number of bottles returned on a day
* A variable called totalPayout that is initialized to 0 o This variable will store the calculated value of totalBottles times .10
* A variable called keepGoing that is initialized to "y"

o This variable will be used to run the program again

**Step 2:** In the pseudocode below, make calls to the following functions under the documentation for Step 2.

* A function call to getBottles that passes totalBottles, todayBottles, and counter.
* A function called calcPayout that passes totalPayout and totalBottles.
* A function called printInfo that passes totalBottles and totalPayout

**Step 3:** In the pseudocode below, write a condition controlled while loop around your function calls using the keepGoing variable under the documentation for Step 3.

**Complete Steps 1-3 below:**

Module main ()

**// Step 1: Declare variables below**

Declare Real totalBottles

Declare Real totalPayout

Declare Real counter

Declare Real todayBotttles

Delare String keepGoing

**// Step 3: Loop to run program again**

While keepGoing = “yes”

**// Step 2: Call functions**

Call getBottles(totalBottles, todayBottles, counter)

Call calcPayout(totalPayout, totalBottles)

Call printInfo(totalBottles, totalPayout)

Display "Do you want to run the program again? (Enter y for yes)."

Input keeyGoing

End While

End Module

**Step 4:** In the pseudocode below, write the missing lines, including:

1. The missing parameter list
2. The missing condition (Hint: should run seven iterations)
3. The missing input variable
4. The missing accumulator
5. The increment statement for the counter

// getBottles module

Module getBottles(int totalBottles, int todayBottles, int counter)

counter = 1

While counter >= 7

Display "Enter number of bottles returned for the day:"

Input todayBottles

totalBottles += todayBottles

counter += 1

End While

End Module

**Step 5:** In the pseudocode below, write the missing lines, including:

1. The missing parameter list
2. The missing calculation

// calcPayout module

Module calcPayout(int totalPayout, int totalBottles)

totalPayout = 0 // resets to 0 for multiple runs

totalPayout = totalBottles \* .10

End Module

**Step 6:** In the pseudocode below, write the missing lines, including:

1. The missing parameter list
2. The missing display statement
3. The missing display statement

//printInfo module

Module printInfo(int totalBottles, int totalPayout)

Display “Total Bottles: “, totalBotttles

Display "Total Payout: “, totalPayyout

End Module

# Lab 5.3 – Flowcharts

Critical Review

In a While loop, the question is asked first. After the statements process, the

control goes back above the condition.

C

o

n

d

i

t

i

o

n

S

t

a

t

e

m

e

n

t

s

False

T

r

u

e

In a Do

-

While loop, the question is asked last. The statements always process at

least one time.

Condition

Statement

s

Fal

s

e

True

In Raptor, you can place the modules before or after the condition depending on

whether you want to use a Do

-

While or a while loop.

This lab requires you to convert your pseudocode in Lab 5.2 to a flowchart. Use an application such as Raptor or Visio.

**Step 1:** Start Raptor and save your document as *Lab 5-3*. The *.rap* file extension will be added automatically. Start by adding a Comment box that declares your variables. The only variable from Lab 5.2 that is different is the keepGoing variable. Name this endProgram instead.

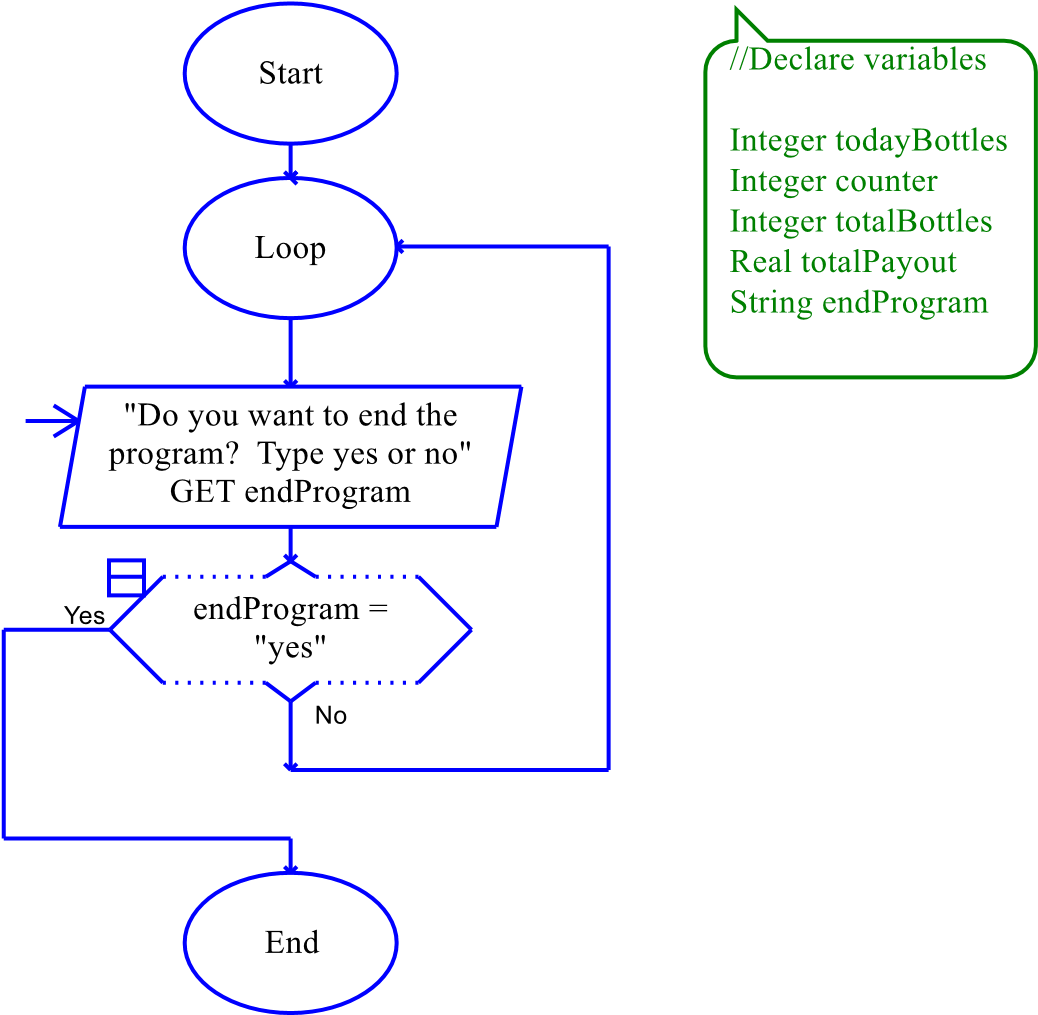
**Step 2:** Click the Loop symbol and drag and drop it between the Start and the End symbol.

**Step 3:** Click the Input symbol and drag and drop it between the Loop symbol and the Diamond symbol.

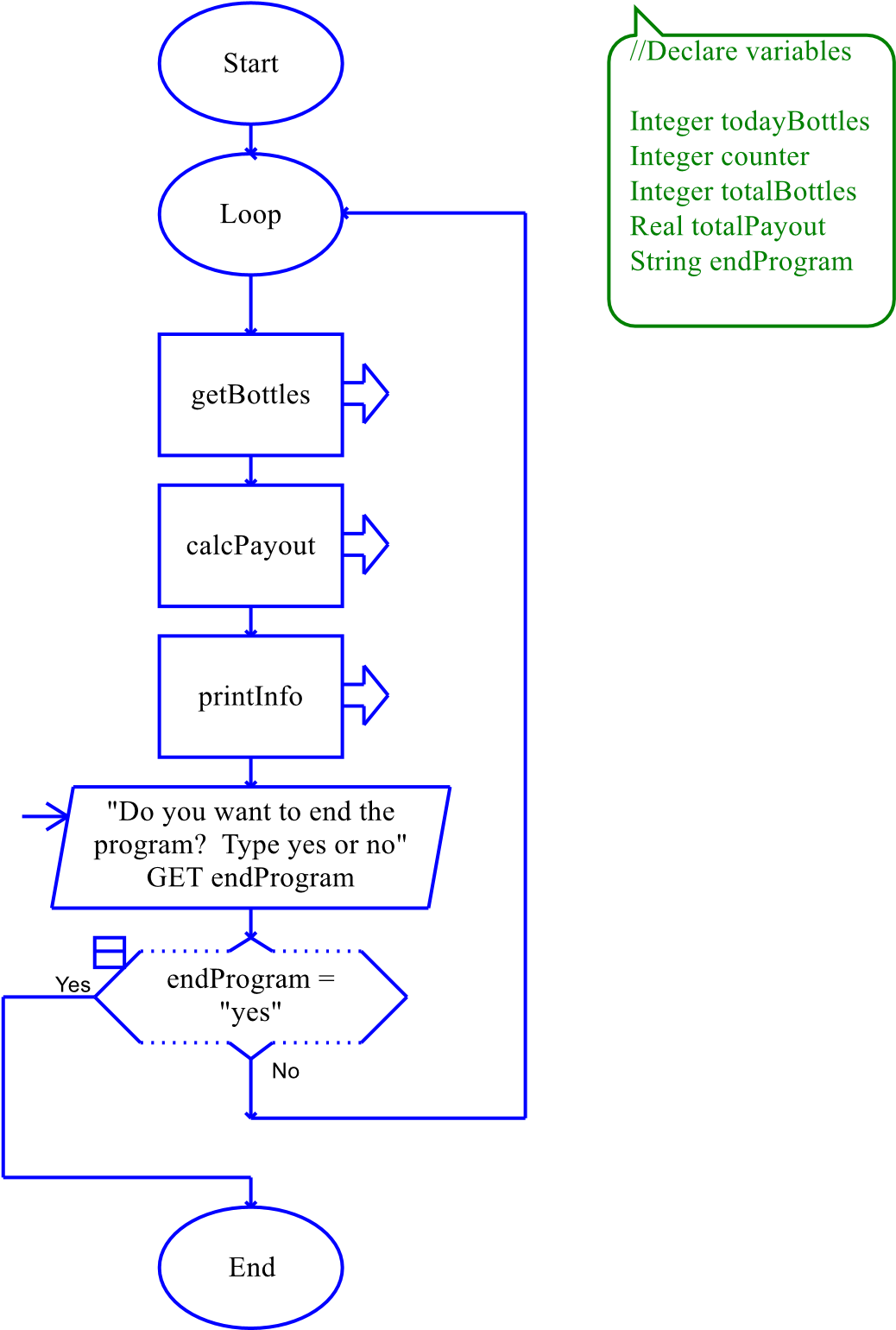
**Step 4:** Double click the Input symbol and ask the question "Do you want to end the program? Enter yes or no:". Store the answer in the endProgram variable.

**Step 5:** Double click the Diamond symbol, and type endProgram = "yes" as the condition.

When the program executes, the user must type "yes" exactly, in order for the program to end. Now, main should look as the following:



**Step 6:** The next step in your flowchart should be to call your methods. Add your modules under the Loop oval. Be sure to click yes to add new tabs for each module. Now, main should look as the following:



**Step 7:** Click on the getBottles tab. Add a Loop symbol between the Start and End symbols. Double click the Diamond symbol and enter the condition as counter >7.

**Step 8:** Add an Input symbol and add the code "Enter the number of bottles returned for today:". Store the value in the todayBottles variable.

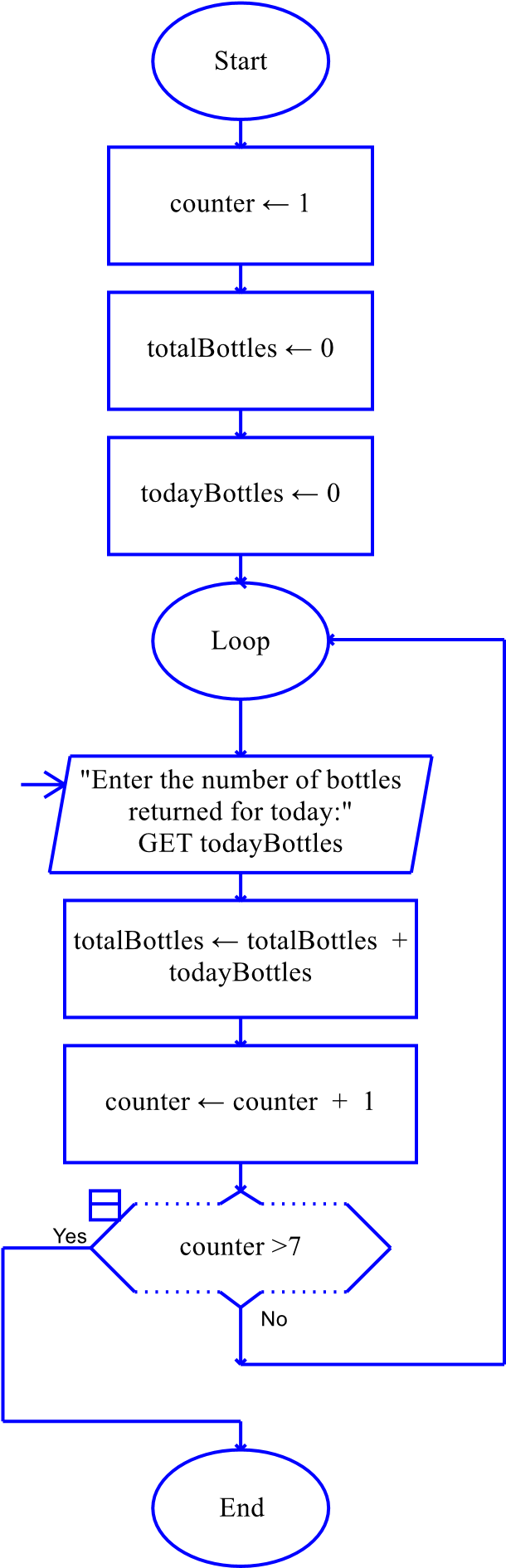
**Step 9:** Add an Assignment symbol next and set totalBottles to totalBottles + todayBottles.

**Step 10:** Add another Assignment symbol next and set counter to counter + 1.

**Step 11:** Save your program and try running it. You'll notice an error occur when the loop starts processing in the getBottles module. This is because totalBottles does not have a starting value.



**Step 12:** To fix the error, to set the counter to 1, and to reset the todayBottles back to 0 for multiple repetitions, add three Assignment symbols above the Loop symbol. In one symbol, set counter to 1. In the other, set totalBottles to 0. In the other, set todayBottles to 0. Your getBottles module should look as follows:



**Step 13:** Click the calcPayout module and add an Assignment symbol. Set totalPayout to totalBottles times .10.

**Step 14:** Click the printInfo module and add two Output symbols that print the total bottles returned and the total amount paid out.

**Step 15:** Test your program against the following values. If there is an error, go back through the steps to locate the problem.

|  |  |
| --- | --- |
| **Input Values** | **Expected Output** |
| Seven days of bottles:  346  238  638  890  1035  899  536 | The total number of bottles collected were: 4582 The total amount paid out is $458.2000 |

**Step 16:** The final step is to insert your finished flowchart in the space below.

**PASTE FLOWCHART HERE**

