5.5 Understanding Control Flow

This Chapter has introduced new statements that can be used to control the sequence of actions the computer performs. These statements allow you to add Branching and Looping paths to your code. The flowcharts presented in Section 5.2 are a great way of visualising the order in which the computer will execute the instructions. To help you fully understand these concepts this section will look at how these statements work within the

5.5.1 Understanding Branching in Perform Guess

Figure 5.53 shows the flowchart for the `Perform Guess` that was develop
Designing Control Flow for Perform Guess. The following sections sh
executes these actions. These illustrations will start at the call into Pe
the illustration of the steps that lead up to this call.

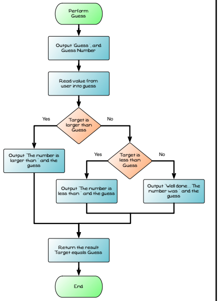

Perform Guess
Output 'Guess ', and Guess Number
Read value from user into guess
Yes
Target is larger than Guess
No
Output 'The number is larger than ' and the guess
Yes
Target is less than Guess
No
Output 'The number is less than ' and the guess
Output 'Well done... The number was ' and the guess
Return the result Target equals Guess
End

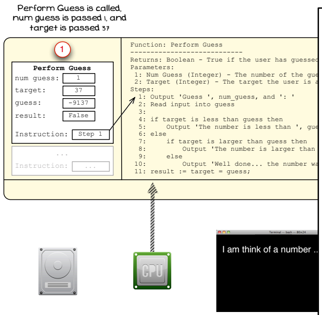
Figure 5.53: Logic for the `Perform Guess` Procedure from Figu

In the following illustrations `Perform Guess` will be called three times wi
being 37 in each case. The following three guesses will be performed, e
through the flowchart are covered.

1. On the first guess the user enters a guess of 50, allowing for the le
flowchart to be followed.
2. The second guess will be 25 to test the middle branch, taking the e
decision and the *true* branch of the second decision.
3. Finally the third guess will be 37, testing the right most path thro

Perform Guess is called for guess 1

In the Guess that Number program, the `Perform Guess` function is responsible for reading in the user's guess and giving them feedback. Figure 5.54 shows the `Perform Guess` code being called for the first time, it is passed 1 to its `num guess` parameter and 37 to its `target` parameter.



```
Function: Perform Guess
-----------------------------
Returns: Boolean - True if the user has guessed
Parameters:
 1: Num Guess (Integer) - The number of the gue
 2: Target (Integer) - The target the user is a
Steps:
  1: Output 'Guess ', num_guess, and ': '
  2: Read input into guess
  3:
  4: if target is less than guess then
  5:     Output 'The number is less than ', gue
  6: else
  7:     if target is larger than guess then
  8:         Output 'The number is larger than
  9:     else
 10:         Output 'Well done... the number wa
 11: result := target = guess;
```

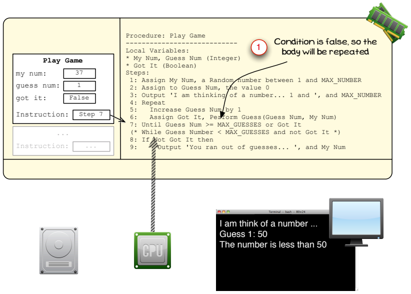
```
I am think of a number ..
```

Figure 5.54: Perform Guess is called for the first time

Note
- In Figure 5.54 the indicated areas show the following:
  1. `Perform Guess` is called, with 1 being passed to `num guess`
  `target`.
- At this point the previous code would have output 'I am thinking
  the Terminal.
- The values in `guess` and `result` have not been initialised, so they h
  was in that memory location previously.

Loop condition is checked at the end of guess 1, with the loop being repeated

At the end of the loop the condition is checked, in this case the loop will run again.



```
Procedure: Play Game
-----------------------------
Local Variables:
* My Num, Guess Num (Integer)
* Got It (Boolean)
Steps:
 1: Assign My Num, a Random number between 1 and MAX_NUMBER
 2: Assign to Guess Num, the value 0
 3: Output 'I am thinking of a number... 1 and ', and MAX_NUMBER
 4: Repeat
 5:   Increase Guess Num by 1
 6:   Assign Got It, Perform Guess(Guess Num, My Num)
 7: Until Guess Num >= MAX_GUESSES or Got It
 (* While Guess Number < MAX_GUESSES and not Got It *)
 8: If Not Got It then
 9:   Output 'You ran out of guesses... ', and My Num
```

```
I am think of a number ...
Guess 1: 50
The number is less than 50
```

Figure 5.70: Condition indicates that the loop's body should be executed again

Note
- In Figure 5.70 the indicated areas show the following:
  1. The condition is checked, and the expression is `false`.
- With **repeat...until** you can evaluate the expression by:
  1. `Guess Num >= MAX_GUESSES` is `1 >= 7`, this is **false**
  2. `Got it`, this is a variable, its value is **false**
  3. Or the above together, `false or false`, this is **false**, repeating the loop.
- With **do...while** you can evaluate the expression by:
  1. `Guess Num < MAX_GUESSES` is `1 < 7`, this is **true**
  2. `Got it`, this is a variable, its value is *false*, so `!Got it`, is not `false`, is **true**
  3. And together these results, `true and true` is **true**, repeating the loop.

C
For C you will need to code this as a C Do While Loop. The code for this will be `do...while(guess_num < MAX_GUESSES && !got_it);`

Pascal
For Pascal you will need to code this as a Pascal Repeat Until Loop. The code for this will be `repeat...until (guess_num >= MAX_GUESSES) or (got_it);`