Combining blocks for the Perform Guess

With the basic theory at hand, we can now start to design the control flow for the Guess that Number program. This process will involve, once again, the idea of **abstraction**. When designing the flow for a program you first need to be able to perform the process yourself, even if its just on paper, and then work out the steps that you undertook so that you can code these within the program.

For the Guess that Number program we can start by designing the control flow within the `Perform Guess` function. The specification of this is shown in Table 5.6. Think about the steps that need to be performed to achieve this. If you had been asked to do need to do?

Function	
`Perform Guess`	
Returns	
`Boolean`	True when the user has guessed the numb otherwise.
Parameter	**Description**
`Guess Number`	The number of the current guess, used in the asking for the user to enter their guess.
`Target`	The number the user is aiming to guess.
Perform Guess is responsible for coordinating the actions perform a single guess within a game of *Guess that Number*. `guess` is read, and the value checked against the `target` valu sage is then output telling the user if the target value is larger than, or equal to their guess. This function returns the user's guess is equal to the target.	

Table 5.6: Specification for the `Perform Guess` Function

The first task the Function needs to perform is to get the guess from t performed in a **sequence**: display a prompt, read the value from the use is shown in Figure 5.26.

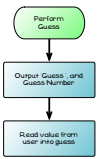

Figure 5.26: Initial Sequence in `Perform Guess`

The next step in this sequence is to give the user feedback based upon their guess and the target number. This code requires a the ability to *select* a given branch. The computer needs to output different messages based upon the users guess. This can be achieved with a **selection** block. Looking back at Figure 5.24 there are three possible alternatives for implementing this selection. The `if` with no `else` is not a valid option as there are three paths we need to take. The case block is also not valid as we are not matching a value, but comparing values to each other. The last option is the `if-else` block, but this only has two branches. It is not going to be possible to code all three options within one block, but it can be achieved using two `if-else` blocks.

The first `if-else` block will check if the `target` is greater than the user's `guess`. If this is true then the computer can take the first branch and output the message 'The number is larger than ' and the value from the user's guess. The flow chart for this part is shown in Figure 5.27. This block is the third task in the sequence, this if block has a single entry, causes a branch in the flow, and will have a single exit.

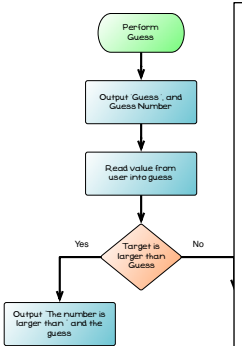

Figure 5.27: First branch in `Perform Guess`

Note

- The conditions within the If Statement are Boolean Expressions
- This condition is checking if `target > guess`.
- There are now two paths through this code, one when `target` is > when it is not.

The Pseudocode for `Perform Guess`

Listing 5.1 contains the Pseudocode for the `Perform Guess` logic from the flowchart in Figure 5.30. Notice how the indentation in this mirrors the block structures in the flowchart. It is good practice to indent your code in this way as it helps you, and any person who reads your code, to see the structure of the logic. You will be able to avoid many errors by making sure that you always indent your code so that it highlights the code's structure.

Pseudocode

```
Function: Perform Guess
-----------------------
Returns: Boolean - True if the user has guessed the Target
Parameters:
 1: Num Guess (Integer) - The number of the guess (1..7)
 2: Target (Integer) - The target the user is aiming for
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 ', guess
  6: else
  7:     if target is larger than guess then
  8:         Output 'The number is larger than ', guess
  9:     else
 10:         Output 'Well done... the number was ', guess
 11: Return the result, target = guess
```

Listing 5.1: Pseudocode for `Perform Guess`

Note

- Code indentation makes it easier to read, and helps locate many common issues.
- Tab you code in within a **structured statement**.
  - Indent the code in the branches of an If Statement and Case Statement.
  - Indent the code within the body of the **While Loop** and the **Do While** or **Repeat Until** loops.
- Make this a habit. When you code a Branching or Looping statement automatically indent the next line of code.
- Always keep you code neat, make it look good.
- The C code for `Perform Guess` is shown in Listing 5.2.
- The Pascal code for `Perform Guess` is shown in Listing 5.3.
- Notice how the two code samples are laid out in a similar way. The indentation makes it easy to identify which statements are associated with each of the branches through the Function.