Pseudocode

What is pseudocode? A simplified, half-English, half-code **outline** of a computer program. It is sometimes given other names, such as *Program Definition Language (PDL).*

Why use it? Because it can help you to clarify your thoughts, and design a routine properly, **before** you start to write any code.

One of the hardest things to resist is the temptation to start writing code. Compared to typing source code, designing the functions which will make up a program seems dull. But spending ten minutes to think out carefully the pros and cons of different approaches to the goal can save you hours of time later on.

Another reason to use pseudocode: it turns into comments in your finished program, so you're save much of the task of going back to a finished program and inserting comments.

Pseudocode in action

Suppose you are given an assignment:
Write a program which plays a guessing game with the user. The user guesses a number between 1 and 100, and the program has to figure it out.

The most efficient way to guess the unknown number is to use a binary search. Let's try to design a program to do that.

Here's the high-level outline:

```
Greet the user

Set initial values

Loop
  guess the middle of the current range
```

```
    get feedback from user
    incorporate the feedback

  Victory!
```

This leaves out a lot of the detail, but shows the basic form of the program.

We can help ourselves by formatting this material as valid Scilab comments:

```
// Greet the user

// Set initial values

// Loop
  //  guess the middle of the current range
  //  get feedback from user
  //  incorporate the feedback

// Victory!
```

Now, we move down a level to fill in some of the details:

```
//  Greet the user

//  Set initial values
//  bottom and top of range of values we will guess
//  number of guesses it has taken so far

//  LOOP
//  guess the middle of the current range

//  get feedback from user
//    'y' for 'yes, that is correct'
//    'h' for 'guess is too high'
//    'l' for 'guess is too low'
```

```
//   incorporate the feedback
//     if we are correct, break out of the loop
//     if too high, re-set range to lower half of current
//     if too low,  re-set range to upper half of current

//   End of LOOP over guesses

//   Victory!
//   print out the number of guesses it took
```

Looks good.

In this simple case, we're now ready to start writing code, using the pseudocode as a guide:

```
//   Greet the user
mprintf('Hello!  Please play a guessing game with me.\n');

//   Set initial values
//   bottom and top of range of values we will guess
bottom = 1;
top = 100;
mprintf('Please guess a number between %d and %d \n', bottom, top);

//   number of guesses it has taken so far
guesses_so_far = 0;

//   LOOP
done_yet = 0;
while (done_yet == 0)

  // guess the middle of the current range
  guess = round((top + bottom)/2);
  mprintf('My guess is %3d.\n', guess);
  guesses_so_far = guesses_so_far + 1;
```

```
    // get feedback from user
    //    'y' for 'yes, that is correct'
    //    'h' for 'guess is too high'
    //    'l' for 'guess is too low'
    user_says = input('Type y if correct, h if too high, l if too low ', 's');

    // incorporate the feedback
    if (user_says == 'y')
      // if we are correct, break out of the loop
      done_yet = 1;

    elseif (user_says == 'h')
      // if too high, re-set range to lower half of current
      top = guess;

    elseif (user_says == 'l')
      // if too low,  re-set range to upper half of current
      bottom = guess;

    else
      // whoops! the user typed an invalid response
      mprintf('sorry, I did not understand that\n');
    end

  // End of LOOP over guesses
  end

  // Victory!
  // print out the number of guesses it took
  mprintf('it took me %d guesses to pick your number %d\n', ...
          guesses_so_far, guess);

endfunction
```

There is just one section of the code above that didn't appear in my pseudocode: the part labelled

```
  // whoops! the user typed an invalid response.
```

I realized I would have to handle this case as I was typing in the if/elseif statements.

Now, all that we need to do to turn this into a functioning Scilab program is add a function header:

```
function guess
//
// Play a guessing game with the user.  Try to guess a number,
//   using a binary search to narrow down the value.
//
// Arguments:
//   none
//
```

Source: http://spiff.rit.edu/classes/phys317/lectures/pseudocode.html