Designing a Program and Subroutines

Summary by: Emmanuel J Rodriguez

Note: Subroutines are commonly called, depending on the programming language, modules, subprograms, methods, and functions.

Top-down design (sometimes called stepwise refinement) is used to break down an algorithm into subroutines.

Top-Down Design Process:

- The overall task of the program is broken down into a series of subtasks.
- Each of the subtasks is examined to determine whether it can be further broken down into more subtasks. This step is repeated until no more subtasks can be identified.
- Once all of the subtasks have been identified, they are written in code.

Three main tools for designing a program and its subroutines:

1. **Hierarchy Chart** – or a structure chart, a top-level visual representation of the main program and the relationships between subroutines.
2. **Flowcharts** – a diagram that graphically depicts the steps that take place in a program.
3. **Pseudocode** – or “fake code” is an informal language that has no syntax rules, it is a “mock-up” program. Each statement in the pseudocode represents an operation that can be performed in any high-level language.

Source: Gaddis, T., *Programming Logic and Design* (New Jersey: Pearson, 2013) pp. 79 – 112.

Top-Down Design
Program: Average and standard deviation

Example from: Gilat. A., *MATLAB: An Introduction with Applications*. (Hoboken, NJ: Wiley, 2017) 242.

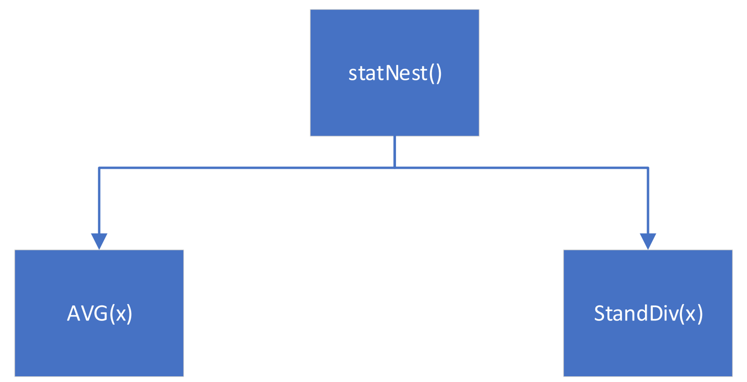
Overall Task:
Calculate the average and standard deviation of a given data set.

Steps that must be taken to perform the task:
1. Calculate average
2. Calculate standard deviation

1. Hierarchy Chart

Main Program
(Input data, Output average)

Subroutines
(Calc avg, calc std dev)

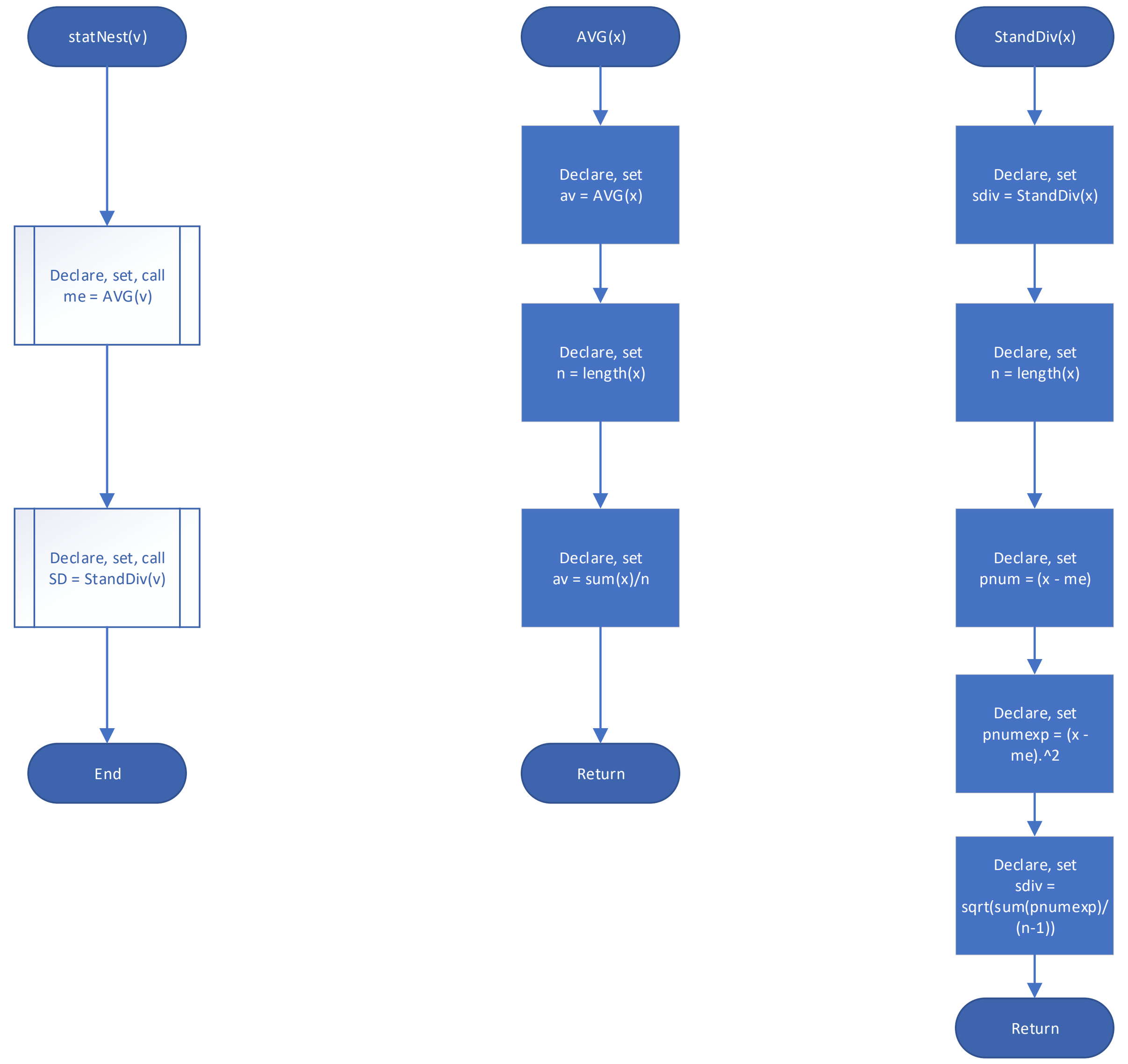


Example from: Gilat. A., *MATLAB: An Introduction with Applications*. (Hoboken, NJ: Wiley, 2017) 242.

Note: Hierarchy charts does not show the steps that are taken inside a subroutine; they do not reveal any details about how subroutines work.

2. Flowchart

Global variable:
grades = v



3. Pseudocode

```
% Global variable for grades data
Grades = v

%% Main program statNest accepts input argument v (grades), outputs arguments me (my average) and SD (standard deviation)
Program [me, SD] = statNest(v)
        % Declare a variable to store the output argument me (average), set it to the AVG subroutine with the grades input argument, and call
        % it to pass the data
        Declare, Set, Call Scalar me = AVG(v)

        % Declare a variable to store the output argument SD (std dev), set it to the StandDiv subroutine with the grades input argument, and
        % call it to pass the data
        Declare, Set, Call Scalar SD = StandDiv(v)
End program

% The AVG subroutine calculates the average by accepting the argument v (grades) and stores it in the reference variable x; once all
% statement lines execute, the result is returned to the main program – to where it left off executing, known as its return point.
% It is important to understand the mechanics on how the above comment line is executed – the AVG subroutine will 'carry' the v data and
% set it equal to the reference variable x.
% A reference variable allows the AVG subroutine to modify the argument in the calling part of the statNest program.
% By using a reference a variable, two things are possible:
% 1. The calling program statNest can communicate with the called subroutine by passing an argument v,
% 2. The called AVG subroutine can communicate with the calling program by modifying the value of the argument v via the reference
% variable x.

% Set subroutine to average variable av
Subroutine av = AVG(x)
        % Declare and set a variable to store the sample size
        Declare Scalar n = length(x)

        % Declare and set a variable to store the average
        Declare Scalar av = sum(x) / n
End subroutine

% The StandDiv subroutine calculates the standard deviation by accepting the argument v and stores it in the reference variable x.

% Set subroutine to standard deviation variable sdiv
Subroutine sdiv = StandDiv(x)
        % Declare and set a variable to store the sample size
        Declare Scalar n = length(x)

        % Declare and set a variable to store the parentheses output of the std dev expression
        Declare Array pnum = (x – me) % output is an array

        % Declare and set a variable to store the pnum variable exponentiated
        Declare Array pnumexp = (pnum).^2 % output is an array

        % Declare and set a variable to store the std dev
        Declare Scalar sdiv = sqrt(sum(pnumexp)/(n-1)) % sum function adds all the array elements
End subroutine
```