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.

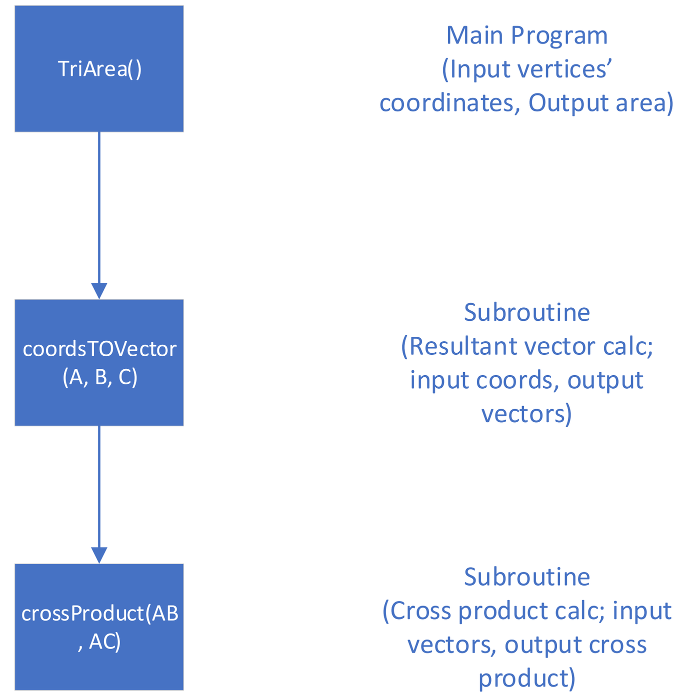
Top-Down Design
Program: Area of a Triangle ABC

Overall Task:
Calculate the area of a triangle given the vertices' coordinates and using the formula given.

$$A = \frac{1}{2}|\boldsymbol{AB} \times \boldsymbol{AC}|$$

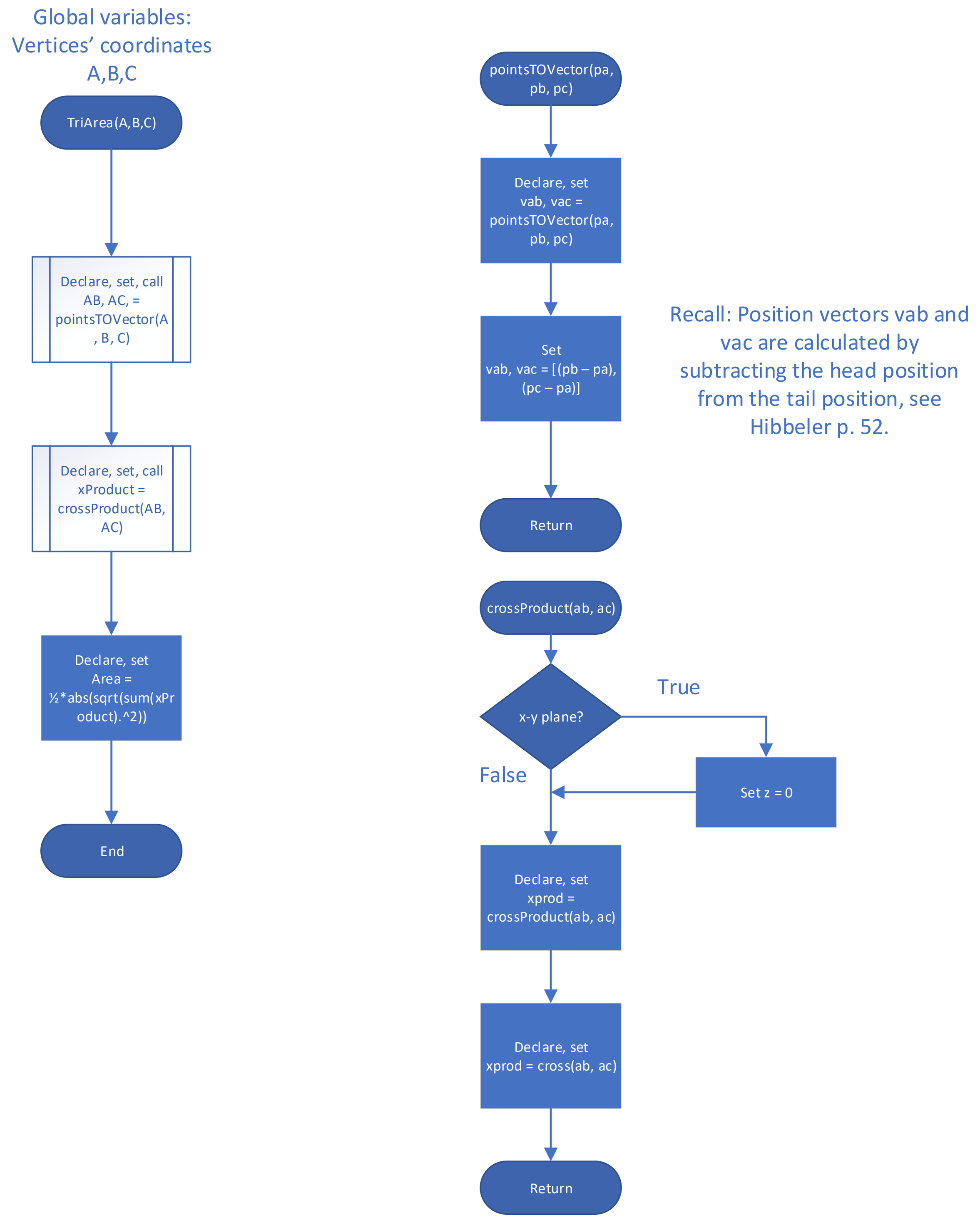
Steps that must be taken to perform the task:
1. Determine vectors AB and AC.
2. Calculate cross product of vectors AB and AC. Calculation should be valid for a triangle in the x-y plane and for a triangle in space (x-y-z).
3. Determine the area of a triangle.

1. Hierarchy Chart



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



3. Pseudocode

```
% Global variables
points = A, B, C

%% Main program TriArea accepts input arguments A, B, C (points or vertices' coordinates), outputs argument Area
Program [Area] = TriArea(A, B, C)
          % Declare variables to store the output arguments AB, AC, set it to pointTOVector subroutine with points argument, and call it to pass
          % the data
          Declare Array AB, AC = pointsTOVector(A, B, C)

          % Declare variable to store the output argument xProduct, set it to crossProduct subroutine with vector arguments, and call it to pass
          % the data
          Declare Array xProduct = crossProduct(AB, AC)

% The pointsTOVector subroutine calculates the position vector by accepting the points argument and stores it in the parameter reference
% variables pa, pb, pc; 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 pointsTOVector subroutine will pass the
% points data to reference variables pa, pb, and pc.
% A reference variable allows the subroutine to modify the argument in the calling part of the main program.
% By using a reference a variable, two things are possible:
% 1. The calling program can communicate with the called subroutine by passing an argument,
% 2. The called subroutine can communicate with the calling program by modifying the value of the argument via the reference variable.

% Declare and set subroutine
Subroutine Array vab, vac = pointsTOVector(pa, pb, pc)
          % Set position vectors
          vab = pb – pa
          vac = pc – pa
End subroutine

% The crossProduct subroutine performs the cross product of two vectors by accepting the vectors argument.
% Declare and set subroutine
Subroutine xprod = crossProduct(ab, ac)
          % Check to if vectors are on the x-y plane, if so, then make the z coordinate equal to zero
          If AB array dimension length == 2 or AC array dimension length == 2 Then % The == is a MATLAB relational operator 'Equal to'
                    ab(3) = 0 % Indexing element #3 of array ab, then setting it to zero
                    ac(3) = 0
          % Set xprod variable to store the cross product array
          Set Array xprod = cross(ab, ac) %Note: MATLAB's cross function must have an input argument of 3 elements per vector.
End subroutine

Area = ½*abs(sqrt(sum(xProduct).^2))  % You must use Pythagorean theorem since xProduct is an array that contains the three sides of the
triangle.

End Program
```