Notes on Algorithms, Pseudocode, and Flowcharts

Introduction

Do you like hot sauce? Here is an 'algorithm' for how to make a good one:

Volcanic Hot Sauce (from: http://recipeland.com/recipe/v/Volcanic-Hot-Sauce-1125)

10-12 scotch bonnets or Habanero, serrano, jalapeno
6 cloves Garlic, peeled and chopped
1/3 c Fresh lime juice
1/3 c Distilled white vinegar
2 tbl Dijon style mustard
2 tbl Olive oil
1 tsp Molasses
1/2 tsp Turmeric
1 tbl Salt or to taste

1. Combine the pepper, garlic, lime juice, vinegar, mustard, oil, molasses, turmeric, and salt in a blender and puree until smooth. Correct the seasoning, adding more salt or molasses to taste.
2. Transfer the sauce to a clean bottle. You can use it right away, but the flavor will improve if you let it age for a few days. Volcanic Hot Sauce will keep almost indefinitely, refrigerated or at room temperature. Just give it a good shake before using.

As you can see, this 'algorithm' is a really a *recipe*, that is, a set of step-by-step instructions that takes raw ingredients and produces a tasty result. In general, an algorithm can be described as a procedure to solve a problem.

In the context of computer programming, an *algorithm*, is defined as a:

"well-ordered collection of unambiguous and effectively computable operations, that when executed, produces a result and halts in a finite amount of time."[1]

Characteristics of an Algorithm

- Well-ordered: the steps are in a clear order
- Unambiguous: the operations described are understood by a computing agent without further simplification
- Effectively computable: the computing agent can actually carry out the operation

Method for Developing an Algorithm

1. Define the problem: State the problem you are trying to solve in *clear* and *concise* terms.
2. List the *inputs* (information needed to solve the problem) and the *outputs* (what the algorithm will produce as a result)
3. Describe the steps needed to convert or manipulate the inputs to produce the outputs. Start at a high level first, and keep refining the steps until they are *effectively computable* operations.
4. Test the algorithm: choose data sets and verify that your algorithm works!

[1] definition from: An Invitation to Computer Science (Gersting/Schneider) via http://www.cs.xu.edu/csci170/08f/sect01/Overheads/WhatIsAnAlgorithm.html (visited 19JUN2009)

Structured Programming

- In 1966, computer scientists Corrado Böhm and Giuseppe Jacopini demonstrated that all programs could be written using three control structures: Sequence, Selection, and Repetition[2].
- The *__sequence__* structure is the construct where one statement is executed after another
- The *__selection__* structure is the construct where statements can executed or skipped depending on whether a condition evaluates to TRUE or FALSE
  - There are three selection structures in C:
    1. IF
    2. IF – ELSE
    3. SWITCH
- The *__repetition__* structure is the construct where statements can be executed repeatedly until a condition evaluates to TRUE or FALSE
  - There are three repetition structures in C:
    1. WHILE
    2. DO – WHILE
    3. FOR

Pseudocode (or Program Design Language)

- Consists of natural language-like statements that precisely describe the steps of an algorithm or program
- Statements describe *actions*[3]
- Focuses on the *__logic__* of the algorithm or program
- Avoids language-specific elements
- Written at a level so that the desired programming code can be generated almost automatically from each statement
- Steps are numbered. Subordinate numbers and/or indentation are used for dependent statements in selection and repetition structures[4].

[2] Corrado; B. and Jacopini, G. (May 1966). "Flow Diagrams, Turing Machines and Languages with Only Two Formation Rules". Communications of the ACM 9 (5): 366–371.

[3] Some programmers also include data declarations in their pseudocode. I think this is a good idea, because it helps you keep track of the variables that you will need in the algorithm and can help you think through what data types are needed.

[4] Some programmers will add an ending 'keyword' on a separate line to make it explicit where a selection or repetition structure ends, for example: **ENDIF**, **ENDWHILE**, etc. On the one hand, this is good because it makes it clear where the selection or repetition block ends, but on the other hand it adds to the length of the pseudocode, and such statements will not 'translate' into an actual line of code in C. In a language like Pascal however, they will.

Pseudocode Language Constructs[5]

- Computation/Assignment
  - **Compute** var1 as the sum of x and y
  - **Assign** expression to var2
  - **Increment** counter1
- Input/Output
  - Input: **Get** var1, var2, …
  - Output: **Display** var1, var2, …
- Selection

  Single-Selection IF

```
1. IF condition THEN (IF condition is true, then do subordinate statement 1, etc. If
   condition is false, then skip statements)
      1.1 statement 1
      1.2 etc.
```

  Double-Selection IF

```
2. IF condition THEN (IF condition is true, then do subordinate statement 1, etc. If
   condition is false, then skip statements and execute statements under ELSE)
      2.1 statement 1
      2.2 etc.
3. ELSE (else if condition is not true, then do subordinate statement 2, etc.)
      3.1 statement 2
      3.2 statement 3
4. SWITCH expression TO
      4.1 case 1: action1
      4.2 case 2: action2
      4.3 etc.
      4.4 default: actionx
```

- Repetition

```
5. WHILE condition (while condition is true, then do subordinate statements)
      5.1 statement 1
      5.2 etc.
```

  DO – WHILE structure (like WHILE, but tests condition at the *end* of the loop. Thus, statements in the structure will always be executed at least once.)

```
6. DO
      6.1 statement 1
      6.2 etc.
7. WHILE condition
```

[5] See "Pseudocode Standard" at http://users.csc.calpoly.edu/~jdalbey/SWE/pdl_std.html for more examples.

FOR structure (a specialized version of WHILE for repeating execution of statements a specific number of times)

```
8. FOR bounds on repetition
    8.1 statement 1
    8.2 etc.
```

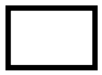
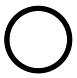
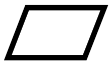
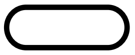
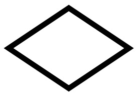
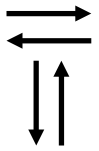
Pseudocode Example

Express an algorithm to get two numbers from the user (dividend and divisor), testing to make sure that the divisor number is not zero, and displaying their quotient using *pseudocode*

```
1. Declare variables: dividend, divisor, quotient
2. Prompt user to enter dividend and divisor
3. Get dividend and divisor
4. IF divisor is equal to zero, THEN
    4.1. DO
        4.1.1. Display error message, "divisor must be non-zero"
        4.1.2. Prompt user to enter divisor
        4.1.3. Get divisor
    4.2. WHILE divisor is equal to zero
5. ENDIF
6. Display dividend and divisor
7. Calculate quotient as dividend/divisor
8. Display quotient
```

Flowcharts

- A graphical tool that *diagrammatically* depicts the steps and structure of an algorithm or program
- Symbols[6,7] (the most commonly used ones)

Symbol	Name/Meaning	Symbol	Meaning
	Process – Any type of internal operation: data transformation, data movement, logic operation, etc.		Connector – connects sections of the flowchart, so that the diagram can maintain a smooth, linear flow
	Input/Output – input or output of data		Terminal – indicates start or end of the program or algorithm
	Decision – evaluates a condition or statement and branches depending on whether the evaluation is true or false		Flow lines – ***arrows*** that indicate the direction of the progression of the program

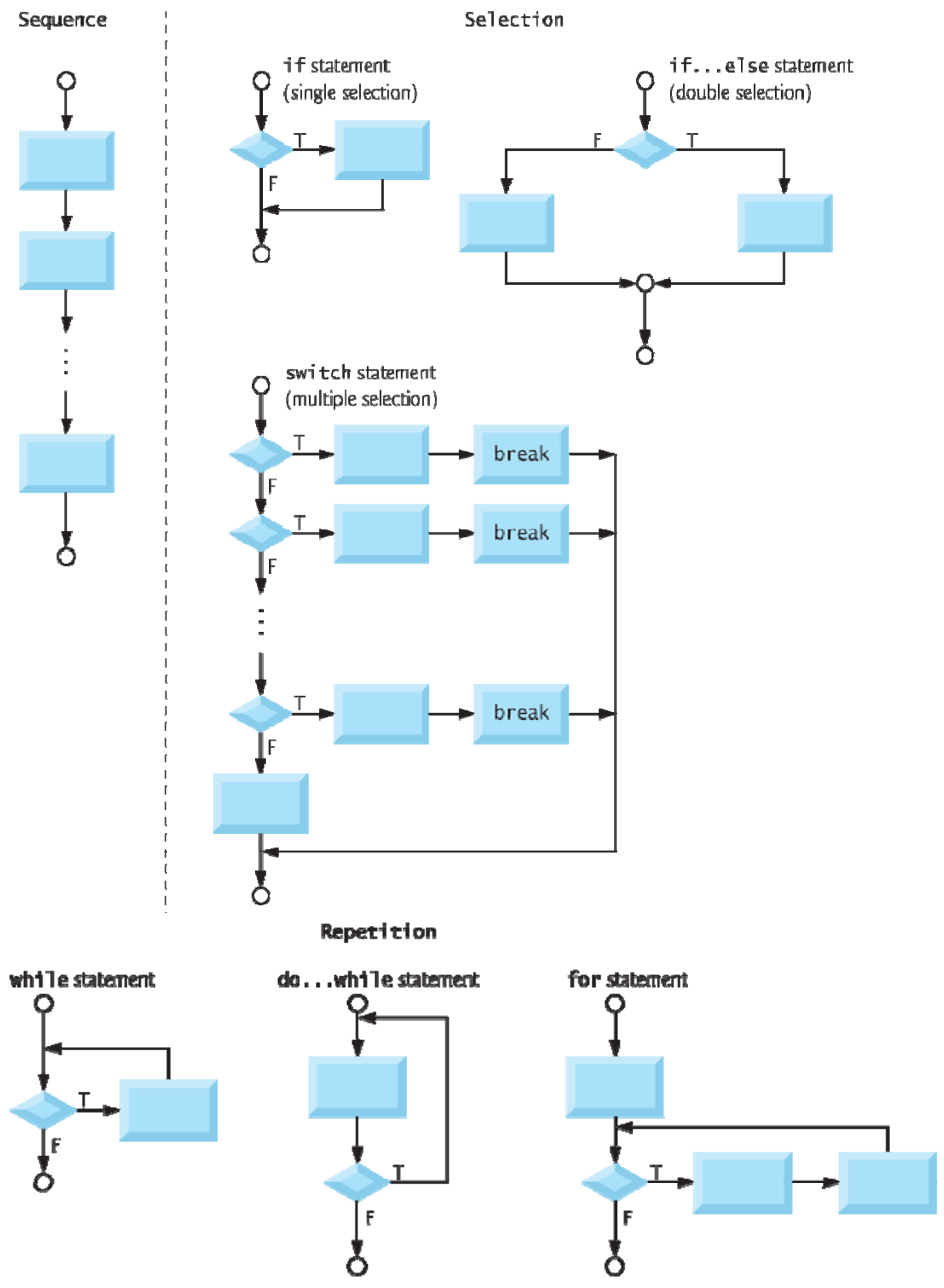
[6] For a comprehensive tutorial on flowcharting, see: Chapin, N. (1970). Flowcharting With the ANSI Standard: A Tutorial, ACM Computing Surveys (CSUR), vol. 2, issue 2, pp. 119 – 146.

[7] MS Word incorporates flowchart symbols in the Draw toolbar. After drawing a symbol, right click to pop-up and select, 'Add text', to easily insert a text expression into the symbol.

- General rules for flowcharts
  - All symbols of the flowchart are connected by flow lines (note *arrows*, not lines)
  - Flowlines enter the top of the symbol and exit out the bottom, except for the Decision symbol, which can have flow lines exiting from the bottom or the sides
  - Flowcharts are drawn so flow generally goes from top to bottom
  - The beginning and the end of the flowchart is indicated using the Terminal symbol

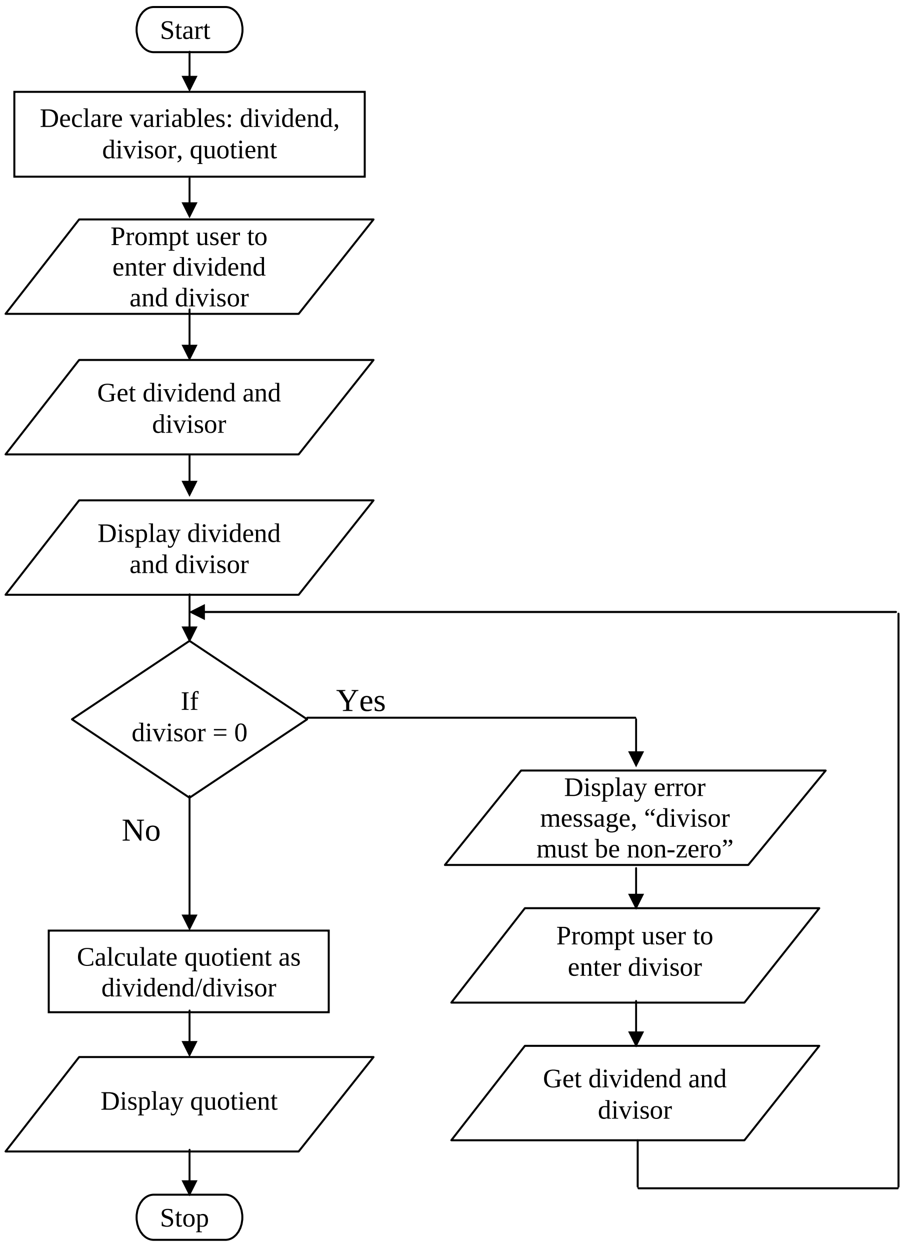
Flowchart Constructs (from Deitel & Deitel, 6th ed., p. 122)

- The flowchart equivalents for the structured programming constructs described earlier are:



Flowchart Example

Express an algorithm to get two numbers from the user (dividend and divisor), testing to make sure that the divisor number is not zero, and displaying their quotient using a *flowchart*.


Start
Declare variables: dividend, divisor, quotient
Prompt user to enter dividend and divisor
Get dividend and divisor
Display dividend and divisor
If divisor = 0
Yes
No
Display error message, "divisor must be non-zero"
Prompt user to enter divisor
Get dividend and divisor
Calculate quotient as dividend/divisor
Display quotient
Stop