Chp 4: Branching

- This chapter has the following important parts:
  - Basics: Why Branching?
  - Relational and Logical Operators
  - if / if-else / if-else if-else / nested if
  - The switch statement
  - Conditional operator

(1)
- Basics: Branching? Why and How?
  - We need something more than sequential logic in programming
  - based on condition(s) during program runtime, computer can decide which program path to go
    - decision making during runtime
  - Two types of basic constructs:
    - if and else
      - binary or two-way selection at one time
      - based on a condition
    - switch
      - multi-way selection at one time
      - based on a value
  - note: flowcharts are good to show branching logic

(2)
- Relational and Logical Operators
  - Purpose: create logical expression for branching conditions
  - Relational Operators
    - Used for comparison between two values.
    - Return boolean result: true or false
    - Operators: ==, !=, <, <=, >, and >=
    - Caution! When comparing equality of two float values
  - Logical Operators
    - Combine one or more relational expressions
    - Yield a logical value: true or false.
    - Operators: !, &&, and ||
      (not, and, or)
    - note the truth table
    - decreasing precedence
      ! -> very high precedence
      && and || -> very low precedence
      (see P.10)

(3)
- if / if-else / if-else if-else
  - Note:
    - You MUST understand the logic!!!
    - It is very mechanical when tracing the program!!!
    - See, understand, and practise some examples!!!
    - programming style: proper indentation
  - if statement
    - Syntax:
      ```
      if (Expression)
          Statement;
      ```
    - Note:
      Statement may be
      (1) a single statement terminated by a semicolon; or
      (2) a compound statement enclosed by { }
  - if-else statement
    - Syntax:
      ```
      if (Expression)
          Statement_1;
      else
          Statement_2;
      ```
    - Note:
      Both Statement_1 and Statement_2 may be a single statement terminated by a semicolon or a compound statement enclosed by { }

```
  - if-else if-else Statement
    - Syntax:
      if ( Expression_1 )
          Statement_1;
      else
      if ( Expression_2 )
          Statement_2;
      else
          Statement_3;
    - Note:
      Each of Statement_1, Statement_2 and Statement_3  may be
      a single statement terminated by a semicolon or a compound
      statement enclosed by { }

(4)
- Nested-if Statement
  - Both the if branch and the else branch may contain if statement(s)
  - The level of nested if statements can be as many as we want (up
    to the compiler limit)
  - See, understand, and practise some examples!!!
  - Important Rule:
    Associates an else part with the nearest unresolved if

(5)
- The switch Statement
  - Basics:
    - Multi-way control flow
    - By matching an integral value during program runtime
  - Syntax:
    - switch ( Expression ) {
         case Constant_1:
            Statement_1;
            break;
         case Constant_2:
            Statement_2;
            break;
         case Constant_3:
            Statement_3;
            break;
         default :
            Statement_d;  // run this if no match
      }
  - Note:
    - switch, case, break and default are reserved words
    - The result of Expression must be integral type
    - Constant_1, Constant_2, ... are called labels.
      - Must be an integer constant, a character constant
        or an integer constant expression
      - unique integer value; duplicates are not allowed
    - may also have multiple labels for a statement
      e.g.,
         case 'a' :
         case 'A' :
            statement ;
            break ;
    - default is optional
    - fall through:
      If we DO NOT use break after some statements in the
      switch statement, execution will continue with the
      statements for the subsequent labels until "break"

- The Conditional Operator
  - Syntax:
    Expression_1 ? Expression_2 : Expression_3
  - Meaning:
    - If Expression_1 is true, use value of Expression_2 as the
      value of the entire expression resulted from this operator
      Else, use that of Expression_3
  - It is a short form, e.g.,
```

```
   max = (x > y) ? x : y ;
is the same as
   if ( x > y )
      max = x ;
   else
      max = y ;
```