CS Bridge Module 5: Branching Statements Part 2

- Another Control Flow we can use in C++ is the Branching Statement; "Switch"

Switch Statement

Syntax:
```
switch (numeric-expression) {
    case constant:
      ...
      break;
    case constant:
      ...
      break;
    default:
      ...
      break;
}
```

- "switch" keyword with a numeric expression to be evaluated
  - "case" clases with constant values
    - statement / instructions to be performed
  - "break" stops the Evaluation

Semantics:
- When execution reaches a "switch statement"
- First, numeric expression is evaluated
- Secondly the numeric expression is compared to "constant one"
  - if the numeric expression and constant are equal, then the case expression/instructions will be executed
- Lastly, After the expression is evaluated, "break" keyword is evaluated, and would "break" out of the switch statement.

Switch Statement - Syntactic Notes

- "Switch" statements are LESS powerful than the "if-else, if, else" statement

- Stuff that can be done with an multi-way "if" can't necesscarily be done with a "switch"

- The condition in an "if" statement can be a complex boolean expression

- The "switch" statement is only comparing numeric values

- Switch statements are useful when we want to implement menus,

Syntactic Notes:

- Numeric expression must be either; int (short int, long int, int), char, or bool
  - Cannot be double, float or complex defined types

- The "case" lables MUST be Constants (literals or named constants)
  - Cannot be expressions, variable names

- If no "case" label matches the value of the "numeric expression", control branches to the default label.
  - If no "default" label, nothing is executed in the "switch" statement and control passes to the following statement after "switch"

- After a branch is taken, the control proceeds until a "break" is reached
  - If there is no "break" statement, control falls to the next switch statement UNTIL we reach the next "break"