STYLE GUIDELINES
CS2308 Spring 2020
Jill Seaman

Header comments (file documentation block) should be at the top of each file and should contain: File Name, Author(s), Date, Assignment Number, Course number and section, Instructor, and a brief description of the purpose of the code in the file.

```
// File Name: assign1_js108.cpp
//
// Author: (Your name here, followed by collaborators, if any)
// Date: 2/3/2020
// Assignment Number: 1
// CS 2308.255 Spring 2020
// Instructor: Jill Seaman
//
// (Description of what the program does goes here).
```

#include directives must follow header comments, before the rest of the program.

Variable names:
--must be meaningful
--loop index names can be simple (i, j, k, etc)
--The initial letter should be lowercase, following words should be capitalized, no other caps or punctuation (ie: `weightInPounds`). This is called "**camel case**".

Named constants:
—use for most numeric literals (including array sizes).
—name should be all capitals with underscores:

```
const double TAX_RATE = 0.0675;
```

—should occur near the top of the program (not inside functions).

Line length of source code should be no longer than 80 characters (no wrapping of lines).

Indentation:
--Use 2-4 spaces (but be consistent throughout your program).
--Indent blocks, blocks within blocks, etc.
--Use blank lines to separate sections.

Comments for variables:
All variable declarations should be commented as follows, with a description:

```
int  rank;   // numeric value for a card, A=1, J=11, Q=12, K=13
```

Comments for functions:

Function definitions should be commented to describe what the function does, what the parameters are, and what the function returns (when appropriate). This comment should precede the function definition. See the template and the example below. If the function body contains more than about five statements, there should be comments to describe the various sections of code in the function body.

Template:

```
//*********************************************************
// function name: short description of what the function does.
//
// param-1 description of first parameter (if any)
// param-2 description of second parameter (if any)
// (remaining params, if any)
// returns: description of what function returns (if not void)
//*********************************************************
```

Example:

```
//*********************************************************
// getBestPlayer: determines which player scored the most points
// p the array of player information
// size the number of players in the array
// returns the name of player who scored the most points
//*********************************************************

string getBestPlayer(Player p[], int size) {
   // function body goes here
}
```

In-code comments:

DO NOT comment every line of code! In general, try to avoid using comments that describe WHAT the code is doing. These are redundant (we can just read the code). Comments that explain WHY the code is doing what it is doing are more helpful. Try to minimize in-code comments, and write readable code instead.

Follow these recognized good programming practices:

The grader may deduct for these issues:

1. Use appropriate data types:

```
double populationSize; // you cannot have a fractional amount
                       // of people like 2008.55, use int
```

2. Avoid duplicate code (don't copy, paste and modify):

```
if (monthlySales > 3000) {
      cout << "Commission: $" << price * 0.25 << endl;
   }
   else {
      cout << "Commission: $" << price * 0.29 << endl;
   }
```

better:

```
double rate;
if (monthlySales > 3000) {
    rate = 0.25;
}
else {
    rate = 0.29;
}
cout << "Commission: $" << price * rate << endl;
```

3. Do not *use* uninitialized variables:

```
int total;          //should be initialized to 0;
for (..;..;..)
   total = total + x;  //on first use, total has garbage in it
```

4. Use a named constant for an array size:

```
const int SIZE = 100;      //NOT: int SIZE;
. . .
double myArray[SIZE];
```

5. Avoid out of bounds array access:
   for example:

```
for (int i=0; i<=SIZE; i++) { // when i == SIZE it goes
                              // beyond the end of the array
  . . . myArray[i] . . .
}
```

6. Do not use global variables (but global named constants are good).

7. Use reference parameters only when necessary.