CSCI 240
Documentation Standards

General Program Comments

The following standard general comments should be included in every C program that you write and turn in:

- Course, Assignment Number, Semester
- Programmer's name
- Section Number
- TA's Name
- Date Due
- General statement of program's purpose, including input and output.

This block of comments should be at the top of your source code, before anything else. For example:

```
/*********************************************************************
CSCI 240 - Assignment X - Semester (Fall/Spring) Year

Progammer: Your name goes here
Section:   Your section number goes here
TA:        Your Teaching Assistant's name goes here
Date Due:  The assignment due date goes here

Purpose:   A brief (2-4 sentences) description of what the program does
           goes here. For example:

           This program accepts a single number from the keyboard
           representing a temperature in Fahrenheit. It then converts
           it to Centigrade, and displays it.
*********************************************************************/
```

Variable Names

Most variable names in your program should describe the quantity or item that they represent. For example, if a variable is to hold a student's test average, don't call it "sa"; call it "studentAverage" or "student_average". If a variable is to hold a person's name, don't call it "s" or "n"; call it "name" or better yet, "studentName" or "employee_name". If a variable is to hold the number of students in a class, don't name it "n" or even "num" or "count"; do name it "studentCount" or "numberOfStudents".

The exception to this rule is for temporary variables that have no intrinsic meaning, especially "counting variables" in for loops:

```
for( i = 0; i < 10; i++ )
...etc...
```

When declaring variables, group and format them in a neat and logical manner.

Bad:

```
int sum=0,numGizmos=0,gizmoID,squareOfGizmos=0,i,j;
double avgGizmos,overallAvg,gizmoSDT,this,that,theOtherThing;
```

Good:

```
int sum           = 0,
    numGizmos     = 0,
    squareOfGizmos = 0,
    gizmoID,
    i,
    j;

double avgGizmos,
       overallAvg,
       gizmoSDT,
       thisThing,
       thatThing,
       theOtherThing;
```

We will not require that every variable be declared on a new line, but it is generally a good idea.

Line And Section Documentation

Line documentation consists of a comment about a single line of code. Often it is on the same line as the code:

```
ave = sum / count;    // calculate the average score
```

If the comment is very long, place it above the line so you don't create "line wrap" when you print:

```
// calculate the area of the triangle: semi is semiperimeter
// s1, s2, and s3 are sides

Area = sqrt(semi * (semi - s1) * (semi - s2) * (semi - s3));
```

A moderate amount of documentation in the main body of your program may be advisable, but if you name your variables and functions with meaningful names, you should not need much. If you name variables well, many programs won't need *any* Line Documentation. The following line does not need documentation.

```
pgmAvg = sumOfPgmPoints / NumPgms;
```

You can use either the

```
/*

some comment

*/
```

or the

```
// some comment
```

format.

Section documentation tells the reader about the section of code that follows. It is usually placed before a loop or decision construct or a series of lines that do a single task. Examples:

```
// Loop to accept and process user-supplied glucose measurements

// Decide if gizmos or widgets are to be used

// Calculate and store averages and standard deviations of measurements
```

Your programs should use a moderate amount of Section Documentation for functions (such as *main()*) that are long or have several distinct tasks. You should put a blank line before any Section Documentation and indent it the same amount as the code block that it describes. Obviously, use of Section documentation is partly a matter of personal judgment - but your programs should include at least a minimal amount of Section Documentation.

Indentation

You must **consistently indent** the body of loops and the alternate blocks of a decision structure. Indent at least two spaces; you may prefer three or four (but no more than 4). The following may be taken as samples:

```
x = 10;
while (x > 0)
  {
  cout << "\n" << x;
  x--;
  }

if (x > 0)
  {
  answer = x * 2;
  cout << "x was positive; answer is " << answer;
  }
else
  {
  answer = -x * 2;
  cout << "x was negative; answer is " << answer;
  }
```

Each nested structure must have its own level of indenting:

```
x = 10;
while (x > 0)
  {
  cout << "\n" << x;
  if (x % 2 == 0)
    cout << "\nx is even";
  else
    cout << "\nx is odd);
  }
```

Note on curly braces:

Many people prefer other arrangements of curly braces.

```
while (x > 0) {
some code...
some more code...
}

while (x > 0)
{
  some code
  some more code
}
```

However, for this course, use the recommended formatting shown above.

Use of White Space

In general, use spaces to help the human reader.

Good	Bad
for (i = 0; i < 10; i++)	for (i=0;i<10;i++)
total = calcTotal(amt, taxRate, tip);	t=fn(a,tr,t);
int i, j, k;	int i,j,k;

Also, between logical sections of the program (or between functions), insert blank lines to help the reader find the main sections. Often each of these sections should have Section Documentation to explain its role in the program. For example:

```
int main()
 {
 int num1, num2, GCF;    // Obtain user input of two integers

 cout << "Enter ...";
 cin >> num1;

 ...                     // Calculate Greatest Common Factor
 ... code to do the calculation ...  //Display result
 cout << "... ";

 ...
 }
```

Function Documentation

Each function that you write for a given program must have a documentation box explaining:

- its name
- its use or function: that is, what does it do? What service does it provide to the code that calls it?
- a list of its arguments briefly describing the meaning and use of each, and in particular whether the function modifies each one
- the value returned (if any) or none.
- notes on any unusual features, assumptions, techniques, etc.

Example 1:

```
/*************************************************************
Function: Sort

Use:      Sorts an array of integers in ascending order using
          Selection sort algorithm

Arguments: 1. Ar: an array of integers (modified by Sort).
           2. numEntries: the number of entries to be sorted.

Returns:   nothing
*************************************************************/

void Sort(int Ar[], int numEntries)
 {
 // code here...
 }
```

Example 2:

```
/*************************************************************
Function: calcTriArea

Use:      calculates the area of a triangle, given the length
          of the three sides.

Arguments: three double values, each representing the length of
           one of the sides.

Returns:  the area of the triangle, or -1 if the lengths do not
          represent a valid (closed) triangle

Notes:    uses Hero's formula; if the quantity whose square
          root is to be found is negative, this signals an
          invalid triangle, and the function returns -1.
*************************************************************/

int calcTriArea(double s1, double s2, double s3)
 {
 // code here...
 }
```