Additional Standards for C Programs in CS120

1. **Local Variable Declarations/Definitions**
   All variable declarations need to have a comment to the right that explains the purpose of the variable. There are no exceptions to this rule. No single-letter variable names are allowed. The only exceptions to the "single-letter" rule are variables used as indexes in a for loop. Only one variable per line may be declared. The exception to the "one variable per line" rule is if you have several loop counters and you want to declare them at the same time. If the comment is too long and won't fit to the right, put it on the line above the variable. All comments that are to the right need to be aligned and should be close to the longest variable declaration as such:

```
  int i, j, k;  /* for loop counters, single—letter OK here */
  int tabs = 0; /* Number of tabs in file                   */
  int length;   /* Length of string                         */
  int begin;    /* The left end of the string               */
  int end;      /* The right end of the string              */
```

2. **File Header Comments**
   All files must contain a comment at the top with information such as the student's name, the section, due date, etc., all aligned properly. This is an example of a file header comment:

```
/******************************************************************************
filename    marathon.c
author      Nigel Tufnel
DP email    tap@digipen.edu
course      CS120
section     D
assignment  9
due date    1/21/2018

Brief Description:
  This program contains the calculation to convert the distance of a
  marathon in miles to kilometers.

******************************************************************************/
```

3. **Function Header Comments**
   All functions must include a function header comment immediately above the function. There are 4 parts to the comment: The function name, a description, inputs, and outputs. The fields must be named as shown and must line up exactly as shown. There should be no blank lines between the header comment and the function itself. However, there should be at least one blank line between the closing brace of the function body and the following function header comment (which is for the function that follows). Here is an example:

```
/******************************************************************************
   Function: read

Description: Reads bytes from a file into a buffer.

     Inputs: fd     - The file descriptor to read from.
             buffer - The buffer to hold the read bytes. The buffer must be
                      large enough to hold all of the bytes.
             count  - The number of bytes to read.

    Outputs: The number of bytes actually read in. If an error occurs, -1
             is returned.
******************************************************************************/
ssize_t read(int fd, void *buffer, size_t count);
```

4. **Functions declarations and definitions**
   These can not be over 80 characters in length, so you will have to format them manually. The function declaration for *jumble* is over 100 characters, so you need to put it on multiple lines. Notice how the parameters are all left aligned right after the left parenthesis. This formatting also allows you to put a comment next to each parameter, if you need to describe them:

```
void jumble(unsigned char *string,          /* a comment could go here */
            const unsigned char *password, /* a comment could go here */
            enum CODE_METHOD method,       /* a comment could go here */
            int passes                     /* a comment could go here */
           );
```

5. **Formatting function code**
   Most functions that return a value will have 3 sections: declarations, executable statements, return statement. There should be a blank line between these sections. Some functions will have several blocks of executable statements and they should also be separated by a blank line.

```
int foo(int param1, int param2)
{
  int var1 = 0; /* comment */
  int var2 = 1; /* comment */
  int var3 = 2; /* comment */

    /* any necessary comments can go here */
  for (var1 = 0; < param1; var1++)
  {
    var2 += param2;          /* or     */
    var3 -= var2 - param2;   /* they   */
    if (var3 == 0)           /* can go */
      var2 = 0;              /* here   */
  }

  return var1 * var2; /* comment */
}
```

6. **Internal Documentation (comments)**
   All code needs to have some documentation associated with it. The amount of detail in the comments depends on the code. Typically, more complex code requires more detailed documentation. Comments can either be placed to the right of the statements they are describing, or on a line immediately above it. Don't put comments to the right unless the comment will fit on one line. Remember, your comments should be explaining the purpose of the code, not re-stating the obvious. Sometimes even a simple statement can be more easily understood by the programmer if it includes a useful comment. The first comment below is useless. The second comment says something that will help a programmer reading the code:

```
pCursor++; /* Increment pCursor (useless comment)    */
pCursor++; /* Move to the next line in the paragraph */
```

   You can put the comment above the statement instead of to the right, but never below the statement:

```
  /* Increment pCursor (useless comment)     */
pCursor++;

  /* Move to the next line in the paragraph */
pCursor++;
```

7. **Multi-line comments**
   Multi-line comments should go above the statement(s) they describe rather than to the right. When using multiple lines for comments, you should format them as such:

```
/*
 * This is a long comment because the line of code below is too
 * complicated to describe in one line. I should really clean up
 * the code so I don't have to write a book about it. But, I don't
 * have enough time to write short code with short comments, so I'm
 * writing this massive missive instead.
 */
```

8. **Documenting #include directives**
   All included header files must have a comment next to them listing the reason for the included files. You only need to list the functions (or symbols) that you are using from the header file. Don't include a header file if you are not using anything from it.

```
#include <stdio.h>  /* printf, scanf, fopen */
#include <stdlib.h> /* rand, atoi, srand    */
#include <time.h>   /* time, clock          */
#include <math.h>   /* sqrt                 */
```

9. **Global variables**
   Global variables are not permitted in any code. The only global objects are functions, enumerations, constants, and #defines. This means that all variables must be declared/defined inside of a function. Global variables are an advanced construct that is not necessary for CS120 programming.

10. **The goto keyword**
    The use of the **goto** keyword is not allowed. Like global variables, the **goto** keyword is an advanced construct and is not necessary in any programs in CS120.

11. **Line length**
    No lines longer than 80 characters are allowed. This is generally not a problem because expressions longer than this are very difficult to understand. However, where you might encounter a line that long is with output statements (e.g. **printf**). Don't feel the need to print everything with a single **printf** call. You can make multiple calls with shorter lines:

```
printf("If the rate is %i feet per second", rate);
printf(" and the time is %i seconds,", time);
printf(" the distance traveled is %i feet.\n", distance);
```

    If you tried to do this in a single printf call, you would have a very long line. Here's another example that has a single call to printf, but wraps the arguments to the next line:

```
printf("Bob's weight is %i lbs, his height %i inches, and his age is %i.\n",
       weight, height, age);
```

    In this case, you want to indent the second line so it lines up with the first double-quote in the line above.

12. **Tabs vs. Spaces for indentation** Proper indentation is a must for readability. In this class, the proper size for indentation is 2 spaces. No tabs. Use spaces instead. Most text editors allow you to configure them so that, when you press the tab key, spaces are inserted instead of the tab character.

13. **Braces**
    Curly braces should always be on a line by themselves and should be aligned vertically. This includes curly braces used in function definitions.

Proper formatting for a 'for' statement

```
for (i = 0; i < 10; i++)
  a[i] = 0;

for (i = 0; i < 10; i++)
{
  if (calc(i) < 0)
    break;

  a[i] = calc(i * i);
  b[i] = calc(a[i]);
}
```

Proper formatting for a 'while' statement

```
while (i < 10)
  a[i++] = 0;

while (i < 10)
{
  a[i] = 0;
  i++;
}
```

Proper formatting for an 'if..else if' statement

```
if (year == 1)
  printf("Freshman\n");
else if (year == 2)
  printf("Sophomore\n");
else if (year == 3)
  printf("Junior\n");
else if (year == 4)
  printf("Senior\n");
else
{
  printf("Invalid year\n");
  printf("Please re-enter it.\n");
}
```

Proper formatting for a 'switch' statement

```
switch (year)
{
  case 1:
    printf("Freshman\n");
    break;
  case 2:
    printf("Sophomore\n");
    break;
  case 3:
    printf("Junior\n");
    break;
  case 4:
    printf("Senior\n");
    break;
  default:
    printf("Invalid year\n");
    break;
}
```

Proper formatting for a 'do...while' statement

```
do
{
  printf("Enter a number: ");
  scanf("%d", &number);

  printf("You entered %i\n", number);

  printf("Enter another number? (1=yes,0=no) ");
  scanf("%d", &choice);
}
while (choice != 0);
```

14. **#defines**
Defines for the preprocessor should be used sparingly and should **only** be used to introduce a constant into the program. All defines must be in all uppercase:

```
#define YARDS_PER_MILE 1760F
#define KILOS_PER_MILE 1.609
#define PI 3.1415926535
```

C++ note: The const keyword behaves differently in C than in C++, so often you must use a #define in C where a const would be used in C++.

15. **Operators and Spacing**
Spaces must surround all binary operators. Unary operators should not have any extra spaces:

```
r = (w + x) * (y + z);
a = -b + c;
f = sqrt(b * b - 4 * a * c);
w = --x + (y++ / z);
theta = sin(a) * -cos(b);
```

16. **Function declarations (prototypes)**
    Although function prototypes do not require names for their parameters, it's good practice to do so. In CS120, all function declarations must include meaningful names for the parameters.

```
  /* Acceptable prototype */
int Calculate(float rate, float time, float distance);

  /* Unacceptable prototypes */
int Calculate(float r, float t, float d);
int Calculate(float, float, float);
```

17. **Calling functions**
    No space between the function name and left parenthesis. (The function-call operator is a unary operator.) If there are arguments, they should have spaces between them:

```
strong();
distance(rate, time);
weak(a, b, c);
funky(1, 5, c + d, e, f);
```