Program Standards

The following standards are to be followed in writing C programs. They are important enough to be part of the grading criteria for all programs. Any constructs not covered in this standard are to be implemented in a consistent and readable fashion.

PURPOSE

The purpose of these rules are (1) to introduce you to a representative set of coding standards that are typical in professional programming and (2) to help you develop good style as a habit, something you will find necessary to successfully complete large programs. There is no doubt that you can write C programs to solve the small problems in this course without attention to style. But, you must keep in mind that you are building professional skills, not merely finding the answers to problems handed out as exercises.

FILE NAMES AND C COMPILER

In this course each C file name should clearly represent what it is and each should have the ".c" suffix. Files names when not specified by the project definition must be meaningful. For example, Project 1 could be project_1.c.

INTRODUCTORY DOCUMENTATION

At the beginning of each source code file, there should be a comment block that contains:

1. The program name, your name, your student ID and your section ID.
2. The date the program was completed.

Additionally, for the file that contains the main() function, include a brief description of the program that describes the method or algorithm used to obtain the solution, the major data structures (if any) in the program, and other general comments including extra features.

DECLARATIONS

1. You should use **mnemonic** names for all identifiers. These are names that have meaning in the problem. The names should suggest the use of the identifier.
2. Variable identifiers should be in all lower-case letters, and multiple words in variable names must be separated by and underscore character (_) e.g. answer, first_answer, temperature_reading
3. Constant identifiers should always appear in all-capital letters; underscore characters delimit multiple words. For example:

```
const float BASE = 2.0;       // divisor to obtain base 2 final
const int NAME_LENGTH = 20;   // names can be 20 characters
```

4. Functions should begin with a lowercase letter, and have any multiple words separated by an underscore character (for example, is_valid_number(), area(), set_width(), etc).
5. Each identifier declaration (including function declarations) must be accompanied by a comment that explains its use.
6. Avoid declaring globally-visible variables. Globally-visible constants, enumerated types and file names are okay, but do not use global variables unless absolutely necessary.

GENERAL RULES

1. Each constant and variable declaration should be on a separate line that includes a comment that explains what each is. For example:

```
float volts;                    // voltage in the circuit
int amps;                       // amperage in the circuit
char circuit_name[NAME_LENGTH]; // name of the circuit
```

2. Each statement should be on a line by itself. For example, instead of

```
left = next_left;    right = next_right;
```

use

```
left  = next_left;     // move to the left
right = nextRight;    // move to the right
```

3. Each function should be identified by a comment block describing its purpose, return type and arguments(s). The format to be used is as follows:

```
/************************************************************
*
* calculates and returns the charge for shipping cargo
* between two planets.
*
* distance:      distance in miles between two planets
* weight:        weight in pounds of item being shipped
*
* returns:       A double representing the cost to ship the item
*                in space
*
************************************************************/

double find_space_cost(long distance, float weight)
{
    // implement here
}
```

4. Comments should be set apart from the code by at least two spaces, making it easy to see where the code ends and comment begins. Align multiple comments for easy readability.

FORMATTING AND COMMENTING OF STATEMENTS

Alignment and indentation of lines of C statements add much to the readability of programs. The following guidelines are general so that you may develop your own programming style. Use indentation and blank lines as you see fit to increase the readability of your programs. Use whitespace (blank lines and spaces on lines) liberally! Use parentheses liberally!!

The body of a control structure, such as an **if**, a **switch**, a **while**, a **do-while**, or a **for** statement, should be indented. The following examples illustrate the only formatting style that is to be used. Each level of indenting is 4 spaces. (All of the below examples should have 4 spaces for indentation).

if statement

```
if (expression)
{
    statements
}
```

if-else statement

```
if (expression)
{
    statements
}

else
{
    statements
}
```

switch statement

```
switch (selector variable)
{
    case case-1-value:
        case 1 statements;
        break;

    case case-2-value:
        case 2 statements;
        break;

    ...

    case case-n-value:
        case n statements;
        break;

    default:
        default statements;
        break;
}
```

while statement

```
while (expression)
{
    statements
}
```

do-while statement

```
do
{
    statements
} while (expression);
```

for statement

```
for (initialization; condition; increment)
{
    statements
}
```

PROGRAM DESIGN

1. Use stepwise refinement. The programming assignments given are ones that can be solved in a step-by-step manner. The method of solution used in your program should exhibit this kind of structure.
2. Modularize your program. When a problem solution readily splits into distinct tasks, your program should handle each task in a separate function and in a straightforward manner.
3. Make your program efficient. Your algorithm should be direct and not unnecessarily complicated. Proper use of data structures will often simplify an algorithm.
4. Use parameters appropriately. All variables used in a method should either be declared as parameters or be declared locally within the method.

OUTPUT

5. All input values should be echoed in a readable manner unless otherwise indicated.
6. The program output should be readable and clearly labeled. Headings should be used when appropriate.
7. Finally, of course, your program should produce correct results!