Guidelines for Writing C Programs

©Professor Hossein Saiedian
Department of EE & Computer Science
College of Engineering — University of Kansas

1 Source Program Requirements

For each programming assignment, you must turn in both the source program as well as its output. Be prepared to turn in the documentations for the initial phases of program development, namely input/output analysis and algorithm development. In addition, you will be asked to provide an executable code that would run under XXXX. Each programming assignment worths 50 points (unless stated otherwise). Adhere to the following guidelines in order to prepare a professional, readable, and maintainable C program.

The points that you receive on a programming assignment is based on its correctness, efficiency of the algorithm you develop, and incorporating these requirements. As many as five points will be initially deducted for each of the following requirements that are not followed.

1. The first few lines of each programming assignment should contain the following:

    /*
    /* Your name */
    /* Program number (e.g., 1, 2, ...)
    /* Book assignment number (e.g., Chapter 1, #6)
    /* Due Date
    /* A brief description of your program
    */

The opening and closing symbols for comment lines should line up (see the above).
2. Use descriptive names for identifiers. Don’t make identifiers too cryptic or too long.

3. Functions headings should begin at the left margin. Separate functions from each other by one or two blank lines. Place comment lines for each function immediately before their headings. Comment lines should briefly describe the purpose and the input and output of each module. If a program is properly divided into small, single-minded modules, then a few comment lines at the beginning of each function will suffice and you no longer need to place comments in the body of the function. Do not include extraneous and unnecessary comment lines as they make a module less readable.

4. Include function type and formal arguments’ types in the function prototype. Function bodies can be placed before or after the main function. For consistency, place a function’s prototype before the main function, and put the function’s body after the main function. Also to achieve consistency, include a return statement for all functions.

5. Begin each C statement on a separate line.

6. Put a space before and after all binary symbols.

7. Put a space after “:”, “,” and “;” symbols.

8. The bodies of { } blocks, DO, FOR, IF, SWITCH, WHILE and similar statements should be indented from the corresponding header keywords. Indent consistently. Statements within these constructs should be aligned.

9. Do not use any features of the language that has not yet been covered in the class. Solve a programming problem only by using programming features that have been covered.

10. Make a source program statement fit on a single line. If the statement is very long, properly break it into two or more statements.

11. Follow your textbook’s guidelines very carefully. More specifically, follow the hints given under “Good Programming Practices.”
The goal of the above requirements is to make your source programs more readable. Take advantage of C’s free format. Include blank lines appropriately. Use parenthesis in complicated expressions to make them more readable even if the expression does not require parenthesis. Use your own judgment if I have left something out. Additional requirements may be given later.

2 Program Output Requirements

With each programming assignment you must turn in an output. A program’s output is as important as the source program itself. Thus you must make every effort to make it readable and intelligent. The output should minimally have the following properties:

1. The output of each program must begin on a new page. That is, the source program and output pages should each start on a new page.

2. Each program output should include one or more heading lines which would consist of programmer’s name, program number, program title, date, etc.

3. Cluttered output is as bad as an unreadable program. Make output of each program readable. Numerical figures should be formatted whenever possible. Include blank lines, good output messages, indentation, etc., to make your output readable and self-explanatory.

3 Input for Programming Assignments

Most programming assignments at the end of each chapter may include or suggest a set of input data to be used for testing a program. (There are some programming assignments that implicitly tell what input data can be used for testing purposes.) Your program should work with any valid input data. However, when preparing the final version of your program and its output, use the input data that is given to you. (If there is no input data given in the book, I will provide the data; in that case everyone should use that same data.) If you are not quite sure what data to use (or if the assignment has several sets of input), ask instructor for clarifications.
4 General Requirements

Here are some general requirements:

- A program must be turn in on the due date at the beginning of the class time. Five points will be taken off for each day a programming assignment is late (including the due day if the assignment is not turned in at the beginning of the class time).

- Do not include any hand-written comments on your source program or its output. The source program and its output should be self-explanatory and self-documented.

- Turn in your programming assignments in a folder with inside pockets. A folder protects your program against tear and damages, keeps it private, makes it easier to turn in multiple programs, and makes it easier for the instructor to grade and return back to you. Put your name on the folder.

- You can use any computer system as long as your programs are written in ANSI C. Computer accounts for XXXX, a DEC ALPHA machine running under DIGITAL UNIX, will be provided. Learn how to use the editor, compiler, and printing procedures properly. When you print your assignments, do not include other information (e.g., the operating systems prompts) in the printout. It is best if you include your source program and its output in separate files and then printing each file separately.

- Detach the perforated edges from the printouts. These edges should not include any code.

- Keep your graded programs until the end of the semester.

5 Grading Criteria

Initially, a source program and its output will each worth equal points. Later in the semester, most of a program’s grade is based on a correct output only. Towards the end of the semester, a program will earn points only if it produces a correct output. (In other words, you will not receive credit for doing a program that is incorrect and produces no usable output.)
6 Preparing an Executable Object Code

If you are planning to use a PC, turn in your object code on a diskette. The diskette should include the source code as well as the object code. If the object code is not available on the diskette, the program will not be graded.

If you are planning to use XXXX, you should follow the following minimal steps to compile and run your programs under XXXX. Note that XXXX runs under DIGITAL UNIX. The prompt symbol on most UNIX systems is %.

6.1 Login

Use telnet to connect to XXXX. Once connected, you will see the following prompt:

```
login:
Password:
```

You need to supply your login account and password in order to login. Once logged in for the first time, you may wish to change your password:

```
% yppasswd

Old password:
New password:
Retype new password:
```

Provide appropriate responses to change your password.

6.2 Logout

To logout, simply enter the following command:

```
% logout
```
6.3 Creating a File

To compile a C program you must create a file and type in your source code. There may be several editors available. One that is initially a little difficult to learn but is extremely powerful is called \texttt{vi}. Another simpler editor is called \texttt{pico}. Each C source program must have an \texttt{.c} extension. To create a file called \texttt{foo.c}, enter the following:

\texttt{% pico foo.c}

Learn the basics of \texttt{pico} (or \texttt{vi} if you choose to use \texttt{vi}) to create and modify a file.

6.4 Compiling a Source File

To compile your source program (\texttt{foo.c}), enter:

\texttt{% cc -o foo.exe foo.c}

The above command compiles file \texttt{foo.c} and if the compilation is successful, it will create an object file called \texttt{foo.exe}. Note that you can name the object file anything you want but it is best if it has a name that is closely related to its corresponding source file. (Note that it is possible to leave out the \texttt{-o foo} option; in that case the object code will be automatically named \texttt{a.out}.)

6.5 Running Your Object Code

To run an object code, simply enter its name:

\texttt{% foo.exe}

The object code will run; by default it expects the input to come from the standard input (stdin) device (i.e., the keyboard) and it will send output to the standard output (stdout) device (i.e., the screen monitor).
6.6 Redirecting Input and Output

In many instances, it is more convenient to place the input data in a file (e.g., foo.dat) and redirect the stdin device to read the data from that file:

```
% foo.exe < foo.dat
```

Similarly, in many instances, it is more convenient to redirect the stdout to place the output onto file, e.g., foo.out:

```
% foo.exe > foo.out
```

Of course you can redirect both stdin and stdout devices:

```
% foo.exe < foo.dat > foo.out
```

The above command directs your object code to read data from foo.dat and place the output onto foo.out.

6.7 Providing Instructor an Executable Code

In the home directory of my XXXX account (called /u/saiedian), I will create a number of directories called P01, P02, P03, ... Once you have finalized a programming assignment, you will need to copy its executable code in the corresponding directory (i.e., the executable object code for program 1 should be placed in P01). To avoid name conflicts, use your lastname (in lowercase) for your object code:

```
% cp foo.exe /u/saiedian/P01/your-lastname
% chmod a+x /u/saiedian/P01/your-lastname
```

The first command above will copy a file called foo.exe to the directory /u/saiedian/P01. The copied file will be called your-lastname. The second command will change the “mode” so that I can execute it.
6.8 Other Useful Commands

The following UNIX commands are quite useful; please learn how to use them effectively:

- **lpr**: for printing purposes
- **mkdir**: to make a directory
- **cd**: to change directory
- **cp**: to copy a file or directory
- **mv**: to move or rename a file or directory
- **Mail**: to read or send e-mail messages

The last command, **Mail**, is especially helpful as you may get many notices, comments, etc. regarding the class via e-mail.