Assignment Expectations

Read the assignment statement carefully. If your assignment does not meet the requirements of the assignment statement, it will lose value.

Read and use any provided source code. Often there will be sample programs related to the assignment, and a “template” program on which you can build your solution. It is always to your advantage to read and understand this code. You will likely need to copy it to a directory of your own to compile and execute it, although in some cases the executable version of a program may also be provided. If there is a template program provided, you may use it as the basis for your solution. Obviously there will be modifications required. The template is provided so you don’t have to spend time on writing code for things that aren’t central to what you’re learning in the assignment.

Make certain your results match the instructor’s results. For each assignment, an executable version of the instructor’s solution will be provided, as well as some sample input data. Run the instructor’s solution on the sample input data. Then run your solution with the sample input data. The results should be identical. Be aware, though, that the provided sample input data is not necessarily comprehensive; your program will likely be tested with additional (different) input data.

Identify yourself and the purpose of the code. The very first thing in each source file you produce should be a comment or two identifying you as the author of the code and the purpose of the code in the file.

Provide a “makefile” to build your program. A “makefile” identifies dependencies among the files in your program and gives explicit rules needed to build the object and executable versions of programs. Your submissions will be evaluated by building the executable from source code only, and this building is facilitated by providing a makefile.

Make certain your program compiles cleanly and uses only acceptable features. Programs are normally expected to be written in a standard C dialect (more notes below), and will be compiled using the “gcc” compiler. You should likely use the “-Wall” option to be warned of potential problems. For some assignments, you may also be allowed to use C++, Java, or Python. If this is the case, then your solution must be capable of being prepared and executed using the corresponding language system available on the loki.ist.unomaha.edu system.

Do not use “contract developers” to write your solution. There are organizations that continually monitor known sites (like freelancer) and report appearance of assignments appearing there to the instructor.

Source program lines should be limited to a reasonable length. Although there is no logical limit on the length of source program lines for many languages, 80 characters is the default width normally associated with a terminal window, and longer lines are more
difficult to read without making the terminal window wider, which consumes more display “real estate.”

**Comments** in C programs should be written using `/*` before the comment and `*/` after the comment. You should use comments to explain what you (and your program) are attempting to accomplish. Not only will these assist the instructor in helping you with your program, it will help you remember what you were trying to do. And don’t wait until the program is complete before writing comments; include them as you write the code. Think of your program as a literate effort.

**Variable declarations** in C programs must be placed in a group before any executable statement in a block (that is, a group of statements enclosed in braces). Do not “commingle” variable declarations with executable statements; it makes the code more difficult to read. If you find your declarations are getting far away from where they’re used, then the block or function to which they belong is too large.

**I/O facilities** not provided by the standard C library must not be used. Use only the functions provided by the standard C library (e.g. `fopen`, `printf`, `scanf`, etc.) Also do not use the C library I/O functions if you are directed to use only system calls for I/O (which will be the case with the first programming assignment).

**Namespaces** are not to be used in C programs.

“**Magic**” numbers are suspect in all languages, and are not to be used. A magic number is any constant value that appears “magically” without any explanation or rationale for its use. One way to avoid such transgressions is to use the preprocessor (in C/C++) or initialized `static final` declarations in Java to give names to constants used in the program. For example, “`#define MAXNAMELEN 37`” and a declaration like “`char name[MAXNAMELEN];`” is better than “`char name[37];`”. It also allows you to use the symbolic name for loop limits, and simplifies changing the value of the constant if necessary.

**Dynamic allocation** of arrays and other objects in C and C++ should be avoided unless the number of elements in an array or linked structure is not explicitly specified in the problem statement. For example, if the problem statement says the maximum size of an input line is 100 characters, it is better to declare an array of characters with a size of 100 to hold an input line rather than dynamically allocating the array. (Of course, the 100 would appropriately be defined as something like `MAXLINELEN` instead of using 100; see the previous comment about magic numbers.) Dynamic allocation, when needed, should be accomplished using only the standard C library functions (e.g. `malloc`, `calloc`, `realloc`, `free`) or similar facilities in other languages, unless other instructions are given in the assignment.

**Read data from the specified source.** If input data is to be read, make certain your program reads it from the source specified in the problem statement. Common sources of data are the command line (i.e. the `argc` and `argv` arguments to the `main` function), the standard input (the keyboard or a file if the standard input is redirected), or a named file (which your program will need to explicitly open for input). Programs are routinely tested by redirecting the standard input to a file.
Avoid being “chatty.” Your program’s output should be virtually identical to that shown in the assignment statement and produced by the instructor’s solution. It is certainly appropriate to use debugging output during the development of your program, but such output should be disabled or removed before submitting the final effort.

Check the value returned by system calls. Almost every system call (and library function) provides some mechanism to indicate errors. Unless you are absolutely certain that there is no possibility of an error, you should include code that tests for such error cases and deals with them. This can help avoid “fork bombs” (see below).

Avoid “fork bombs.” The `fork()` system call will be used in one or more programming assignments to create new processes. If successful, it returns twice – once in the process that executed `fork()` and once in the newly-created process. If you don’t understand exactly what you’re doing, it is possible to create programs that repeatedly create processes *ad nauseam*. Since there is a limit on the number of processes you may create, after a while you will not be able to execute any commands – you’ll have used your entire process quota. If this happens, use the “`ps`” command (probably in a separate login session) to show the processes you have running and the “`kill`” command to send an appropriate signal to the errant processes to terminate them.

Indentation should be used consistently to reflect the organization of your program. Your indentation style, where you place braces, whether you leave a space after `if` and before the left parenthesis, and other similar things is a matter of programming style and isn’t dictated. What is expected is that you adopt a consistent style and use it throughout your coding. Always assume a tab character will move the cursor to the next column numbered $8n + 1$ (e.g. column 1, 9, 17, …).

Use functions (or methods) to organize your code into units of manageable size. A function should not normally exceed a few hundred lines, but this isn’t an absolute rule. You should, however, use functions for each major (and sometimes minor) task.

Note problems with your solution. If you know your program has problems and does not implement the solution to the assignment, it helps to indicate you know this rather than have the instructor find that your solution doesn’t work and then wonder if you knew it or not. Such comments will indicate that you’ve identified the difficulty, even though you haven’t solved it. And that’s worth more than just submitting something without any indication of whether you know that it works or not.

Do use the test cases supplied by the instructor with the assignment statement. For each assignment, sample input and expected output will be provided. Make certain you execute your program using this sample data, and compare your results with those provided. Use redirection of the standard input to your program to verify that works, too.

Do not use only the test cases supplied by the instructor with the assignment statement. Prepare some additional test cases to exercise your solution. “Boundary cases” are often useful in identifying problems. For example, if a problem says an input line may be between 1 and 100 characters long, use a line of length 1 and a line of length 100 to test your solution. The instructor’s solution (in executable form) will usually be provided for your use so you can see what that program would do with a particular input data set. Use
it to determine what your solution should do (and the desired appearance of the output). Test data set construction is a significant part of program development.

**Do not waste time checking for input data violations** if you are told the input data will be syntactically and semantically correct. In most assignments you will be given a precise specification of the input and output data format. You will also likely be given a guarantee that the instructor’s test data will not violate the specifications. Although checking input data is necessary in programs designed for “human consumption,” the programs you write for this course need not have all of the usual checking included.

**Note extra credit options.** if any are provided, before beginning your solution. Do not necessarily attempt to implement the extra credit option before getting the basic solution to the problem working, but keep the extra credit option in mind as you develop your solution. This may facilitate adding the extra credit option after the basic solution is completed.

**Pay attention to the due date.** Although the instructor may sometimes be flexible about the date when you submit your work, there will always be some date after which your work will lose credit for late submission, or not be accepted at all. Similarly, don’t wait until a day or so before the assignment is due before beginning work. Some assignments may appear deceptively simple, but really require quite a bit of effort. Likewise, computer systems have been known to go down! Unless the down time is extended, such events will not likely be acceptable as excuses for the failure to submit your work on time.

**Test your solution on the system where it will be evaluated.** The system used to evaluate assignments is loki.ist.unomaha.edu, and you have (or will have) an account on that system. Although you may likely be able to do some development of your solution on other systems, make certain your solution compiles and runs successfully on loki.

**Check the on-line problem statement** on the instructor’s web pages occasionally. It may be updated to add or modify conditions on the problem, or to amplify solution expectations.

**Ask questions if you don’t understand something.** You are not expected to know everything, but you are expected to demonstrate a desire to learn. Also don’t hesitate to use additional resources (books, web sites, etc.) to obtain additional information and insight into the assignment.

**Do your own work.** Your code is to be your work only unless you are specifically given algorithms or source code to use, in which case you should make appropriate note of this in comments associated with the code.

**Do not waste time looking for solutions to the assignments on the web.** The assignments used for each offering of this course are unique, although they may obviously address the same concepts presented in previous course offerings. There are also externally-funded efforts to identify students who attempt to hire someone to write a solution. In the past, students doing this have been reported to the instructor, with the expected results.

_Last modified on January 12, 2016._