Computer Science 126 and 128 **Spring 1996**

Handout # 2: Programming Information

Assignments will be described assuming that students are using the SPARCstations in CS 101. Most students will find it easiest to use these machines initially, but many students may find other machines (say, those in the colleges) more convenient later in the semester. Students may use any computer at their disposal to do programming assignments, though electronic submission of programs and access to course materials will require network access to `phoenix`. Sometimes programs written on other computers may not run properly on the SPARCstations in the lab. *It is the students' responsibility to take care of compatibility problems.* Before submitting programs not written on the SPARCstations, compile and run them at least once on a SPARCstation.

Program submission

Most weeks, programs have to be submitted electronically by **10pm** on Wednesdays. To submit a program, type

```
/u/cs126/submit <prognumber> <filenames>
```

or

```
/u/cs128/submit <prognumber> <filenames>
```

as appropriate on any machine for which `phoenix` is mounted. This will make a copy of the files for the TAs, identified with your login ID. You need not submit all files for a programming assignment with a single `submit` command. You can execute `submit` any number of times; older submissions are replaced with newer ones. The TAs won't look at the programs until after the deadline. Be sure to keep a copy of the files you have submitted in your own directory as a backup.

Hardcopy submission

Also print a hardcopy of your program and prepare *documentation* for it. Hand it in during lecture on Thursday in the appropriate box (126 or 128). Late assignments (both COS 126 and COS 128 assignments) can be placed in the **COS 126 Hand-in slot** in the second floor lobby of the CS building. **Make sure you use the official cover sheet for the hardcopy you submit**. (These are available in the lab, Room 101 and are color coded for 126 versus 128.) Make sure that your work is legible, neat, and multiple pages are stapled together.

The purpose of the hardcopy submission is to help the grader determine how well the submitted code solves the assigned problem. Your grade will depend upon both the electronically submitted code and the level of documentation in your hardcopy. (See below.)

What is documentation and why is it required?

A computer program (especially one written late at night on a deadline) can sometimes be incomprehensible. In fact, it could be incomprehensible even to the person who wrote it (wait until this happens to you!).

Your documentation should consist of a brief (1-2 page) narrative that explains what the various parts of the program do, and how. The narrative should be typed or typeset; although you may choose to add handwritten annotations or diagrams on top of it. Preparing this documentation needn't take a lot of time, but should be done with care.

In addition, the hardcopy of your program should contain plentiful *comments* and be properly indented for easy reading.

Writing good documentation is an acquired skill, whose usefulness you will appreciate more and more as you write and maintain large programs. At the same time, do not get carried away while putting comments in the program. For example, the following comment is not very useful.

```
i = i+1; /* increment the variable i by one */
```

Help from the laboratory assistants

Undergraduate laboratory assistants will be in room 101 evenings to help you. They are there to help if your program doesn't compile or run properly (i.e., has *bugs*). They will also gladly answer questions about C and how they relate to the assignment. However, the lab assistants have been instructed **not** to give you substantial hints on the assignment itself. Figuring out the assignment is **your job.** To help you in that, Friday classes will be devoted to discussing the assignment that is due the following Wednesday. Class instructors are also available during their office hours (or by appointment) to further discuss the assignment with you.

In general, bear in mind that successful programmers do not "compose" a C program while sitting in front of the computer. Instead, they design it carefully away from the computer; making crucial design choices and hand-writing important pieces of code before they have even started up that emacs window. Following this rule will **save you hours of frustration in the lab.**

The current schedule for laboratory assistants in room 101 is:

Sunday afternoons (check exact time on posted schedule)

Sundays 8pm-midnight

Mondays 8pm-midnight

Tuesdays 8pm-midnight

Wednesdays 7pm-midnight

Thursdays 8pm-midnight

How your assignment will be graded

The following questions will be considered while evaluating submitted programs.

1. *Does the program solve the problem?* The graders will first try to compile and run your submitted program. If the program doesn't run, you will lose a big portion of your grade. However, *if the grader sees from your documentation that you basically had the correct idea, you will still get some points.*

   Conversely, if your program runs fine, but your documentation is bad, *you will lose points.*

   Sometimes it may be appropriate to say how your program might be improved, though that is no substitute for actually implementing the improvement.

2. *Is the program well-structured?* Part of the art of good programming is the process of breaking a problem into manageable parts and providing reasonable interfaces among the various pieces. There are a great many ways to solve a problem with a C program; you should evaluate alternatives and make reasonable choices as you put your program together.

3. *Did the student understand the problem?* The assignments generally explain a problem in detail, and give specific instructions/hints about solving it. Class discussion will provide further hints. Sometimes it may be possible to use these hints (plus some "guesswork") to produce working code, without understanding the problem.

   If your code or documentation reveals that you did not fully understand the problem, you will lose a big portion of your points. Before starting to write code, make sure you understand the basic ideas behind the problem and method of solution and *why* various steps are taken.

   Finally, we remind you that **there is a strict policy on collaboration**, which you should read and understand (see *Handout 1*). If it is not clear enough, **ask us**.