All About Homework

The homework will consist of both programming exercises and written questions. This handout describes some general thoughts and techniques for doing homework, as well as what is required, how to submit it, how late homeworks are handled, and other administrative matters.

Turning In Homework

All homework is due at noon on the due date, unless noted otherwise on the assignment. These will be graded and returned to you as quickly as possible; we'll try for three class periods, but can't guarantee it .

For written homework, you must turn in an ASCII, a PostScript, or a PDF version of your answers (you can use any text processor you like to generate these). If you submit PostScript, please be sure the file will print on our department printers (use *ghostscript* or *gs* to check this; if it displays the file properly, the file should print correctly). If your file is a postscript file, choose a name that ends in ".ps". If it is an ASCII file, please choose a name that ends in ".txt". If your file is a PDF file, choose a name that ends in ".pdf".

For programs, turn in the source code and any related information (such as man pages and README files). Be sure that we can recompile it *without errors* by typing "make". You are free to use any programming language that is available on the CSIF and that the ECS 153 graders can get to. C, C++ or assembly is acceptable. Any of the languages in the programming languages class is acceptable (assuming compilers and interpreters are available in the CSIF), and if you can write your programs in such a way that *troff*(1) or *latex*(1) can execute them, that's fine too. (Yes, someone once wrote a BASIC interpreter as a set of *troff* macros. It was very slow, but it worked.) But use lots of comments!

Please turn in both your written exercises and programs electronically. Suppose you want to turn in the files *answers.ps* and *prog.c* for homework 3. To do this, go to the directory containing both and type

```
handin cs153r hw3 answers.ps prog.c
```

This program will submit your files to the ECS 153 grader. (A manual page for the *handin* program is attached.) You have to do this from the CSIF; *handin* does not work from other systems.

Doing Written Exercises

When you are asked to analyze something, or explain something, please be complete, and ***show your work*** (including any commands you give, and their output, to show how you did the problem). Otherwise, even if you get the right answer, you will get *ZERO* (that's **0**, **zip**, ***nada***, **nothing**) points. Think your answer through and do a rough draft. Students (and professionals, actually) often overlook this, but it is *vital*. Write clearly and cogently. If the question asks for an opinion, state your opinion clearly, justify it, and don't ramble. Answers that start, "My opinion is yes …" and conclude with " … on the other hand it could equally well be no" won't get much credit.

Each homework will have one problem that will require you to ask a question. Your question can come from something in class, in the notes, something you read or heard about in the media, or just something that strikes your fancy. Your question must be specific, in the sense that it challenges an assumption or raises issues of analysis for a problem. Here are some examples:

1. A firewall is supposed to allow only some messages through from the outside to the inside. The assumption is that the firewall's software works right. How do they tell when the code that does this filtering is working right?
2. You said in class that using *strcpy* is bad because it doesn't check bounds, and we should use *strncpy*. But what happens if we give *strncpy* a negative length?
3. Is "ethical hacking" really ethical?

You can have fun with this. We'll put the best questions of each assignment on a web page. We may answer some (it depends on how complex the answers are, and what we plan to cover in class).

The reason for this problem lies in the nature of security. Invariably, the assumptions a security mechanism or policy makes are the points at which that mechanism or policy are most vulnerable. Why? Assumptions, designs, and implementations invariably make certain assumptions about the environment. One of the goals of this class is to teach you to question these assumptions.

Doing Programming Exercises

Please do not leave assignments for the last minute. The assignments are non-trivial and will require significant design time before you start programming and debugging. When we decide on the due dates, we assume you will spend significant amounts of time on design as well as coding and debugging. If you choose not to do this, you will have difficulty finishing the assignments on time.

Please take the time to design your program carefully. More programming problems arise from improper design than anything else, and the few hours you spend on design will be amply repaid by shorter coding and debugging phases. So please think the design and interfaces through, and—as always—try to find the simplest way to do the assignment (within the limits given in the assignment, of course)!

We do not mind being asked for help; indeed, we welcome it because it helps us know what students are finding difficult or confusing, and sometimes a few words about the problem in class will clarify the assignment immensely. We **do** mind being asked for help before you have tried to think the problem through. The classic objectionable question (this really happened) occurred on a homework assignment in which the class was given a buggy program. The assignment said the program did not work, and the homework was to debug it and make it work. That particular class period discussed how to deal with bugs, and gave tips and techniques on how to debug programs. Within 10 minutes of the end of the class during which the assignment was given out, the instructor got this request for help: "The program doesn't run. What do I do now?"

So, before asking for help, please be sure that you have:

- spent a significant amount of time on the design of your solution;
- used a debugger if the problem is a programming bug;
- read all relevant handouts, and news articles (because your question may be answered there); and
- tried everything you could think of to solve the problem.

When you come to us, or send us a note, asking for help, please show us whatever you have done to solve the problem, because the first question we will ask you is "What have you tried?" This isn't because we think you're wasting our time. It's because understanding how you have tried to solve the problem will help us figure out exactly what your difficulty is and what we can do to help you. Remember, we will do everything we can to avoid solving the problem for you. When we give you help, our goal is to help *you* solve the problem yourself.

What We Look For In Programming Exercises

When we grade your homework, we look for simplicity, clarity, elegance, and documentation. Here's a rough weighting of the various factors that go into the grade of each program:

Correctness	60%
Commenting, ease of reading	20%
Clean, readable output	10%
Documentation (README, man page, *etc*.)	10%

If a program does not compile (or assemble), the maximum you can get is 30% of the value of the program. So check your programs before you submit them!

Late Homework

You can turn in your homework up to one class period late (unless the assignment says otherwise). If you turn it in late, we will grade it normally, and then deduct 20% as a late penalty. Requests for exceptions will be handled on a case-by-case basis; please do feel free to ask!

Grade Appeals

If you feel that there is an error in grading, please come see me or the TA and we'll look over it (and possibly talk with you about it). However, don't dally; any such request must be made within one week of when the grades were made available. After that, we won't change your grade.

Handin Program

NAME

handin – file submission program

SYNOPSIS

/usr/pkg/bin/handin *touser* [*subdirectory* [*files* ...]]

DESCRIPTION

handin provides a secure means of submitting files to another user, recounting what has already been submitted, and listing what subdirectories exist for containing submissions.

USAGE

Submitting files

With *touser*, *subdirectory* and *files* all specified, each file is copied to *~touser*/handin/*subdirectory*/*fromuser*, named with the original file's *basename*(1), and made owned by *touser*. The directory *fromuser* is made if it doesn't already exist and is named after the invoking user. Each file specified should have a *basename*(1) unique among any files already submitted by that user to *subdirectory*, unless overwriting is desired.

Recounting submissions

Without *files* specified, information on previous submissions by the user to the specified *subdirectory* is shown.

Listing existing *subdirectories*

Run with only *touser* specified, **handin** just lists the existing subdirectories (regardless of accessability).

EXAMPLES

The following examples illustrate the use as a homework submission facility to the pseudo-user ``cs101'' created for this purpose:

```
example1% handin cs101
Existing subdirectories (comments in parentheses):
Asn1    (Due Mar 18)
Asn2    (Due Mar 25)
example2% handin cs101 Asn1 part1 part2
Submitting part1... ok
Submitting part2... ok
example3% handin cs101 Asn1
The following input files have been received:
Thu Mar 17 14:50:49 1994        1599 bytes      part1
Thu Mar 17 14:50:49 1994        3412 bytes      part2
```

SEE ALSO

rcvhandin(8)

DIAGNOSTICS

handin itself provides only a little of the diagnostic information that's given and returns the number of errors encountered as its exit status. Any other information comes from *rcvhandin*(8).

Skipping *file*: file non-existant or irregular
The named file didn't exist or was probably a directory. The user should check to make sure that the file they specified was indeed the file they intended to submit.

Skipping *file*: file not readable
The named file was not readable by the user.

Submitting *file*... failed [: *reason*]
The named file was not successfully submitted. If at all possible a reason is provided by *rcvhandin*(8).

Submitting *file*... ok
The named file was successfully submitted.

NOTES

handin is really just a front-end to the *rcvhandin*(8) program. The primary function of **handin** is to open the named *files* with the effective user ID of the invoking user and pass on their contents to the *rcvhandin*(8) program having the effective user ID of *touser*. This design provides a simple and portable means for implementing a file submission facility in even a non-homogeneous, network-file-system environment.

AUTHOR

Lou Langholtz, Department of Computer Science, University of Utah, 1994