CSCI-E26 Course Outline Introduction

Csci-e26 teaches C and Unix programming. C and Unix are used for all sorts of programming - scientific, database, financial, text processing, operating systems. In this course, we shall focus on CGI programming as an on-going example.

The web browser provides a user interface. The C and Unix programs serve as engines to process, store, and retrieve data. We shall see how to build these engines in C and Unix, and we shall see how to use C and Unix programs to connect these engines to web pages. Class 1. The Big Idea

Computer programs have not really changed much in decades. The sequence of events is always something like: 1. The program asks user for input 2. The user types in something 3. The program receives the input from the user 4. The program does some processing, maybe looking up
information in external files, maybe storing data
somewhere, maybe doing some calculations. 5. The program sends results back to the user Everything from a hand-held calculator or video game to a weather analysis program works pretty much along these lines. The basic operation of programs does not change, but the languages and tools for building the programs and the devices used for input and output do change. Once, paper tape and punch cards served as input and output and the programming was done in FORTRAN and COBOL. Later, teletypes were used for input and output, and new languages appeared - BASIC, Algol, Lisp, APL. Later yet, video display tubes with keyboards appeared and even more languages appeared. Video terminals that connected users to central computers began to be replaced by personal computers. On a personal computer, entering user input, processing data, and displaying results were all done on one machine.

Now the world wide web has appeared. The web revives the model of separating the user interface from the data processing system. The web pages are fancier than traditional video terminal screens, but they serve exactly the same purpose. Is this a step backwards? Not really. There is an important difference between the video terminals and the web pages. Video terminals provide access to a single machine using a cable. Web pages provide access to any number of machines using network connections.

With the web, a computer user can send data to computers anywhere on the network and receive the results back. The web has made popular the concept of a *software engine*. People speak of the search engines they like, or engines that generate driving directions between any two addresses in the country. The web has made it possible to think of software the way people think of automobiles - the sedan model with the V8 engine, or the minivan model with the V8 engine, or the minivan model with the v6 engine.

What does this have to do with C and Unix? Simple. C is a language used to build software components - little motors that perform specific tasks. Unix is a programming language designed to combine software components into programs. Some people have complained that C and Unix programs have unfriendly user interfaces. That is like saying that a Chevy straight 6 motor has a crummy dashboard. Unix makes it easy to add any user interface to a C program, including a web interface. Just as the straight 6 engine doesn't care what color the steering wheel is, C and Unix programs don't care what the user interface looks like.

What is this course about, then?

This course will teach you how to program in the C programming language. We shall use C to build software components - special purpose programs with terse, unfriendly, command-line interfaces. The course will teach you how to use the Unix programming language to combine components to create useful applications with friendly user interfaces. Finally, the course will teach you how to create and attach web pages as user interfaces to components and applications.

The C and Unix programming parts of the course are useful even if you never use the web as a user interface. The web interface part is useful even if you decide to program in some other language or operating system.

Unix: Operating System or Programming Language?

In the preceding paragraphs, I have been speaking about the Unix *programming language*, but most people refer to Unix as an *operating system*. What gives? In this first class, we shall see that Unix is an operating system and that Unix is also a programming language.

An Extended Project

We discuss the plans for an extended project: a web site and tools for a database of MBTA commuter rail schedules. (Note for those not from the Boston area: The MBTA is the metropolitan Boston Transit Authority - the agency that runs public transportation)

The web site will include functions for searching, tabulating, and mail-merge. We sketch the general architecture of the system. We shall build the pieces in some of the homework assignments. Class 2. Software Tools

The web offers a user interface. Shell scripts act as dispatchers - collecting requests from the browser and invoking tools to do the work. Software tools do the work.

Consider the paper and pencil activity of registering for a course or two at the Extension School. The paper application is the user interface; the applicant puts information into data fields on the form. The form is submitted. Then, at 51 Brattle Street someone opens the envelope and sends the contents to different departments: registrar, computer services, financial services, student records...

We start with software components. A Unix software tool is a simple machine with input, output, storage, and code. The input can be attached to a file or a process. The output can be attached to a file or a process. The tool accepts command line arguments that affect its behavior.

What goes on *inside* of one of these tools? We look at the structure of a C program - functions, variables, input/output, control flow. Unix and the web work with text, so we focus on programs that process characters and strings.

Strings are simply arrays of characters. We study arrays.

All transactions on the Web are text-based. Most Unix programs operate on textual data. We write some programs to process the train schedule database: one to capitalize names, one to convert semicolons to tabs. We add these hand-crafted tools to our set, and we incorporate them into shell scripts. Class 3. String Processing: Lists of variable=value

A web browser accepts data and sends it across the 'net . The fields and their values are encoded as a set of var=val pairs. The shell uses var=val to to store session information. Our train schedule db db uses var=val to store info. This is a popular format. Let's see how to write programs that work with this model of information storage.

We look at data formatting programs.

We write code to parse a line like

```
TR=1806;dir=i;day=sa;TI=12:10;stn=hyde park;Line=attleboro
```

into separate strings and then examine and output those strings.

String operations are so central to Unix programming that the system comes with a lot of functions to assign, copy, compare strings in all sorts of ways.

Once we collect our data formatting tools, data analysis tools, and data cleaning tools, we have the pieces we need to create scripts to analyze, clean, format, process all sorts of textual data. Class 4. Building Interfaces: Shell Scripts and Web Pages

We have seen how to construct software engines to perform specific operations. We can use these components directly, typing in data or sending it command line arguments. It is now time to create user interfaces to these components.

We start with shell scripts that prompt the user for information and then passes that information to the engines. We use shell variables to store user responses. Sometimes we use variables to store output from the engines.

In the second half of the class, we step back one more level - across the Internet. We create a simple web page that uses a *form* to accept user input. When the user 'submits' the form, all that input is delivered to a program on the Unix machine. That program then passes that input to the engine. The output from the engine is then sent back to the user over the Internet.

The data input part of the program runs on the user's computer, while the data processing part runs on the web server. The questions we look at are: 1. How does one create a form? 2. How does the input get from the form to the server? 3. How does the output get from the server to the user? We are introduced to a software tool (an engine) that converts the user input values into shell variables. With it, we have a continuous path from browser screen to C program and back.

We work through several examples to get accustomed to this three-tier architecture. Class 5. Strings and Pointers

Strings are a central data type in C/Unix and Web programming. We look at them in more detail. How are they stored? How does one process them? How do they get passed to functions? We meet pointers. We use pointers to process strings. The idea is not too tricky, but the syntax takes some getting used to. We do lots of examples; pictures are provided. Class 6. Structs and Linked Lists

One of the functions provided by our web site is to tabulate data. For example, we can ask how many trains stop daily at a given station, or for a given station, we could ask the times of the trains that stop there or the frequency of trains per hour.

We map out an abstract filing system for this project. We then look at three implementations: parallel arrays, an array of structs, a linked list. Pointers figure prominently in our work. Class 7. Managing Complexity: Multi-file Programs, Storage Classes

As our programs start to grow, we need a way to keep the code simple and maintainable. C programs can be written in several files and linked together. We see how it's done.

Once you split a program into separate files, you need to think about how that division affects variables. We see how files can have local variables just as functions can have local variables. Class 8. File I/O and Command Line Arguments

Our FirstYear data access system processes data from a file. Many programs work with files of data. How can a C program read and write data from a file? We see how to open a file, get characters and strings from it, put characters and strings into a file, and how to close a file.

The 'user interface' for most Unix programs is the command line. It allows the user to specify file names, parameters, and options. To complete our study of writing software engines, we see how to process command line arguments.

We then go back to a web page and show how check boxes or radio buttons can be used to invoke command line options on the C program. Class 9. Focus on Shell Scripts: The Middle Tier

Having studied C in detail, we now turn our attention to the middle level - shell scripts. We have been using shell scripts to transfer data between the web browser and the software tools.

The shell is a powerful programming language in its own right, with variables, if..then, loops, functions. For some projects, it is a better tool than C.

We look at several examples to see how to use write shell scripts that do interesting, and sometimes complicated, things. Class 10. Shell Scripts Part II: Loops

We continue to learn about using the shell as a programming language. We focus on file and directory processing, using wildcards and loops. Class 11. Shell Scripts Part III: CGI in sh

The shell is a programming language, so one can use it as the processing part of a CGI program. It is particularly useful for indexing directories and building dynamic html. Class 12. Web Programming: Design Decisions

We also look at alternatives to C/Unix CGI programming. A web-based program spans three levels: web page, request handler, software tools. Where on these three levels do you store data? Where do you do processing? What are the options? What are the trade-offs?

Now that we have studied the details of all three levels, we consider design decisions for building complete applications. Class 13. Unix Text Tools: Regular Expressions

We have worked with data in text files, but there are other types and sources of data (images, sound), other sources (program output, devices, web services), and other processing tools (netpbm, sox, regular expressions). Today we explore some of these other sources, types, and associated Unix tools.