1 The course

The aim of this course is to develop the necessary systems programming skills in C and Linux as a foundation to tackle the design, implementation, and integration of a large software project working in small teams. The challenge for the course is to quickly get people up to speed so there is sufficient time to get into the details of a complex software design project. The first part of the course serves to develop design, programming and other systems skills such as source code management, testing, and debugging. The second part of the course is all about the project and team work. Good team work will lead to success. That’s the message.

The syllabus in a nutshell:

- Linux: shell, commands, shell programming, advanced material (sockets, process, signals, threads);
- C: structure, arrays, pointers, dynamic memory, files;
- Programming tools: gcc (GNU compiler), make (maintain groups of programs), gdb (GNU debugger), cvs (source code management), gprof (GNU execution profile);
- Software development methodology: design, implement, unit test/debug, integration, demo;
- Thematic programming assignments: building a search engine (crawler, indexer, query engine) - you can be google too!
- Group project: Embedded linux programming with robots/sensors - real-time system control.

The course includes weekly programming assignments for the first part of the course (approx. 7 weeks). The last part (approx. 3 weeks) is held over for project work. There are no lectures in the last part of the course but the projects are run with a formal design review, code review, and demo as well as periodic progress meetings where the team can brainstorm problems and come up with solutions. There will be a common project goal for all teams based on a “robotic treasure hunt” but students are free to develop their own ideas beyond this common goal - we want you to be entrepreneurial.

This is a new/revised course where most of content is new - there could be buggy. Caveat: we might have to make changes to the syllabus based on how people progress with the material. So you will have to cut us some slack and we will reciprocate.
Note, there is a significant amount of programming in this course requiring a significant time commitment on the part of the student. You will need to be well organized to complete all the programming assignments and project. It may not be all plain sailing but we hope it will be fun - you will certainly learn a set of new skills that will be very useful in the software industry. We will learn together.

2 Grading

There is no midterm or final exam in this class :) There is, however, a number of programming exercises, reading assignments, and a group project :( - well :) really. The grading for the course is as follows:

**15% - Class contribution.** There will be a discussion in class regarding the course reading material. Active involvement in that discussion will help toward the class contribution part of the grade.

**35% - Laboratory exercises.** There will weekly laboratory assignments for the first six or seven weeks. These are designed to help you learn the languages, tools, and the design skills you will need for your final project. These assignments are to be done individually unless otherwise stated in the assignment. Typically, assignments will be handed out on Friday and be completed the following Friday. X-hours will be used for working on programming assignments.

**50% - Team project.** The project is made up of a small team (three or four people) and requires strong collaboration and a problem solving mindset to get the job done. The instructor will put together the teams (to balance skill sets) with each member being responsible to deliver against a part of the overall system design, implementation, testing, and integration. The goal of this activity is to help you develop the confidence, skills, and habits necessary to write large computer programs while part of a multi-person team. You will become conversant in software engineering paradigms, and be exposed to various public-domain and open source tools that make the software development process easier. In addition, you will develop vital skills in self-directed learning, problem solving, and communication. The project will have a design and code review as well as the demo. A project report that captures the design and implementation will be submitted as part of the assessment. The project report will be written using a text editor, the LaTeX language, and Linux latex command.

See the course webpage for other details on the course. Please check the webpage frequently for updates. The webpage is [http://www.cs.dartmouth.edu/~campbell/cs23/](http://www.cs.dartmouth.edu/~campbell/cs23/)

Please also checkout the late submission policy on the webpage.

3 Lecture goals

We plan to learn the following in today’s lecture:

- Logging on to a Linux machine;
• Looking at the home directory and its files;
• Editing and compiling our first program in C;
• Taking a look at the compiler process and the code chain: from C to machine code;
• Logging out; and
• Some housekeeping business.

In the first four lectures we will cover Linux, the shell, and shell programming. We may touch on a few advance topics (process, sockets, threads) that we will come across while programming - if we have time. This is not meant to be a detailed presentation of the Linux OS and its programming tools. No, we’d need a complete course to cover that material. We need to know enough about Linux and its tools to be able to navigate our way around the system, write some basic shell scripts, and use its programming tools.

It is important that you use these notes as a starting point and like any budding hacker you need to do some experimenting and get online and read up on the details. You need to go on the web and find information if you have gaps in your knowledge then come see the instructor for help. There are a number of references dotted through the notes and at the end of the notes to get more detailed information.

In this lecture, we plan to cover what will become a familiar process to you: logging on to a Linux machine, writing a C program, running it, and logging off. We’ll also delve into the compilation process and discuss the program from its C origins to an executable running on a microprocessor.

Caveat: Please take note that lecture notes will not always be detailed. You will need to augment these notes with your own comments and by read the references and reading assignments so you can dive deeper into the topic.

OK. Let’s get started.

4 Logging into a remote machine using Secure SHell (ssh)

First, let’s log on to one of the Linux machines in the Sudikoff Lab 001. You can take a tour of the lab and its Linux machines: [http://www.cs.dartmouth.edu/~wbc/suditour/001/index.html] Sudikoff Lab 101 tour.

I’ll login in from my Mac using the Secure SHell (ssh) linux command line to remotely log into “moose” (Linux machine) or to give it its full IP name - moose.cs.dartmouth.edu. The ssh command replaces telnet (remote communications with another computer) and rlogin (remote login) because they lack security. The ssh utility is mainly used these days because your password gets encrypted when it’s transmitted over the network, not sent in clear text. I’ll log on from an xterm (X Window terminal - we will discuss this later) initially and then later using the Mac OSX terminal application - I’m a Mac person, which means Unix under the hood, sweet!

```
[atc@Macintosh-7 atc]$ ssh -Y -l campbell moose.cs.dartmouth.edu
```
campbell@moose.cs.dartmouth.edu’s password:
SSH provides a secure way for a user to access a remote computer and run commands on that computer. There are a large number of Linux commands, 100s. We will learn a small set of commands that will be useful for the course and project. Commands are entered directly at the console or remote terminal (terminal, xterm, puTTY - we will talk about this later today). Each Linux command has a short abbreviated command (e.g., Secure SHell (ssh)) and its associated syntax that typically includes various arguments, and options/switches; typically, these options (or switches as they are also known) are single letter preceded by a hyphen (e.g., -X). For example,

ssh [options] hostname [command]

The switch “-Y” enabled trusted X11 forwarding from the remote machine. The other switch “-l” informs the ssh that the username of the user logging in is campbell. SSH established a secure channel and uses public-key cryptography to authenticate the remote computer and the user.

5 Getting Information using the online manual (man)

If you want to the detailed syntax of a Linux command you can either use the manual command followed by the command:

[atc@Macintosh-7 atc]$ man ssh

NAME
ssh -- OpenSSH SSH client (remote login program)

SYNOPSIS

DESCRIPTION
ssh (SSH client) is a program for logging into a remote machine and for executing commands on a remote machine. It is intended to replace rlogin and rsh, and provide secure encrypted communications between two untrusted hosts over an insecure network. X11 connections and arbitrary TCP ports can also be forwarded over the secure channel.

This is just a snippet of the man ssh output (man is short for manual). The manual output includes all the nitty gritty details on options and about the command. For most commands you can use the common option “-help” (two hypens) to get a brief breakdown of the command and its switches. This doesn’t work for all commands but in that case the use of -help is processed as an invalid entry by the shell and lists of the options - so same results.
BTW, the shell is a very nifty program and acts as the command interpreter for Linux.

The online documentation for Linux (commonly called the Linux manuals) are divided into a number of sections that you can specify in the command line for “man”. These include:

- `man1` standard utility programs
- `man2` the operating system calls
- `man3` standard library routines
- `man4` special files and hardware support
- `man5` significant file and kernel structure definitions
- `man6` the standard games
- `man7` miscellaneous manual entries
- `man8` system administration
- `man1` any local enhancements to the system
- `man2` any networking details

The manuals locate information by searching in the order:

1, n, l, 6, 8, 2, 3, 4, 5 and 7.

There are different levels of information associated with a searched item depending on its context. For example, there is information on sockets can be found in system calls (man2) and in the networking section (man2). Similarly, there is manual information on the wait command as a standard utility (aka command) and system call. Selecting a section depends on what you are looking for. You can use “man -k keyword” to search through the manual pages for matches on a keyword.

```
[campbell@moose ~]$ man n socket
...
[campbell@moose ~]$ man 1 wait
...
[campbell@moose ~]$ man 2 wait
```

6 Your home directory and its files

Each user has a home directory. After you have logged in using ssh you are in your home directory.

We can look at our home directory “path” using the `pwd` (print working directory) command:

```
[campbell@moose ~]$ pwd
/net/nusers/campbell
```

Let’s take a look at the contents of my home directory (using the `-l` switch which means long format):

```
[campbell@moose ~]$ ls -l
total 434
drwx------ 2 campbell faculty 48 Dec 22 15:29 bin
```
The Linux model for files is simply a linear stream of bytes. Files can be plain files, directories or special files (we will talk about this in the next lecture). We can see that each file has file permission and other data associated with it; for example:

```
drwxr-xr-x 4 campbell ug 728 Oct 26 2006 public_html
```

You can traverse directory trees assuming you have the appropriate permission.

Linux supports a number of shells (command line interpreters). If we use the `echo` command we can look at the environment variable that tells us which shell is running. For this course we will use the `bash` (Born Again SHell).

Again, the shell is command line interpreter.

```
[campbell@moose ~]$ echo $SHELL
/bin/bash
```

More on the bash shell later.

### 7 BTW, if all we wanted to do was this, here is quicker way

SSH is a cool command because it allows you to execute remotes commands if you don’t want to login to the remote computer and just wanted to execute a Linux command such as “ls -l” then do this

```
[atc@Macintosh-7 atc]$ ssh -Y -l campbell moose.cs.dartmouth.edu ls -l
campbell@moose.cs.dartmouth.edu’s password:
total 434
-rw------- 1 campbell faculty 435438 Dec 14 2006 Sent
-rw------- 1 campbell faculty 1017 Mar 22 2007 Sent Messages
drwx------- 2 campbell faculty 48 Dec 22 15:29 bin
drwx------- 2 campbell faculty 48 Dec 22 15:30 lib
drwx------- 3 campbell faculty 1336 Dec 22 19:22 mail
drwx------- 3 campbell faculty 104 Nov 6 12:01 papers
drwxr-xr-x 4 campbell ug 728 Oct 26 2006 public_html
drwx------- 3 campbell faculty 72 Dec 11 15:14 teaching
```

If you use ssh and a command is specified, it is executed on the remote host instead of a login shell; its the equivalent to logging in, executing ls -l, and then logging out - all on one command line - isn’t that cool.
Another useful command for copying files between machines is the `scp` - secure copy (remote file copy program).

```
/atc@Macintosh-7 l1]$ pwd
/Users/atc/teaching/cs23/notes/l1

[atc@Macintosh-7 l1]$ scp lecture.html campbell@moose.cs.dartmouth.edu:public_html/cs23/.
campbell@moose.cs.dartmouth.edu’s password:
lecture.html 100% 1 0.0KB/s 00:00
```

That file is a little light on content!

You can install lots of fancy programs to provide nice GUIs that uses these basic commands such as puTTY (for Window machines), or MacSSHPPC, NiftyTelnet, Fugu (my favorite) but it’s nice to see what’s under the hood - which, in essence, is the command line.

OK, let’s log back in using ssh but from a standard terminal (using the OSX Terminal application - it looks very much like an xterm, but it’s different). No need for the -Y switch because this is not X11 now.

```
[atc@Macintosh-7 atc]$ ssh -l campbell moose.cs.dartmouth.edu
campbell@moose.cs.dartmouth.edu’s password:
Last login: Sat Dec 22 19:45:59 2007 from c-75-69-130-98.hsd1.nh.comcast.net
```

8 First C program

First, we need to write the source code by opening and editing a file.

We would like you to use a text editor to write code and documentation - the emacs editor is a good editor for developing software and supports context sensitive editing for C, shell programming, and latex (you will be using latex to write your report). If you know “vi” that is fine too. If you are not familiar with emacs checkout this short [http://www2.lib.uchicago.edu/keith/tcl-course/emacs-tutorial.html] Emacs Tutorial. If you open the emacs editor (as below) and look under “help” you will find an bundled tutorial.

In this course we stick to the command line as much as possible. We are not using an IDE (Integrated Development Environment) such as the Eclipse IDE in the first part of the course. Later, we may think about using Eclipse during the project phase but for now, no IDEs, we want to run all the tools from the command line interface. Once you know what’s under the hood then we can hide the details and more up the abstraction tool chain to an IDE (note, you can’t become a good architect until you know what bricks and mortar are and how they get used, right? Same thing here.)

So let’s open the emacs editor and write the code:

```
[campbell@moose ~]$ emacs hello.c
```

Here is your first program:
OK, now we have written our first C program let's compile and execute it. We will use the GNU (aka GNU is Not Unix) tool chain and its gcc compiler. We will make better use of the gcc compiler in the course but for know the compiler produces an “executable” called a.out (meaning an assembler output file). a.out is the default output executable created by gcc when no output name is specified in the command line, as is the case below.

```
$ gcc hello.c
$ ./a.out
Welcome C23, ready to hack?
```

C is a “not so high level language” and can be coded to take advantage of the microprocessor it executes on in order to write high performance code (e.g., C’s register language support). In this Java and OO world we live in we can easily forget about the underlying processor. Java abstracts you away from the processor and the OS which is good and bad.
So let’s take a quick peak at the compilation process - the code chain. There are three phases to producing the executable: the compiler (man gcc), the assembler (man as), and the linker (man ld). You can use the “man gcc” command or type “info gcc” (GNU info system) for all the details on the compiler. The GNU gcc compiler supports a large number of microprocessor families including the Intel “x86” family which is commonly used on Linux machines.

9.1 Code chain: C to assembler code

Let’s play with the assembler by getting gcc to produce assembly code from the hello.c example. We will look at the assembler code. Then get the assembler program to take the assembler code and produce an executable object code - also called machine code, the binary code that the processor executes. The microprocessor doesn’t execute, assembler, C, shell script, Java - only machine code. So we better translate the program into the execution language so the “machine” can understand. We use the “-S” option when compiling hello.c This gets gcc to translate the C code found in hello.c into assembler code in hello.s.

[campbell@moose ~]$ gcc -S hello.c

Lets take a look at the code. We’ll not analyze it in detail but make some observations.
.file "hello.c"
.rodata /* Data Section - string to be printed out */
.string "Welcome C23, ready to hack?"
.text
.globl main
.type main, @function /* Entry point of function */
main:
pushl %ebp
movl %esp, %ebp
subl $8, %esp
andl $-16, %esp
movl $0, %eax
addl $15,%eax
addl $15,%eax
shrl $4, %eax
sall $4, %eax
subl %eax, %esp
subl $12, %esp
pushl $.LC0 /* push the address of the string .LC0 on stack */
call puts /* calls put string provided by std. GNU library*/
addl $16, %esp /* Clean up the stack on return */
movl $0, %eax /* Exit status */
leave
ret
.size main, .-main
.ident "GCC: (GNU) 4.0.2 20051125 (Red Hat 4.0.2-8)"
.note.GNU-stack,"",@progbits

GCC translate hello.c (C code) into assembler code in hello.s. Lets take a look at the assembler code. We'll not analyze it in detail. You do not need to understand every line of code to see what is happening here in the Intel assembly language. The code is broken into sections using the .section command. The .string (in the data section of the code) holds the sting to be written out to the display. The global function name “main” is visible in the .text section (code section). The GNU gcc compiler is identified (GNU 4.0.2 and the Linux version). There is some work on the stack on entry and exit to the main function.

If you want to know more about the x86, stack frames, etc., then check out this short note on [http://www.unixwiz.net/techtips/win32-callconv-asm.html] Intel x86 function-call conventions - assembly vconfiiew.
9.2 Code chain: C to machine code

The assembler removes all comments from the hello.c code. The comments in the code were added not by the assembler but the instructor, just to add a little context to the code. We can see how hello.c and hello.s relate or translate. Note, that the “puts” library code is not part of this assembly code. It is added in the next phase of the compilation. The “linker” pulls in the necessary libraries that are needed to create the machine executable code. In essence, the GNU linker pulls in the standard libraries, resolves the call to the puts function by linking in the real object code for that function among other things.

The linker “ld” takes the input of the hello.s (the assembler code) and produces a runnable executable file (machine code).

[campbell@moose ~]$ as hello.s

The linker creates an executable in a container format understood by the target Linux system. The Executable and Linking Format (ELF) is used by the GNU linker used to produce the executable object file. Typically, programmers don’t call the linker directly. GCC take care of that through pipelining the code through the various compilation stages.

The output of the objdump tool is designed to make it easier to understand the contents of an executable. You can use the “od -x a.out” command to simply dump the object code in hexadecimal (the “-x” switch) form. If you try and look at the executable using emacs you will not be able to understand its binary representation (try it). In contrast, objdump displays all the headers within the a.out binary. It essentially performs a reverse engineering job (dissembler) on the binary code (meaning the 1s and 0s of the machine code) - disassembling the executable sections of the binary code.

The output of the objdump tool looks like this:
[campbell@moose ~]$ objdump -x -d -S a.out

a.out: file format elf32-i386
a.out
architecture: i386, flags 0x00000011:
HAS_RELOC, HAS_SYMS
start address 0x00000000

Sections:

<table>
<thead>
<tr>
<th>#</th>
<th>Name</th>
<th>Size</th>
<th>VMA</th>
<th>LMA</th>
<th>File off</th>
<th>Algn</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>.text</td>
<td>00000033</td>
<td>00000000</td>
<td>00000000</td>
<td>00000034</td>
<td>2**2</td>
</tr>
<tr>
<td></td>
<td>CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>.data</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000068</td>
<td>2**2</td>
</tr>
<tr>
<td></td>
<td>CONTENTS, ALLOC, LOAD, DATA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>.bss</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000068</td>
<td>2**2</td>
</tr>
<tr>
<td></td>
<td>ALLOC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>.rodata</td>
<td>0000001c</td>
<td>00000000</td>
<td>00000000</td>
<td>00000068</td>
<td>2**0</td>
</tr>
<tr>
<td></td>
<td>CONTENTS, ALLOC, LOAD, READONLY, DATA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>.comment</td>
<td>0000002d</td>
<td>00000000</td>
<td>00000000</td>
<td>00000084</td>
<td>2**0</td>
</tr>
<tr>
<td></td>
<td>CONTENTS, READONLY</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>.note.GNU-stack</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>000000b1</td>
<td>2**0</td>
</tr>
<tr>
<td></td>
<td>CONTENTS, READONLY</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

SYMBOL TABLE:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Type</th>
<th>Value</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>l df</td>
<td><em>ABS</em></td>
<td>00000000 hello.c</td>
</tr>
<tr>
<td>0</td>
<td>l d</td>
<td>.text</td>
<td>00000000 .text</td>
</tr>
<tr>
<td>0</td>
<td>l d</td>
<td>.data</td>
<td>00000000 .data</td>
</tr>
<tr>
<td>0</td>
<td>l d</td>
<td>.bss</td>
<td>00000000 .bss</td>
</tr>
<tr>
<td>0</td>
<td>l d</td>
<td>.rodata</td>
<td>00000000 .rodata</td>
</tr>
<tr>
<td>0</td>
<td>l d</td>
<td>.note.GNU-stack</td>
<td>00000000 .note.GNU-stack</td>
</tr>
<tr>
<td>0</td>
<td>l d</td>
<td>.comment</td>
<td>00000000 .comment</td>
</tr>
<tr>
<td>0</td>
<td>g</td>
<td>.text</td>
<td>00000033 main</td>
</tr>
<tr>
<td>0</td>
<td><em>UND</em></td>
<td>00000000    puts</td>
<td></td>
</tr>
</tbody>
</table>

Disassembly of section .text:

00000000 <main>:

 0:   55 push %ebp
 1:   89 e5 mov %esp,%ebp
 3:   83 ec 08 sub $0x8,%esp
 6:   83 e4 f0 and $0xfffffffff0,%esp
 9:   b8 00 00 00 00 mov $0x0,%eax
 e:   83 c0 0f add $0xf,%eax
11:   83 c0 0f add $0xf,%eax
14:   c1 e8 04 shr $0x4,%eax
17:   c1 e0 04 shl $0x4,%eax
1a:   c2 29 c4 sub %eax,%esp
1c:   83 ec 0c sub $0xc,%esp
1f:   68 00 00 00 00 push $0x0
 20: R_386_32 .rodata
 24: e8 fc ff ff ff call 25 <main+0x25>
 25: R_386_PC32 puts
 29:   83 c4 10 add $0x10,%esp
 2c:   b8 00 00 00 00 mov $0x0,%eax
 31:   c9 leave
 32:   c3 ret
10 Logging out

OK, we ready to logout from our session on moose.cs.dartmouth.edu.

[campbell@moose ~]$ logout
Connection to moose.cs.dartmouth.edu closed.

It's been a good start. We have covered a number of important issues that we will revisit in the course.

11 Housekeeping - things you need to do

11.1 Linux accounts - there is an algorithm described here!

Follow the pseudo code:

Do you have a department linux account? If you do that is great you can use that account for this course. If you don’t have an account make sure you give me your name and two two preferences for a linux login name (8 characters or less, all lower case alphanumeric and - no _ other punctuation). Wayne will setup the accounts within 24 hours. He will blitz you your new account information with an initial passwords. You can change this initial password using Linux command $passwd (man passwd). To do that you will have to ssh onto galehead.cs.dartmouth.edu and type passwd.

How may asynchronous tasks (i.e., processes) are there? What does each task do? How do they communicate?

11.2 Access to Sudikoff after hours

In order to obtain access to Sudikoff after hours, and to get into Sudikoff’s Lab 001, you will need to have your Dartmouth ID card activated for the appropriate access. To do this, stop by and see Kelly at 101 Sudikoff on a weekday between 8:30am-12:00pm, or 1:00-4:00pm, and bring your Dartmouth ID card. Inform Kelly that you are taking CS23, and require access to Lab 001. You will have to fill out and sign a form stating that you understand the various policies about access to the labs in Sudikoff.

Keep in mind that it may take 24 hours for access to be activated, so please plan ahead!

Please note that the exterior doors of Sudikoff are automatically locked after 6:00pm weekdays, and also every weekend and holiday. In addition, the laboratory doors are locked at all times. You will need your access card to pass through locked doors.

11.3 Remotely accessing CS Linux machines

Assuming you have a Mac or Window laptop:
**Window** Students wishing to connect to CS Linux machines from their own Windows machines can use PuTTY (SSH client). People wanting to do this check out the [PuTTY/Cygwin tutorial (in PDF)](http://www.cs.dartmouth.edu/~campbell/cs23/putty-cygwin-tutorial.pdf) written by Ben Meister (CS23 007 student).

The tutorial also covers Cygwin which is a Linux-like environment for Windows that consists of a DLL (cygwin1.dll) that offers a Linux API emulation layer. Cygwin is not Linux and while it is convenient to run C and Linux like commands on your laptop you would be better to run your code on the Linux machines directly using PuTTY.

**Mac** Mac OSX Unix conforms to POSIX specifications for the C API, shell utilities, and threads and can compile and run your existing code. This is really exiting for Unix/Linux development.

You can also use the ssl command line to remotely log on to computers, as discussed in the lecture. You can use the Terminal application, or an `xterm` - but you'll need X11 to do that: If you would like too run the X Window System (more commonly called X11 on your Mac you need to install X11. This includes the GNU tools such as gcc, make, etc. - very cool! X11 is available as an optional install on the Mac OS X v10.3 Panther, and Mac OS X v10.4 Tiger install disks. It comes preinstalled with Mac OS X Leopard (so you are all set). If you wish to do the install and run X11 then checkout this note on [configuring and running X11 applications on Mac OS X](http://developer.apple.com/opensource/tools/runningx11.html).

### 12 X-hour Tuesday

If your Linux account is set up by the X-hour then do the following. Read through these lecture notes and execute the commands and code as best you can - write a simple program using the emacs editor (read the emacs tutorial). Check out the gcc options to look at the assembler code.

For information about the lab and the Linux machines see: [Sudikoff Lab 101 tour](http://www.cs.dartmouth.edu/~wbc/suditour/001/index.html).

If you need an install disk to install X11 on your Mac then we can provide one.

We plan to show you a demo of one of the robots, so you can get a feel for the project.

### 13 Reading material for the next class on Wednesday

Throughout the course we will read a number of articles that relate to programming and the software development cycle. Some will be anecdotal based on experience working in the software industry and others more academic. We will discuss these in class. It is important that you read them and come armed with thoughts, opinions, questions (all goes towards that 15% class contribution). It is not a quiz but it will be clear to me if students haven't read them.

Please read this material before the next class Wednesday 9, January:
14 Useful links

Here are some useful links cited in the notes. Please read them. I’ll only put links to material you need to read.

[http://www2.lib.uchicago.edu/keith/tcl-course/emacs-tutorial.html] Emacs Tutorial
[http://www.unixwiz.net/techtips/win32-callconv-asm.html] Intel x86 Function-call Conventions - Assembly View