Lecture 01
Introduction to C and UNIX

In this lecture

- **Operating System**
- **Unix system shell**
- **Installing SSH**
- **SHELL Scripting**
- **Imperative Programming**
- **Learning C**
- **C Program Development Process**
- **Compilation, Linking and Preprocessing of programs**
- **ANSI-C Standard**
- **The C compiler – gcc**
- **Moving from Java to C**
- **Additional Readings**
- **Exercises**

1.1 Operating System

Each computer needs an Operating System (OS). At a higher level, an operating system is software that manages coordination between application programs and underlying hardware. OS manages devices such as printers, disks, monitors and manage multiple tasks such as processes. UNIX is an operating system. The figure one demonstrates the high level view of the layers of a computer system. It demonstrates that the end users interface with the computer at the application level, while programmers deal with system utilities and operating system level details.

Copyright @ 2011 Ananda Gunawardena

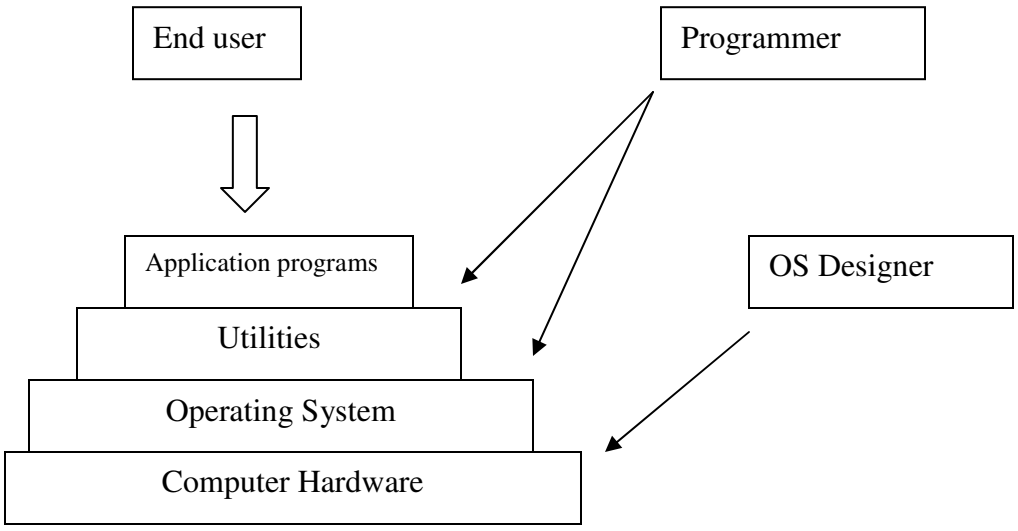


Figure 1

Unix is an operating system developed by AT&T in the late 60's. BSD (Berkeley Unix) and Linux, are unix-like operating systems that are widely used in servers and many other platforms such as portable devices. Linux, an open source version of Unix-like operating system was first developed by Linus Torvalds. Linux has become a popular operating system used by many devices. Many volunteer developers have contributed to the development of many linux based resources. Linux is free, open source, and have very low hardware requirements. This makes linux a popular choice for devices with limited hardware capabilities. Linux is also installed in low cost personal computers.

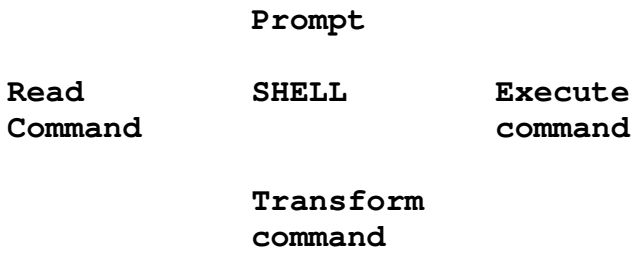
In this course, we will begin with a basic introduction to the unix operating system. We will be using Andrew Linux and we will see how we can use the power of unix to manipulate the Andrew File System (AFS) and use unix tools, C programming and shell and perl scripting to accomplish interesting tasks. Our focus would be on the unix features that are more directly related to writing, debugging and maintaining C programs. We will also focus on unix shell scripting, so we can develop powerful scripts for managing tasks such as unix system calls, file manipulation etc.

Copyright @ 2011 Ananda Gunawardena

1.2 The Unix System Shell

Note: You must attend the recitation and/or read below about installing SSH to learn how to use the UNIX shell in this course.

Although UNIX can be accessed with graphical user interfaces such as x-windows, we will be focusing our work at the shell level. After login, we interact with the unix through a program called a "unix shell". Shell is a command interpreter. In other words, you provide commands that you would like to be interpreted by the shell. The command interpretation cycle of the shell is as follows.



First the shell prompts for the command. A command is entered, then interpreted and executed by the system. An example would be

% uname –o

This finds out which version of the operating system you are running.

In general, uname prints system information. Uname –a will print all information.

Another example would be ls command:

- **ls**

The ls command list all files and folders in current directory. The same command with option such as –R

- **ls –R**

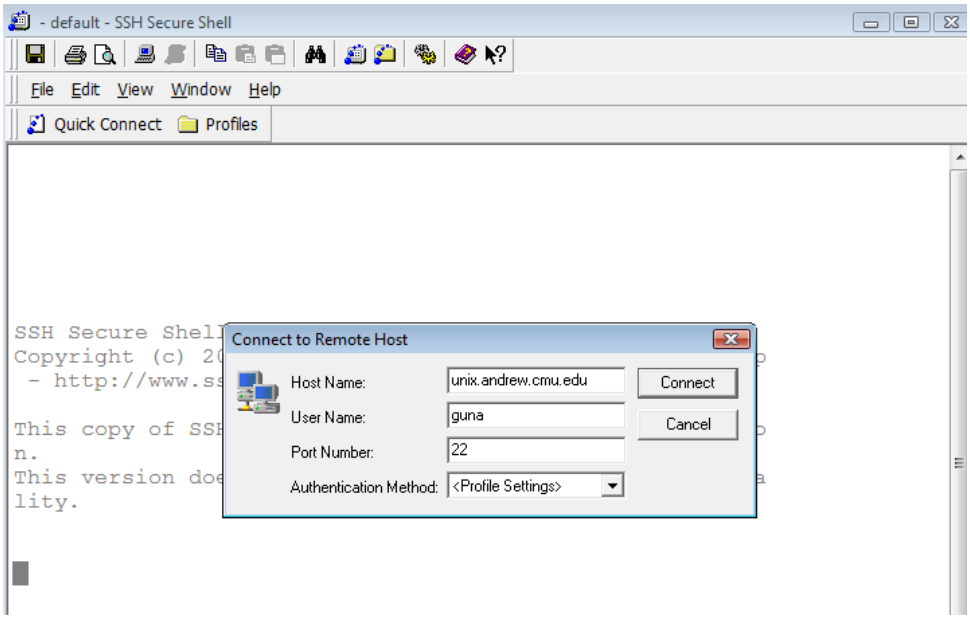
displays all files within the current folder and its sub folders recursively.

1.3 Installing SSH

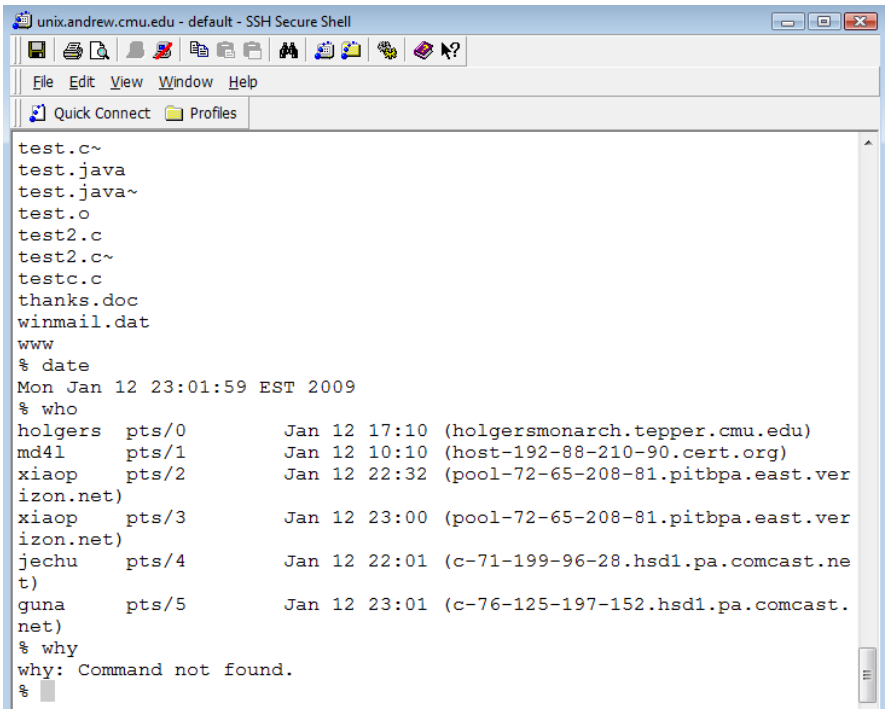
In order to see how the shell works, we need to be able to have access to Andrew linux shell. You can access SHELL from machines in Gates Linux Clusters or to set up SSH (secure shell) see Bb→ External Links → Setting up SSH in

Copyright @ 2011 Ananda Gunawardena

your machine. Once the SSH client is installed, then you can connect to your Andrew account by typing your Andrew login information. Your host is “unix.andrew.cmu.edu”. This will connect you to one of the available Andrew machines.



Once logged in, you will have access to the unix shell that will interpret the commands you provide.



Copyright @ 2011 Ananda Gunawardena

Once a command is given to the shell, for example

% cp file1 file2

The shell interprets the command and executes it. Virtually anything you do on Andrew linux is done by issuing a command at the shell level. The generic form of a command is
% command arg1, arg2, ….

Exercise: open unix shell and type commands like: **who, date, cat filename**
And see what happens. To find out more about what these commands mean, type: **man command** at the unix prompt.

Here are some other things you can try:

% mkdir 15123 --- makes a directory in your Andrew home
% cd 15123 --- change directory to subdirectory 15123
% emacs cheatingPolicy.txt --- start editing a file in linux. See the document given in Bb and type it using an editor.

- ***We will cover emacs editor commands in the recitation. But you can use any editor such as pico or vim. Feel free to use any ASCII editor. You can find most of the common emacs commands in Bb->external links.***

% cp cheatingPolicy.txt /afs/andrew/course/15/123/handin/lab0/YOUR_ID
--- copies your file to submission folder
% cd /afs/andrew/course/15/123/handin/lab0/YOUR_ID
--- now switch to lab0 folder
% ls --- lists all files in the directory. (You should see your submission. Make sure you do this after submitting each assignment)
% ls -l --- show long listings of all files in the directory.

A typical record looks like this

```
-rw-r--r--  1 guna       staff 1749118 Mar 27  2005 Tsunami.zip
drwxr-xr-x  4 guna       staff    2048 Jul 16 14:28 WebSite1
```

% fs la --- see what permission you have for the current folder
% fs sa . system:anyuser none
--- remove all file permission from any user
To find the description of any command, simply type
% man command (eg: man ls)
(at the : prompt press the space bar to see more or type Q to quit the man pages)

Linux manual pages are very handy tool for us to find out how to use all the linux commands we need in this course and beyond. A summary of commonly used commands are given below.

Copyright @ 2011 Ananda Gunawardena

Summary of commonly used shell commands

login	authenticates your ID and password then connects you to a shell
logout	breaks your connection to the shell and terminates any programs you were running.
passwd	lets you change your password.
who	lists all the other user connected to this Unix machine.
whoami	tells who is logged onto this terminal.
finger	displays information about a particular user (user does not have to be logged in)
whois	completely different than who. Treats arg as domain name and looks up info about that domain name
date	displays current date and time
pwd	displays the full path of the dir you are in right now (current working directory).
ls	displays list of files in specified directory.
cat	displays contents of specified file or can be used to concatenate files.
od	displays specified binary file in octal or hex.
more	displays contents of specified file one screen at a time.
less	similar to more.
head	displays first few lines at top of file.
tail	displays last few lines at bottom of file.
files	examine specified file and make good guess as to its type.
cp	copy a file. If copy is in same dir it must be given a different name.
mv	move a file or directory
rm	delete specified file(s). Warning! this command can be a WMD
find	searches specified directories for files matching specified criteria. Displays a list of matching file.
grep	searches inside specified file(s) for the specified text pattern(s). For each match a filename and the matching line in displayed.
wc	for specified file(s), displays number of lines, words, chars and bytes in that file.
diff	displays the line by line differences between two text files.
comm	like diff but displays common lines instead of differences.
cmp	like diff, but for binary files.
cd	step into the specified directory
mkdir	create a new directory.
rmdir	remove specified directory.
~	synonym for your home dir. Example: **cd ~** puts you in your home directory as at login time.

courtesy: Tim Hoffman

1.4 SHELL SCRIPTING

Number of shell commands can be combined to create a program, which we call a shell script that will perform number of shell commands at the same time. For example, one can read a file of Andrew ID's and then create a directory for each ID in the file. SHELL scripting is a great way to automate tasks that are too tedious to do manually. We will discuss more about this in the second half of the course.

Copyright @ 2011 Ananda Gunawardena

1.5 IMPERATIVE PROGRAMMING

Imperative programming is a programming paradigm that describes computation in terms of statements that changes the state of the machine. For example, the statement x = x + 1 changes the state of the memory location of x by adding 1 to the existing value of x. Procedural programming describes an imperative programming model where a program consists of one or more procedures, each of which contains statements that changes the state of the program. Imperative programming indicates that programming language constructs are fairly close to the machine. The earliest imperative languages were the machine languages that were used to program computers. Machines languages were hard to code and debug. Over time, high level languages were developed to make the programming process easier. For high level languages, compilers had to be developed so that it takes high level code and translate into machine code that can be executed directly on machines hardware. All programs must eventually be executed at the machine level. Many other paradigms such as object oriented programming and functional programming were developed to make the programming process more structured and correct. However, all programs must eventually be converted into machine instructions.

1.6 LEARNING C

C is a programming language developed in 1970's that allows program development flexibility (low level constructs) and tools (libraries) to write efficient code. Most low level programmers prefer C, as Java programming language forces more rigorous structure and Object Oriented programming style. But in applications where many millions of data needs to be processed, speed is critical, or better programmer controlled memory management is required, java may lack the efficiency to provide a practical solution. C is also widely used in numerical applications such as solving large systems of equations, developing low level utilities such as device drivers, programming data compression algorithms, graphics applications, and computational geometry applications and game programming. C places a high "trust" on the programmer and allows the programmer to use constructs freely. This provides flexibility and a great deal of power, but the programmer must take great care in developing, debugging and maintaining programs.

Copyright @ 2011 Ananda Gunawardena

C and UNIX provide the ideal programming environment for the experienced programmer. Much of Unix kernel is written in C. Learning to program in C gives a set of low level programming tools that is unmatched by any other programming language. The power of C is its ability to express programming instructions using a combination of low level and high level constructs and has great of control over how your programs allocate, use and free memory.

1.7 Program Development Process

High level languages like Java uses classes and object interactions (although the initial OO design may be harder for some) and the programmers have access to a large well documented API. Java programs are easier to debug, since dynamic memory is automatically managed (automated garbage collector) and error messages and exceptions are more descriptive. On the other hand, C programs are harder to develop and debug but they are more efficient in execution. C programmers must learn to do procedural decomposition in order to write good programs. C programmers must learn how to use a debugger such as gdb in order to efficiently debug programs. C program management can be automated using *make* files. We will discuss gdb and makefile concepts later in the course.

1.8 Preprocessing, Compilation, and Linking

There are 3 major steps to developing a C program.

- Editing – The process of creating the source code
- Compiling – The process of translating source code to object code
- Linking – The process of linking all libraries and other object codes to generate the executable code

The process of editing allows C programs to be written using a UNIX editor such as emacs or vim. The preprocessing is performed to replace certain text in the file by others. For example:

#define pi 3.14

The above statement causes C preprocessor to replace all “pi” references by 3.14. Pi can be referred to as a “macro”. We will discuss more about Macros later in the course.

Copyright @ 2011 Ananda Gunawardena

We can also include an external library (that is not part of the standard libraries) such as “mylibrary.h”.

```
#include “mylibrary.h”
#include <stdio.h>
```

Note that the “ “ is used to distinguish external libraries from standard libraries such as stdio.h.

1.9 ANSI C Standard

American National Standards Institute (ANSI) formed a committee to establish a C standard for all programmers. The ANSI C standard is based on an extended form of traditional C and allows greater portability among platforms. The most important ANCI C feature is the syntax of declaring and defining functions. The parameter types are declared inside the function parameter list. This allows compilers to easily detect mismatched function call arguments. Other ANSI C features include assignment of user defined structures, enumeration, single precision floating point arithmetic (32 bits) (traditional C supports only double precision arithmetic – 64 bits). The ANSI standard also bans interchange of pointers and integers without explicit type conversions. In ANSI programming all variables must be declared before any statements. For example;

```
int y = 10;
Y = y + 1;
int x = 12;
```

may **NOT** compile under ANSI standard.

ANSI C does not allow commenting using “//” and must use /* … */ style of commenting.

ANSI C also provides standard libraries for IO, strings, math, system calls etc. gcc compiler conform to ANSI standard. You can compile your program under –ansi flag to make sure it conforms to ANSI standards. To check if your program is written according to ANSI C, compile as

- **gcc –ansi myprogram.c**

if the program is syntactically correct, if the proper libraries are available for you to link, then a file called a.out is created. The file a.out is a binary file that can

Copyright @ 2011 Ananda Gunawardena

be executed only under the platform the program was developed in. To see the current files in your working folder type
% ls -l

To run the program, you type
% ./a.out

The shell command looks for the binaries in the working folder and executes the program.

In this course, we will be using many switches during compilation to help us debug and manage and make our programs more efficient. For examples we will typically compile code using

% gcc -Wall -ansi -pedantic -O2 main.c

-ansi -pedantic -W -Wall -O2 these are switches that customize the behavior of our compilation. Remember we promised to show you how to get all the help the compiler can give you. Using these switches tells the compiler to apply more scrutiny to your code so that those things which can be detected at compile time will be reported to you as warnings and errors. The -ansi switch warns you when your code does non-ANSI things like call functions that are not part of the standard ANSI libraries or mixing code and data. The -pedantic -W -Wall switches are requests for more scrutiny on such things as unused arguments passed into functions. The -O2 ("oh two" not "zero two") switch is calling for code optimization at a level of 2. This course does not really address code optimization with any rigor or formality, but -O2 switch does detect use of un-initialized variables. There are many other switches you can in your compilation command that we will not cover in this course. The history of how these switches came about - and what things they detect is a rather random and spurious. As the language evolved switches were added or changed in a very ad-hoc manner. For example -Wall means "warnings all". So you might think that means it warns on all infractions. Well, not quite. If you want to detect failure to use argv or argc then you must add -W which is just "warnings". Go figure. Better yet, use them as shown and never ignore warnings. In this course you are never allowed to hand in code with warnings. You will be penalized.
Source: Tim Hoffman

1.10 The C Compiler

A compiler, such as GNU C Compiler(gcc) translates a program written in a high level language to object code that can be interpreted and executed by the underlying hardware. Compilers go through multiple levels of processing such as, syntax checking, pre-processing macros and libraries, object code generation, linking, and optimization among many other things. A course in compiler

Copyright @ 2011 Ananda Gunawardena

design will expose you to many of the tasks a compiler typically does. Writing a compiler is a substantial undertaking and one that requires a lot of attention to detail and understanding of many theoretical concepts in computer science.

1.11 Jobs and Processes

Each C program executable, when executed creates a process. Unix can maintain multiple processes at the same time. Each process is a job executed by the shell. To see what current jobs are running in your environment, we type

```
% jobs -l
```

To see what processes are running in the background of your environment, we type

```
% ps
  PID TTY          TIME CMD
31977 pts/3    00:00:00 csh
31988 pts/3    00:00:00 ps
```

Any process can be killed by using the command
`% kill PID` --- PID is the process ID

Killing a Process

It is very common that as we do programming assignments in this course, we run into situations where program does not terminate. This can be caused by an infinite loop or some weird behavior in the program. In such cases, we need to forcefully terminate the program by using

Control C

Ctrl-C kills the foreground process. If you press

Cntrl-Z, then the current process is placed in the background and shell returns a prompt.

You can bring background processes to foreground by typing

% fg

Or find the process ID using ps and kill the process.

Copyright @ 2011 Ananda Gunawardena

1.12 Moving from Java to C

There are some major differences between Java and C programming. Java is an object oriented language where applications are developed using classes that encapsulates the methods and states. Each object instantiated from the class communicates with other objects by sending messages. Java programs are interpreted and runs under the Java virtual machine(JVM). Java programs are converted into byte code that executes under JVM. This allows Java programs to be portable across multiple platforms.
On the other hand, C is a procedural programming language where programs are developed using procedural decomposition. That is, application tasks are divided into meaningful procedures and procedures are called from the main program to solve the problem. An executable version of the program is called C binaries and C binaries are not portable across platforms.

One of the best ways for you to start learning C (if you are a die hard java programmer) is to convert a simple Java program into C code. Let us consider the following java program. This is a java program that sorts a set of random numbers using a sorting algorithm called bubble sort. It takes the number of elements in the array as a command line argument. Assuming that the name of the java source file is javasort.java, you can run the program by typing

- javac javasort.java
- java javasort 10000

where 10000 is the number of elements in the array. When you consider command line arguments, this number can be obtained by using args[0]

```
import java.io.*;
import java.util.*;

public class javasort {

    public static void main(String[] args) throws Exception {

      long begin, end;
      begin = System.currentTimeMillis();
      int n = Integer.parseInt(args[0]);
      int[] A = new int[n];
      Random R = new Random();
      for (int i=0;i<n;i++)
          A[i] = R.nextInt();
```

Copyright @ 2011 Ananda Gunawardena

```
      for (int i=0; i<n;i++)
        for (int j=0; j<n-1;j++)
            if (A[j+1]<A[j])
                {int temp = A[j];
                  A[j] = A[j+1];
                  A[j+1] = temp;
                    }

      end = System.currentTimeMillis();
      System.out.println("The elapsed time is " + (end-begin)/1000);
    }
}
```

Lets think about how to convert this code to C. Java import statements allows specific libraries to be included in the project. In C, inclusion of external libraries are accomplished by using include statement. First you need to find equivalent libraries for java.io.* and java.util.*

Java.io.* is used for System.out.println operation and java.util.* is used for Random class.

There are few other things you need to convert. Java command line arguments args[] need to be replaced by argc and char* argv[]. In C, argc gives the number of command line arguments and argv[] is the array of char* or Strings. char* is interpreted as a pointer to (or an address of) a char. String in C is an array of characters ending with null character '\0'. You can see the equivalent Java and C programs at the end of this lecture.

Copyright @ 2011 Ananda Gunawardena

```java
import java.io.*;
import java.util.*;

public class javasort {

    public static void main(String[] args)
         throws Exception {

        long begin, end;
        begin = System.currentTimeMillis();
       int n = Integer.parseInt(args[0]);
       int[] A = new int[n];
       Random R = new Random();
       for (int i=0;i<n;i++)
            A[i] = R.nextInt();

       for (int i=0; i<n;i++)
           for (int j=0; j<n-1;j++)
                if (A[j+1]<A[j])
                    {int temp = A[j];
                       A[j] = A[j+1];
                       A[j+1] = temp;
                     }

        end = System.currentTimeMillis();
        System.out.println("The elapsed time is " +
(end-begin)/1000);
    }
}
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int parseInt(char*);

int main(int argc, char* argv[]){
      long begin, end;
      int i, j;
      begin = time(NULL);
      int n = parseInt(argv[1]);
      int A[n];
      for (i=0;i<n;i++)
            A[i] = random();

      for (i=0; i<n;i++)
          for (j=0; j<n-1;j++)
               if (A[j+1]<A[j])
                   {int temp = A[j];
                      A[j] = A[j+1];
                      A[j+1] = temp;
                    }

       end = time(NULL);
 printf("The elapsed time is %lf ",end-begin);
    }

int parseInt(char* s){
      // complete this code
      // note: all strings end with null '\0'
    return 0;
```

Additional Readings:

[1] K & R – Chapter 1 – Tutorial Introduction to C – pages 5-21

Exercises

1.1: What libraries in C contain a functions equivalent to java println and Random.nextInt?

1.2: Write a program to search an array of strings in Java and then convert it to a C program. (see example above)

1.3: Using unix commands find out how many sub-folders are under the current working folder. You need to recursively count all the sub-folders (later we will write a script to do this. For now, we do this manually)

Copyright @ 2011 Ananda Gunawardena

1.4: type ls –l at the command shell. Interpret the meaning of the output given.

```
-rwxr-xr-x 1 guna nfsnobody 7530 Aug 25 15:20 a.out
-rw-r--r-- 1 guna nfsnobody  846 Aug 25 12:05 csort.c
-rw-r--r-- 1 guna nfsnobody 1061 Aug 25 15:19 javasort.class
-rw-r--r-- 1 guna nfsnobody  687 Aug 25 11:47 javasort.java
-rw-r--r-- 1 guna nfsnobody   89 Apr 30  2008 main.c
```

Copyright @ 2011 Ananda Gunawardena

Answers

1.1: What libraries in C contain a functions equivalent to java println and Random.nextInt?

Ans: println ➔ printf in C in <stdio.h>
Random.nextInt ➔ rand() in <stdlib.h>

1.2: Write a program to search an array of strings in Java and then convert it to a C program.
Ans:

1.3: Using unix commands find out how many sub-folders are under the current working folder. You need to recursively count all the sub-folders (later we will write a script to do this. For now, we do this manually)

Ans: ls -R will recursively display all the files and folders in the current directory. Then you may want to "pipe" the output to "wc" to count the number of entries. For example,
% ls -R | wc -l

1.4: type ls -l at the command shell. Interpret the meaning of the output given.

```
-rwxr-xr-x 1 guna nfsnobody 7530 Aug 25 15:20 a.out
-rw-r--r-- 1 guna nfsnobody  846 Aug 25 12:05 csort.c
-rw-r--r-- 1 guna nfsnobody 1061 Aug 25 15:19 javasort.class
-rw-r--r-- 1 guna nfsnobody  687 Aug 25 11:47 javasort.java
-rw-r--r-- 1 guna nfsnobody   89 Apr 30  2008 main.c
```

Ans: This lists the file type (directory or file), file permission (rwx) for user, groups and other, login id (guna), size of the file, date it is created, file name etc.

Copyright @ 2011 Ananda Gunawardena