COMPUTATION FOR ENGINEERS (IC-150P)

Lab Manual

ABSTRACT

The document serves as the lab manual for the course IC-150P

AUGUST-DECEMBER 2015

{By: Dr. Abhishek Dixit, Dr. Kaustav Sarkar, Dr. Varsha Jain, Dr. Nitu Kumari}

Indian Institute of Technology Mandi

Table of Contents

I. Introduction 1

1. Essential Linux, Editor, and Debugger Commands 1

a. Getting Started 1

b. Linux Shell Commands 1

c. Special characters in file and directory names 2

d. Editor 2

e. Debugger 3

2. Standards for C Code 3

a. Names 3

b. Internal Documentation 4

c. Layout 6

d. Useful Features 6

II. Week I 8

1. Programming Assignment 1.1: Linux Commands and Text Editors 8

2. Problem 1.2 (Use an editor of your choice): 8

3. Problem 1.3 (Linux Commands): 8

I. Introduction

In this chapter, we introduce the basics of Linux, and C programming.

1. Essential Linux, Editor, and Debugger Commands

a. *Getting Started*

1. To login, type your username (Your roll number in lowercase letters) at the **Login:** prompt, and your password (same as your username) at the **Password:** prompt.
2. Open a shell window from Applications -> Accessories -> Terminal menu.

b. *Linux Shell Commands*

Table gives the overview of the important shell commands

Table 1: Shell Commands and their meaning

Commands	Meaning
mkdir *dirname*	Make a directory *dirname*
rmdir *dirname*	Remove the directory *dirname*
cd *dirname*	Change the current working directory to *dirname*
cd	Change the current working directory to the parent directory
cd ~	Change the current working directory to your home directory
pwd	Show your current working directory
mv srcfile destfile	Rename the srcfile as destfile
cp srcfile destfile	Copy one file, srcfile to destfile
cp srcfile(s) destDir	Copy many file, srcfile(s) to destDir
rm -i *file(s)*	Delete *file(s)*
ls -l	List files in the directory with their details (size, time of creation)
gcc –o *prog prog.c*	Compile the C program in the file *prog.c* and create the executable file *prog*
gcc –g –o *prog prog.c*	Compile the C program in the file *prog.c* and create

	the executable file *prog* that can be used for debugging with **gdb**
/prog	Run the program *prog*

c. *Special characters in file and directory names*

Table 2 gives the special characters and their meaning.

Table 2: Characters and their meaning

Characters	Meaning
*	wildcard matches any string
?	matches any single character
~	your home directory

d. *Editor*

The course will use a version of Ubuntu (a popular GNU/Linux distribution). Ubuntu comes with a number of editors that you can use for the purposes of this course. Typically, an editor will be used to input and save your C programs. Some editors that you can use include Emacs, Vim, gedit, kate, and nano. You can use any of these editors for this course (please become familiar with one of them).

Note: The learning curve is very steep in the beginning for Emacs. However, if you still decide to use Emacs, then a set of basic commands is listed below:

Start the Emacs editor from:

Applications -> Accessories -> GNU Emacs menu

The commands in Emacs menu are given in Table 3.

Table 3: Commands in Emacs

Commands	Meaning
save as *filename*	Save the content in the file filename
save	Save the content in the current file
cut	Cut the marked block of text
copy	Copy the marked block of text
paste	Paste the marked block
close	Close the current file
exit	Save the current file and exit Emacs

help->tutorial	An online tutorial on the basic commands of Emacs

e. Debugger

To debug *prog*, start the **gdb** debugger in the Shell window using: **gdb** *prog*

gdb Commands at the (gdb) prompt (Table 4):

Table 4: gdb commands

Commands	Meaning
list	List 10 lines of the C source of *prog*
break *nnn*	Set a break-point: Program execution stops when it reaches line *nnn*
run	Start program execution
continue	Continue execution from a break-point
next	Execute the next line in program and then break. Does not break in functions.
step	Execute the next line, stepping into functions, and then break
<ENTER>	Pressing the <ENTER> key repeats the previous command
print x	Display the value of the the variable x
set x=10	Set the variable x to 10
quit	Quit the debugger

gdb commands can be abbreviated, *e.g.* **l** for list, **b** for break, **n** for next, **p** for print, etc.

2. Standards for C Code

Following a coding standard is part of professional programming. This enhances the readability and quality of the code, and makes it easier for other programmers to read and modify the code.

a. Names

1. To make the code self-documenting, choose meaningful names for variables. Abbreviations may be used so long as they are widely accepted. A good test of names is: *can you read your code to a fellow programmer over the phone?*
2. For names that consist of multiple words, capitalize the first letter of each word.
3. Distinguish classes of names as follows:

Functions, Macros, Types, Classes: First letter uppercase (eg. GetInput(), LengthType, Compute()).

***Constants*:** All uppercase, separate words with '_' (eg. MAX_LINE_LEN, PI, VOTING_AGE)

***Variables*:** First letter lowercase (eg. roomMessDistance, inBuf, myId, windowHt, wallWidth)

4. Names should differ in more than one character, especially if they are of the same type. E.g., for the transmitter and receiver buffers, txBuf and rxBuf differ in only the first character which occurs on adjacent keys on the keyboard. txBuf and rcvBuf is a better choice.
5. Use the abbreviations in the following Table to identify particular names:

Table 5: Abbreviations for various Data types

Abbreviations	Type
Type	Defined type (e.g. typedef struct {...} MsgType;)
Ptr	Pointer (e.g. bufPtr, msgPtr, pktPtr)
Fl	Boolean (e.g. moreFl)
Str	String (e.g. promptStr)
Chr	Character (e.g. inChr, outChr)
Tab	Table (e.g. relayTab, relayTabPtr)
Num	Number (e.g. numCourses) ["No" could be confused with the negative]
Ctrl	Control (e.g. CTRL_C)
Cmd	Command (e.g. LastCmd)
Cnt	Count (e.g. wordCnt)
Que	Queue (e.g. inBufQuePtr)
Len	Length (e.g. roadLen)

b. Internal Documentation

1. Apart from external documentation such as pseudo-code, flow-charts, state transition diagrams, function-call hierarchies, and prose, the program files should contain documentation. Begin **each file** with a comment, as shown in the example below:

```
/*****************************************************************
* sort.c – for sorting integers  filename with one-line description
```

* Purpose: uses bubble-sort algorithm... *purpose in detail*

* Compilation: use the supplied makefile *Instructions for compiling*

* Revision history: Chronological list of changes/bug-fixes

A. Programmer, 7/7/77

released version 1.0

C. Debugger, 8/8/88

fixed stack overflow with null input

Eager B. Eaver, 9/9/99

added ANewProc() to support 3-D

* Bugs: Known bugs/limitation/testing to be done

The program occasionally crashes when two users access the database simultaneously during the new moon.

***/

2. Declare **each variable** on a separate line, followed by an inline comment explaining the purpose of the variable. Use

/**

char *inBuf; // buffer for received keystrokes

char *outBuf; // buffer for text going to the printer

rather than

char *inBuf, *outBuf; // input and output buffers

***/

3. If there are a large number of variables, group them in blocks by function, and alphabetically within each block. Note: temporary variables such as loop indices need not follow some of these rules.
4. Preceding **each function**, include a comment block as follows:

/***

* GetInput - get input from the keyboard.

* Args: Stores the string in the buffer buf, max size is bufSize

* Returns: number of characters stored in buf or -1 on error.

* Method: a brief description if necessary.

* Bugs: list known bugs and limitations

* To be done: if anything

**/

/***

int GetString(char *buf, int bufSize)

{

```
...
}         /*  End of GetString() */
    ******************************************************************/
```

5. Within the body of the function, on separate lines at the start of **each major block** (if, while for, switch), describe briefly the purpose and peculiarities of the block. For obscure statements, include an inline comment.
6. Avoid obvious comments such as:

```
i++; /* increment i */
```

c. Layout

Indent the code according to the following scheme and use blank lines to indicate breaks in the flow of control. This improves the readability.

```
/******************************************************************
while (moreFl)          /* The main loop, terminates when done */
{
        if (i == 2)
DoSomethingAppropriate();
        else
DoSomethingElse();
        for (j = 0; j < maxFile; j++)    /* Mumbo-jumbo for each file */
                {
                        total += table[i].wordCnt;
                        i = j + k;
cnt = j – 1;
                }
} /* while (moreFl) */
******************************************************************/
```

d. Useful Features

Some C language features that will enhance the quality of your code:

Header files: collect macro, type, constant and global variable declarations and prototypes for public functions in one or more .h include files. Never include code in .h files. Group logically related functions into separate .c files. Use a utility such as *make* to automate rebuilding the program.

Information hiding: declaring a function static makes it private to the module (i.e., file) in which it is declared. Likewise for data. In a header file, define *#define PRIVATE static* and use it for private functions and data:

PRIVATE int myCount;

PRIVATE void LocalFunc();

Function prototypes: use these to enable the compiler to check for consistency of arguments. In a header file, include function prototypes for all public functions. Remember to use void for functions that do not return any value.

Enumerated types: use *enum* rather than a sequence of *#defines*. This is less error-prone and enables the compiler to check type consistency.

Type casts: use explicit typecasts to avoid warning messages from the compiler about operands of different types.

II. Week I

1. Programming Assignment 1.1: Linux Commands and Text Editors

1. Login
2. Editor of your choice (e.g., EMACS Editor)
3. Linux commands:
    1. Creation of directory
    2. Rename a file
    3. Copy a file
    4. Delete a file

2. Problem 1.2 (Use an editor of your choice):

1. Use the editor to type a letter to your friend describing your first semester experiences at IIT Mandi.
2. It must be at least three paragraphs with seven or more sentences each.
3. Delete the second and last sentences of the second paragraph.
4. Move the first sentence of third paragraph as the second sentence of second paragraph - you should not retype.
5. Copy the fourth sentence of first paragraph as the last sentence of second paragraph.
6. Now read the letter and edit (delete and insert) necessary words/sentences so that it sounds sensible.

3. Problem 1.3 (Linux Commands):

1. Save the letter of Problem-0 as a "file".
2. You want to send the same letter to five more friends
    1. Make five copies of the same (use the cp command)
    2. Open the copies and change the names of your friends
    3. Delete the file containing the letter for your third friend (use "rm" command)
    4. You wanted to store these files in a separate place that you could remember.
    5. So, create a directory called "Assignment0" and move these files to it.