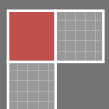




# UNIX & SHELL PROGRAMMING LAB MANUAL

II B. Tech I Semester

Department of Computer Science & Engineering  
Swarnandhra College of Engineering & Technology  
Seetharampuram, Narasapur 534280



---

CONTENTS

---

Lab Objective	3
Guidelines	4
List of Lab Exercises	6
Basic UNIX Commands	9
UNIX Command Exercises	18
Solutions to Exercise	20
Week – 1	22
Week – 2	23
Week – 3	24
Week – 4	25
grep Exercise	26
sed Exercise	27
awk Exercise	29
Week – 5	30
Week – 6	31
Week – 7	32
Week – 8	33
Week – 9	35
Shell Programming Exercises	36
Week – 10	37
Week – 11	41
Week – 12	45
Bibliography	49

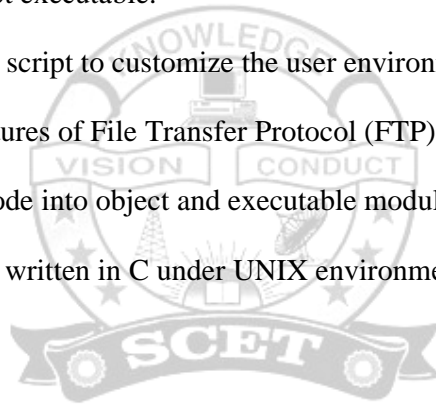


## LAB OBJECTIVE

---

Upon successful completion of this Lab the student will be able to:

1. Demonstrate how to use the following Bourne Shell commands: cat, grep, ls, more, ps, chmod, finger, ftp, etc.
2. Use the following Bourne Shell constructs: test, if then, if then else, if then elif, for, while, until, and case.
3. Learn tracing mechanisms (for debugging), user variables, BourneShell variables, read-only variables, positional parameters, reading input to a BourneShell script, command substitution, comments, and exporting variables. In addition, test on numeric values, test on file type, and test on character strings are covered.
4. Copy, move, and delete files and directories
5. Write moderately complex Shell scripts.
6. Make a Shell script executable.
7. Create a ".profile" script to customize the user environment.
8. Use advanced features of File Transfer Protocol (FTP)
9. Compile source code into object and executable modules.
10. Execute programs written in C under UNIX environment



## GUIDELINES TO STUDENTS

---

### How to Run Shell Scripts

There are two ways you can execute your shell scripts. Once you have created a script file:

#### Method 1

Pass the file as an argument to the shell that you want to interpret your script.

Step 1 : create the script using vi, ex or ed

For example, the script file show has the following lines

```
echo Here is the date and time
date
```

Step 2 : To run the script, pass the filename as an argument to the sh (shell )

```
$ sh show
Here is the date and time
Sat jun 03 13:40:15 PST 2006
```

#### Method 2:

Make your script executable using the chmod command.

When we create a file, by default it is created with read and write permission turned on and execute permission turned off. A file can be made executable using chmod.

Step 1 : create the script using vi, ex or ed

For example, the script file show has the following lines

```
echo Here is the date and time
date
```

Step 2 : Make the file executable

```
$ chmod u+x script_file
$ chmod u+x show
```

Step 3 : To run the script, just type the filename

```
$ show
Here is the date and time
Sat jun 03 13:40:15 PST 2006
```

### How to run C programs

Step 1 : Use an editor, such as vi, ex, or ed to write the program. The name of the file containing the program should end in .c.

For example, the file show.c contains the following lines :

```
main()
{
    printf(" welcome to SCET ");
}
```

Step 2 : Submit the file to GCC ( the GNU C Compiler )

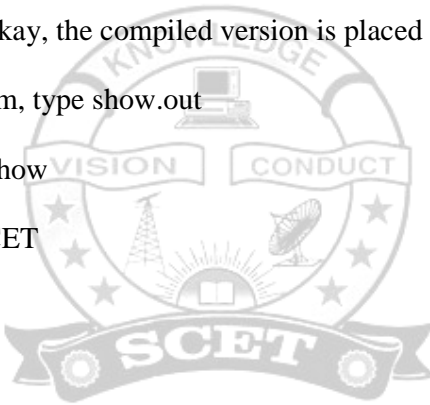
```
$ gcc -o show show.c
```

If the program is okay, the compiled version is placed in a file called show.out

Step 3 : To run the program, type show.out

```
$ show.out or ./show
```

Welcome to SCET



## LIST OF LAB EXERCISES

---

### WEEK1

#### Session 1

1. Log in to the system
2. Use Vi editor to create a file called myfile.txt which contain some text.
3. Correct typing errors during creation
4. Save the file
5. Logout of the file

#### Session 2

- a) Log into the system
- b) Open the file created in session 1
- c) Add some text
- d) Change some text
- e) delete some text
- f) Save the changes
- g) Logout of the system

### WEEK2

- a) log into the system
  - b) Use the cat command to create a file containing the following data. Call it mutable use tabs to separate the fields
- |      |      |       |
|------|------|-------|
| 1425 | ravi | 15.65 |
| 4320 | ramu | 26.27 |
| 6830 | sita | 36.15 |
| 1450 | raju | 21.86 |
- c) use the cat command to display the file, my table
  - d) use the vi command to correct any errors in the file, my table
  - e) use the sort command to sort the file my table according to the first field. Call the sorted file my table(same name)
  - f) print the file my table
  - g) use the cut & paste commands to swap fields 2 and 3 my table. Call it mytable(same name)
  - h) print the new file, my table
  - i) logout of the system

### WEEK3

- a) log in the system
- b) use the appropriate commands to determine ur login shell
- c) use the /etc/passwd file to verify the result of step b.
- d) use the who command redirect the result to a file called myfile1. Use the more command to see the contents of myfile1.

- e) Use the date and who commands in sequence ?(in one line) such that the output of date will display on the screen and the output of who will be redirected to a file called my file2. Use the more command to check the contents of myfile2.
- a) write a sed command that deletes the first character in each line in a file
- b) write a sed command that deletes the character before the last character in each line in a file.
- c) Write a sed command that swaps the first and second words in each line in a file

#### WEEK4

- a) pipe ur /etc/passwd file to awk and print out the home directory of each user.
- b) Develop an interactive grep script that asks for a word and a file name and then tells how many lines contain that word
- c) Repeat
- d) Part using awk

#### WEEK5

- a) Write A shell script that takes a command –line argument and reports on whether it is directory ,a file,or something else
- b) Write a shell script that accepts one or more file name as a arguments and converts all of them to uppercase,provided they exists in the current directory
- c) Write a shell script that determines the period for which a specified user is working on the system

#### WEEK6

- a) write a shell script that accepts a file name starting and ending line numbers as arguments and displays all the lines between the given line numbers
- b) write a shell script that deletes all lines containing a specified word I one or more files supplied as arguments to it.

#### WEEK7

- a) Write a shell script that computes the gross salary of a employee according to the following
  - 1) if basic salary is <1500 then HRA 10% of the basic and DA =90% of the basic
  - 2) if basic salary is >1500 then HRA 500 and DA =98% of the basicThe basic salary is entered interactively through the key board
- b) Write a shell script that accepts two integers as its arguments and computes the value of first number raised to the power of the second number

#### WEEK 8

- a) Write an interactive file handling shell program. Let it offer the user the choice of copying ,removing ,renaming or linking files. Once the use has made a

choice, have the program ask the user for necessary information, such as the file name ,new name and so on.

- b) Write a shell script that takes a login name as command –line argument and reports when that person logs in
- c) Write a shell script which receives two files names as arguments. It should check whether the two file contents are same or not. If they are same then second file should be deleted.

## WEEK 9

- a) Write a shell script that displays a list of all files in the current directory to which the user has read write and execute permissions
- b) Develop an interactive script that asks for a word and file name and then tells how many times that word occurred in the file.
- c) Write a shell script to perform the following string operations.
  - 1) To extract a sub string from a given string
  - 2) To find the length of a given string

## WEEK 10

Write a C program that takes one or more file or directory names as command line input and reports the following information on the file.

- 1) file type
- 2) number of links
- 3) read, write and execute permissions
- 4) time of last access

(Note: use /fstat system calls)

## WEEK 11

Write C program that simulate the following unix commands

- a) mv
- b) cp

## WEEK 12

Write a c program that simulates ls command

(Use system calls /directory API)



## BASIC UNIX COMMANDS

Command	CAT
Syntax	cat [argument] [specific file]
Description	“cat” is short for concatenate. This command is used to create, view and concatenate files.
Examples	<p>cat /etc/passwd</p> <p>This command displays the "/etc/passwd" file on your screen.</p> <p>cat /etc/profile</p> <p>This command displays the "/etc/profile" file on your screen. Notice that some of the contents of this file may scroll off of your screen.</p> <p>cat file1 file2 file3 &gt; file4</p> <p>This command combines the contents of the first three files into the fourth file.</p>
Command	pwd
Syntax	pwd
Description	"pwd" stands for print working directory. It displays your current position in the UNIX filesystem.
Examples	<p>pwd</p> <p>There are no options (or arguments) with the "pwd" command. It is simply used to report your current working directory.</p>
Command	ls
Syntax	ls [options] [names]
Description	"ls" stands for list. It is used to list information about files and directories.
Examples	<p>ls</p> <p>This is the basic "ls" command, with no options. It provides a very basic listing of the files in your current working</p>

	<p>directory. Filenames beginning with a decimal are considered <i>hidden</i> files, and they are not shown.</p> <p><code>ls -a</code></p> <p>The <code>-a</code> option tells the <code>ls</code> command to report information about all files, including hidden files.</p> <p><code>ls -l</code></p> <p>The <code>-l</code> option tells the "<code>ls</code>" command to provide a <i>long</i> listing of information about the files and directories it reports. The long listing will provide important information about file permissions, user and group ownership, file size, and creation date.</p> <p><code>ls -al</code></p> <p>This command provides a <i>long</i> listing of information about <i>all</i> files in the current directory. It combines the functionality of the <code>-a</code> and <code>-l</code> options. <i>This is probably the most used version of the <code>ls</code> command.</i></p> <p><code>ls -al /usr</code></p> <p>This command lists long information about all files in the <code>/usr</code> directory.</p> <p><code>ls -alR /usr   more</code></p> <p>This command lists long information about all files in the <code>/usr</code> directory, and all sub-directories of <code>/usr</code>. The <code>-R</code> option tells the <code>ls</code> command to provide a <i>recursive</i> listing of all files and sub-directories.</p> <p><code>ls -ld /usr</code></p> <p>Rather than list the files contained in the <code>/usr</code> directory, this command lists information about the <code>/usr</code> directory itself (without generating a listing of the contents of <code>/usr</code>). This is very useful when you want to check the permissions of the directory, and not the files the directory contains.</p>
Command	<code>mv</code>
Syntax	<code>mv [options] sources target</code>

Options	<p>-b backup files that are about to be overwritten or removed</p> <p>-i interactive mode; if dest exists, you'll be asked whether to overwrite the file</p>
Description	The "mv" command is used to move and rename files.
Examples	<p>mv Chapter1 Chapter1.bad</p> <p>This command renames the file "Chapter1" to the new name "Chapter1.bad".</p> <p>mv Chapter1 garbage</p> <p>This command renames the file "Chapter1" to the new name "garbage". (Notice that if "garbage" is a directory, "Chapter1" would be moved into that directory).</p> <p>mv Chapter1 /tmp</p> <p>This command moves the file "Chapter1" into the directory named "/tmp".</p> <p>mv tmp tmp.old</p> <p>Assuming in this case that tmp is a directory, this example renames the directory tmp to the new name tmp.old.</p>
Command	rm
Syntax	rm [options] files
Options	<p>-d, --directory unlink FILE, even if it is a non-empty directory (super-user only)</p> <p>-f, --force ignore nonexistent files, never prompt</p> <p>-i, --interactive prompt before any removal</p> <p>-r, -R, --recursive remove the contents of directories recursively</p> <p>-v, --verbose explain what is being done</p>

Description	The "rm" command is used to remove files and directories. (Warning - be very careful when removing files and directories!)
Examples	<p>rm Chapter1.bad</p> <p>This command deletes the file named "Chapter1.bad" (assuming you have permission to delete this file).</p> <p>rm Chapter1 Chapter2 Chapter3</p> <p>This command deletes the files named "Chapter1", "Chapter2", and "Chapter3".</p> <p>rm -i Chapter1 Chapter2 Chapter3</p> <p>This command prompts you before deleting any of the three files specified. The -i option stands for <i>inquire</i>. You must answer y (for yes) for each file you really want to delete. This can be a safer way to delete files.</p> <p>rm *.html</p> <p>This command deletes all files in the current directory whose filename ends with the characters ".html".</p> <p>rm index*</p> <p>This command deletes all files in the current directory whose filename begins with the characters "index".</p> <p>rm -r new-novel</p> <p>This command deletes the directory named "new-novel". This directory, and all of its' contents, are erased from the disk, including any sub-directories and files.</p>
Command	cp
Syntax	<p>cp [options] file1 file2</p> <p>cp [options] files directory</p>
Options	-b      backup files that are about to be overwritten or removed

	<p>-i interactive mode; if dest exists, you'll be asked whether to overwrite the file</p> <p>-p preserves the original file's ownership, group, permissions, and timestamp</p>
Description	<p>The "cp" command is used to copy files and directories.</p> <p>Note that when using the cp command, you must always specify both the source and destination of the file(s) to be copied.</p>
Examples	<p>cp .profile .profile.bak</p> <p>This command copies your ".profile" to a file named ".profile.bak".</p> <p>cp /usr/fred/Chapter1 .</p> <p>This command copies the file named "Chapter1" in the "/usr/fred" directory to the current directory. This example assumes that you have write permission in the current directory.</p> <p>cp /usr/fred/Chapter1 /usr/mary</p> <p>This command copies the "Chapter1" file in "/usr/fred" to the directory named "/usr/mary". This example assumes that you have write permission in the "/usr/mary" directory.</p>
Command	grep
Syntax	grep [options] regular expression [files]
Options	<p>-i case-insensitive search</p> <p>-n show the line# along with the matched line</p> <p>-v invert match, e.g. find all lines that do NOT match</p> <p>-w match entire words, rather than substrings</p>
Description	<p>Think of the "grep" command as a "search" command (most people wish it was named "search"). It is used to search for text strings within one or more files.</p>

Examples	<pre>grep 'fred' /etc/passwd</pre> <p>This command searches for all occurrences of the text string 'fred' within the "/etc/passwd" file. It will find and print (on the screen) all of the lines in this file that contain the text string 'fred', including lines that contain usernames like "fred" - and also "alfred".</p> <pre>grep '^fred' /etc/passwd</pre> <p>This command searches for all occurrences of the text string 'fred' within the "/etc/passwd" file, but also requires that the "f" in the name "fred" be in the first column of each record (that's what the caret character tells grep). Using this more-advanced search, a user named "alfred" would not be matched, because the letter "a" will be in the first column.</p> <pre>grep 'joe' *</pre> <p>This command searches for all occurrences of the text string 'joe' within all files of the current directory.</p>
Command	<code>mkdir</code>
Syntax	<code>mkdir [options] directory name</code>
Description	The "mkdir" command is used to create new directories (sub-directories).
Examples	<pre>mkdir tmp</pre> <p>This command creates a new directory named "tmp" in your current directory. (This example assumes that you have the proper permissions to create a new sub-directory in your current working directory.)</p> <pre>mkdir memos letters e-mail</pre> <p>This command creates three new sub-directories (memos, letters, and e-mail) in the current directory.</p> <pre>mkdir /usr/fred/tmp</pre> <p>This command creates a new directory named "tmp" in the directory "/usr/fred". "tmp" is now a sub-directory of "/usr/fred". (This example assumes that you have the proper</p>

	<p>permissions to create a new directory in /usr/fred.)</p> <pre>mkdir -p /home/joe/customer/acme</pre> <p>This command creates a new directory named /home/joe/customer/acme, and creates any intermediate directories that are needed. If only /home/joe existed to begin with, then the directory "customer" is created, and the directory "acme" is created inside of customer.</p>
Command	rmdir
Syntax	rmdir [options] directories
Description	The "rm" command is used to remove files and directories. (Warning - be very careful when removing files and directories!)
Examples	<pre>rm Chapter1.bad</pre> <p>This command deletes the file named "Chapter1.bad" (assuming you have permission to delete this file).</p> <pre>rm Chapter1 Chapter2 Chapter3</pre> <p>This command deletes the files named "Chapter1", "Chapter2", and "Chapter3".</p> <pre>rm -i Chapter1 Chapter2 Chapter3</pre> <p>This command prompts you before deleting any of the three files specified. The -i option stands for <i>inquire</i>. You must answer y (for yes) for each file you really want to delete. This can be a safer way to delete files.</p> <pre>rm *.html</pre> <p>This command deletes all files in the current directory whose filename ends with the characters ".html".</p> <pre>rm index*</pre> <p>This command deletes all files in the current directory whose filename begins with the characters "index".</p> <pre>rm -r new-novel</pre> <p>This command deletes the directory named "new-novel". This directory, and all of its' contents, are erased from the</p>

	disk, including any sub-directories and files.
Command	cd, chdir
Syntax	cd [name of directory you want to move to]
Description	"cd" stands for change directory. It is the primary command for moving around the filesystem.
Examples	<p>cd /usr</p> <p>This command moves you to the "/usr" directory. "/usr" becomes your current working directory.</p> <p>cd /usr/fred</p> <p>Moves you to the "/usr/fred" directory.</p> <p>cd /u*/f*</p> <p>Moves you to the "/usr/fred" directory - if this is the only directory matching this wildcard pattern.</p> <p>cd</p> <p>Issuing the "cd" command without any arguments moves you to your <i>home</i> directory.</p> <p>cd -</p> <p>Using the Korn shell, this command moves you back to your previous working directory. This is very useful when you're in the middle of a project, and keep moving back-and-forth between two directories.</p>
Command	kill
Syntax	kill [options] IDs
Description	kill ends one or more process IDs. In order to do this you must own the process or be designated a privileged user. To find the process ID of a certain job use <a href="#">ps</a> .
Examples	
Command	ps



Syntax	ps [options]
Description	The "ps" command (process statistics) lets you check the status of processes that are running on your Unix system.
Examples	<p>ps</p> <p>The ps command by itself shows minimal information about the processes <i>you</i> are running. Without any arguments, this command will not show information about other processes running on the system.</p> <p>ps -f</p> <p>The -f argument tells ps to supply <i>full</i> information about the processes it displays. In this example, ps displays full information about the processes <i>you</i> are running.</p> <p>ps -e</p> <p>The -e argument tells the ps command to show <i>every</i> process running on the system.</p> <p>ps -ef</p> <p>The -e and -f arguments are normally combined like this to show full information about every process running on the system. <i>This is probably the most often-used form of the ps command.</i></p> <p>ps -ef   more</p> <p>Because the output normally scrolls off the screen, the output of the ps -ef command is often piped into the more command. The more command lets you view one screenful of information at a time.</p> <p>ps -fu fred</p> <p>This command shows full information about the processes currently being run by the user named <i>fred</i> (the -u option lets you specify a username).</p>

## BASIC UNIX COMMAND EXERCISES

---


1. Verify that you are in your home directory.
2. Make the directory *adir* using the following command:  

```
mkdir adir
```
3. List the files in the current directory to verify that the directory *adir* has been made correctly.
4. Change directories to *adir*.
5. Verify that you have succeeded in moving to the *adir* directory.
6. Verify that the file *testfile* exists.
7. List the contents of the file *testfile* to the screen.
8. Make a copy of the file *testfile* under the name *secondfile*.
9. Verify that the files *testfile* and *secondfile* both exist.
10. List the contents of both *testfile* and *secondfile* to the monitor screen.
11. Delete the file *testfile*.
12. Verify that *testfile* has been deleted.
13. Clear the window.
14. Use the "bang" command to re-run the command that verified that *testfile* has been deleted.
15. Rename *secondfile* to *thefile*.
16. Issue the command to find out how large *thefile* is. How big is it?
17. Copy *thefile* to your home directory.
18. Remove *thefile* from the current directory.
19. Verify that *thefile* has been removed.
20. Copy *thefile* from your home directory to the current directory.
21. Verify that *thefile* has been copied from your home directory to the current directory.

22. Change directories to your home directory.
23. Verify that you are in your home directory.
24. Verify that a copy of *thefile* is in your home directory.
25. Remove *thefile* from your home directory.
26. Remove *thefile* from the directory *adir*.
27. Remove the directory *adir* from your home directory with the following command.  

```
rm -r adir
```
28. Verify that *thefile* and *adir* are gone from your home directory.



- 
- The logo of SCET (Sri Chaitanya Engineering & Technology) is a circular emblem. At the top, the word "KNOWLEDGE" is written in an arc. In the center, there is a computer monitor and keyboard. Below this, the words "VISION" and "CONDUCT" are written on banners, separated by the word "or". The bottom half of the emblem features a rising sun with rays, flanked by a radio tower on the left and a satellite dish on the right. There are five stars around the perimeter of the circle. At the very bottom, a banner displays the acronym "SCET" in large, bold letters, with two gear-like symbols on either side.

23. pwd

24. ls

25. rm thefile

26. rm adir/thefile **or** rm /home/*your-clid*/adir/thefile

27. rmdir adir **or** rmdir /home/*your-clid*/adir

28. ls



WEEK1

---

## Session 1

1. Log in to the system
2. Use Vi editor to create a file called myfile.txt which contain some text.
3. Correct typing errors during creation
4. Save the file
5. Logout of the file

Sol:

```
$ login: <user name>
```

```
$ password: *****
```

```
$ vi
```

```
~ Unix is Case Sensitive
```

```
~ Never leave the Computer without logging out when you are working in  
a time sharing or network environments.
```

```
Type <Esc>
```

```
: wq myfile
```

```
$
```

## Session 2

1. Log into the system
2. Open the file created in session 1
3. Add some text
4. Change some text
5. delete some text
6. Save the changes
7. Logout of the system

Sol:

```
$ login: <user name>
```

```
$ password: *****
```

```
$ vi myfile
```

```
~ Unix is Case Sensitive
```

```
~ Never leave the Computer without logging out when you are working in  
a time sharing or network environments.
```

```
~ Shell Programming
```

```
: wq
```

## WEEK2

Log into the system

Use the cat command to create a file containing the following data. Call it mytable use tabs to separate the fields

1425	ravi	15.65
4320	ramu	26.27
6830	sita	36.15
1450	raju	21.86

- use the cat command to display the file, my table
- use the vi command to correct any errors in the file, my table
- use the sort command to sort the file my table according to the first field. Call the sorted file my table(same name)
- print the file my table
- use the cut & paste commands to swap fields 2 and 3 my table. Call it mytable(same name)
- print the new file, my table
- logout of the system

Sol:

\$ login: <user name>

\$ password:\*\*\*\*\*

a) Creating the file mytable. Use tabs to separate the fields.

```
$ cat > mytable
1425      Ravi  15.65
4320      Ram  26.27
6830      Raj  36.15
1450      Sita 21.86
^d
```

b) Displaying the file mytable contents:                      \$ cat mytable

c) Sorting the file mytable:                                      \$ sort mytable

d) Swaping the fields 2 and 3

```
$ cut -f1 mytable >file1
$ cut -f2 mytable >file2
$ cut -f3 mytable >file3
$ cut -f4 mytable >file4
$ paste file1 file3 file2 file4 >mytable
$ cat mytable
```

f) Printing the file mytable:                                      \$ lpr mytable

WEEK3

---

- a. log in the system
- b. use the appropriate commands to determine ur login shell
- c. use the /etc/passwd file to verify the result of step b.
- d. use the who command redirect the result to a file called myfile1. Use the more command to see the contents of myfile1.
- e. Use the date and who commands in sequence ?(in one line) such that the output of date will display on the screen and the output of who will be redirected to a file called my file2. Use the more command to check the contents of myfile2.
- f. write a sed command that deletes the first character in each line in a file
- g. write a sed command that deletes the character before the last character in each line in a file.
- h. Write a sed command that swaps the files and second words in each line in a file

Sol:

- a. *\$echo "the login shell is \$SHELL"*
- b. *\$echo "the login shell is \$0"*
- c. *\$date ; who > myfile2*
- d. *\$more myfile2*
- e. *Delete second character: sed "s/^(.).\1/" file1*
- f. *Delete character before last character: sed "s/^(.)\$\1/" file1*
- g. *Delete word before last word: sed "s/\*[^ ]\*( \*[^ ]\*)\$\1/" file1*



WEEK4

---

pipe ur /etc/passwd file to awk and print out the home directory of each user.  
Develop an interactive grep script that asks for a word and a file name and then tells how many lines contain that word

Repeat

Part using awk

Solutions:

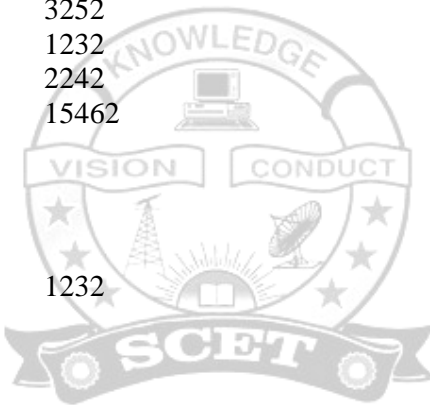
- a. `$ grep -c word filename`
- b. `awk ' $2 == "Computers" && $3 > 10000 {print}' Sales.dat`

I/P:

1	Clothing	3141
1	Computers	9161
1	Textbooks	21312
2	Clothing	3252
2	Computers	1232
2	Supplies	2242
2	Text books	15462

O/P:

2	Computers	1232
---	-----------	------



## GREP EXERCISE

Create the file `grepdata.txt` with the following data and then write a series of `grep` statements that do the following:

```
Sep. 17, 2007
Esperanza High School
1830 N. Kellog Dr.
Anaheim, CA 92807-1281
Steve Marshal
714-555-7870 x7310
aztecwrestling@example.com
Brian Fortenbaugh
714-555-7870 x7309

Sep. 24, 2007
Sonora High School
401 S. Palm St.
La Habra, CA 90631
Carl Hohl (aka Krazy Rabbit)
562-555-9800

Oct. 1, 2007
Lakewood High School
440 Briercrest Ave.
Lakewood, CA 90713-9112
Andy Miramontes
562-555-1281

Oct. 8, 2007
North Torrance High School
2007 W. 182nd

Torrance, CA 90504
Don Henderson
310-555-4412

Oct. 15, 2007
El Dorado High School
1651 N. Valencia Ave.
Placentia, CA 90631
Steve Lawson
714-555-5350 x1220
Lawsonhawk@example.com

Nov. 5, 2007
El Dorado High School
1651 N. Valencia Ave.
Placentia, CA 90631
Steve Lawson
714-555-5350 x1220
Lawsonhawk@example.com

Nov. 12, 2007
Rosemead High School
9063 E. Sepulveda Dr.
Rosemead, CA 91770
Daren de Heras
daren103@example.com
```

- Print all lines that contain a phone number with an extension (the letter `x` or `X` followed by four digits).
- Print all lines that begin with three digits followed by a blank. Your answer *must* use the `\{` and `\}` repetition specifier.
- Print all lines that contain a date. Hint: this is a *very* simple pattern. It does not have to work for any year before 2000.
- Print all lines containing a vowel (a, e, i, o, or u) followed by a single character followed by the same vowel again. Thus, it will find “eve” or “adam” but not “vera”. Hint: `\(` and `\)`
- Print all lines that do not begin with a capital S.

Write `grep` statements that use command-line options along with the pattern to do the following:

- Print all lines that contain CA in either uppercase or lowercase.
- Print all lines that contain an email address (they have an `@` in them), preceded by the line number.
- Print all lines that do *not* contain the word `Sep.` (including the period).
- Print all lines that contain the word `de` as a whole word.

## SED EXERCISE

Create a file with the name docbook with following data.

```
<article>
<title>About the Web</title>

<para>
This is an article about the World Wide Web.
The World Wide Web is a collection of documents that are linked to
one another. The Web is <emphasis>not</emphasis> the same as the
Internet. The Internet is a world-wide network of networks, and it
does far more than simply serve up Web pages.
</para>

<para>Tim Berners-Lee, the inventor of the World Wide Web, put
special
emphasis on the portability of web pages. Rather than create a
proprietary format, he made Web pages dependent only upon plain ASCII
text.</para>

<para>
Web pages are written in a markup language called HTML. Here is what
it
looks like. The &lt; and &gt; mark off elements.
</para>

<listing>
<lt;body>
<div id="top-navig">
<a id="top"></a>
<a href="index.html">CIT 040 Index</a>
&gt;
Assignment 1
</div>

<h1>Assignment 1</h1>
<p>This exercise shows you how to use the two computer
environments that
you will use in this class. You will:</p>
<ol class="upper-roman">
<li>Set up your directories on Windows. This is
where you will write your HTML documents.</li>
</ol>
</listing>

<para>It looks difficult, but it is possible to learn HTML in a few
weeks. <emphasis>You, too can create web pages for viewing by
friends and family!</emphasis>
Note that, in our listing, we had to encode &gt; as &amp;gt;.
</para>
</article>
```

Write a sed file that does the following.

1. Lines with `<article>` and `</article>` should be deleted.
2. Replace `<title>` with `Title:`, and replace `</title>` with nothing.
3. Replace all `<para>` and `</para>` tags with the null string. If the resulting line is empty, delete the line. (You may need to use curly braces to make this happen.)
4. Replace all `<emphasis>` and `</emphasis>` tags with asterisks. Thus:

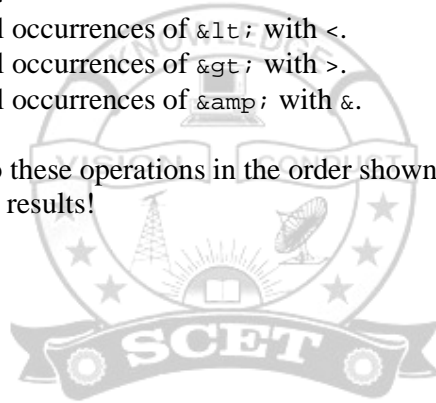
This is a `<emphasis>great</emphasis>` bargain.

will become

This is a `*great*` bargain.

5. Replace the word `web` with `web everywhere`.
6. Replace lines starting with `<listing>` by `---begin listing`
7. Replace lines starting with `</listing>` by `---end listing`
8. Between the `<listing>` and `</listing>`, do these things (you *must* use curly braces to do this!):
  - o Replace all occurrences of `&lt;` with `<`.
  - o Replace all occurrences of `&gt;` with `>`.
  - o Replace all occurrences of `&amp;` with `&`.

Note: you must do these operations in the order shown above; otherwise, you will get the wrong results!



## AWK EXERCISE

---

1.

Save the following data in a file named teamlist.txt:

```
Tom,Red,5,17,22
Joe,Green,3,14,22
Maria,Blue,6,18,21
Fred,Blue,2,15,23
Carlos,Red,-1,15,24
Phuong,Green,7,19,21
Enrique,Green,3,16,20
Nancy,Red,9,12,24
```

Write an `awk` script that will compute the average score for every person in the list, the average score for each test, and the average score for each team. If a score is negative, that means the person missed the test, and the score must *not* become part of the average.

2.

Create a file with following data.

```
Mike Harrington:(510) 548-1278:250:100:175
Christian Dobbins:(408) 538-2358:155:90:201
Susan Dalsass:(206) 654-6279:250:60:50
Archie McNichol:(206) 548-1348:250:100:175
Jody Savage:(206) 548-1278:15:188:150
Guy Quigley:(916) 343-6410:250:100:175
Dan Savage:(406) 298-7744:450:300:275
Nancy McNeil:(206) 548-1278:250:80:75
John Goldenrod:(916) 348-4278:250:100:175
Chet Main:(510) 548-5258:50:95:135
Tom Savage:(408) 926-3456:250:168:200
Elizabeth Stachelin:(916) 440-1763:175:75:300
```

- a. Print the names of those who contributed between \$75 and **\$200** in the first month.
- b. Print the names and **phone numbers** of those with an average monthly contribution greater than \$200.
- c. Add \$10 to Chet's second contribution **and print the changed record**.
- d. Change Nancy McNeil's name to Louise McInnes **and print the changed record**.

WEEK5

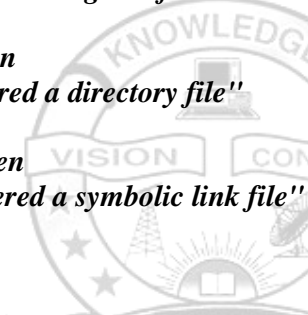
---

- a) Write A shell script that takes a command –line argument and reports on whether it is directory ,a file,or something else
- b) Write a shell script that accepts one or more file name as a arguments and converts all of them to uppercase,provided they exists in the current directory
- c) Write a shell script that determines the period for which a specified user is working on the system

Sol:

a.

```
#!/bin/csh
if( -f "$argv[1]") then
    echo "you have entered a regular file name"
else
    if( -d "$argv[1]") then
        echo "you have entered a directory file"
    else
        if( -l "$argv[1]") then
            echo "you have entered a symbolic link file"
        endif
    endif
endif
```



b.

```
#!/bin/csh
set c = 1
set x = $#argv
while( $x != 0)
tr '[a-z]' '[A-Z]' <$argv[$c]
@ c = $c + 1
@ x = $x - 1
End
```

WEEK6

---

- (a) Write a shell script that accepts a file name starting and ending line numbers as arguments and displays all the lines between the given line numbers
- (b) Write a shell script that deletes all lines containing a specified word I one or more files supplied as arguments to it.

**Sol:**

**a.**

```
#!/bin/csh  
head -$argv[3] $argv[1] | tail -n $argv[2]
```

**b.**

```
#!/bin/csh  
grep -v $argv[1] $argv[2]
```



WEEK7

---

a) Write a shell script that computes the gross salary of a employee according to the following

- 1) if basic salary is <1500 then HRA 10% of the basic and DA =90% of the basic
  - 2) if basic salary is >1500 then HRA 500 and DA =98% of the basic
- The basic salary is entered interactively through the key board

(b)Write a shell script that accepts two integers as its arguments and computes the value of first number raised to the power of the second number

a.

```
print Enter Basic salary
read bs
if((bs < 1500 ))
then
    hra=$(( bs * 0.1 ))
    print hra=$hra
    da=$(( bs * 0.9 ))
    print da=$da
else
    hra=500
    print hra=$hra
    da=$(( bs * 0.98 ))
    print da=$da
fi
gs=$(( bs + hra + da ))
print Gross salary=Rs $gs
```



b.

```
#!/bin/csh
set c = 1
set result = 1
while($c <= $argv[2])
    @ result *= $argv[1]
    @ c = $c + 1
end
echo "$argv[1] raised to the power of $argv[2]=$result"
```



## WEEK 8

- (a) Write an interactive file handling shell program. Let it offer the user the choice of copying ,removing ,renaming or linking files. Once the use has made a choice, have the program ask the user for necessary information, such as the file name ,new name and so on.
- (b) Write a shell script that takes a login name as command –line argument and reports when that person logs in
- (c) Write a shell script which receives two files names as arguments. It should check whether the two file contents are same or not. If they are same then second file should be deleted.

**Sol:**

**a.**

```
#!/bin/csh
echo "MENU"
echo "1.FILE COPYING"
echo "2.FILE RENAMING"
echo "3.FILE REMOVING"
echo "4.FILE LINKING"
echo "ENTER YOUR CHOICE"
set choice = $<
switch ($choice)
case 1:
    echo "enter the source file name"
    set source = $<
    echo "enter the target file name"
    set target = $<
    if( -f $source ) then
        cp $source $target
        echo "file copied successfully"
    else
        echo "source file doesnot exist"
    endif
    breaksw
case 2:
    echo "enter the source file to rename"
    set source = $<
    echo "enter the new name for the source file"
    set target = $<
    mv $source $target
    echo "file renamed suceessfully"
    breaksw
case 3:
    echo "enter the source file to remove"
    set source = $<
```

```
rm $source
echo "file removed successfully"
breaksw
case 4:
echo "enter the source file to provide link"
set source = $<
echo "enter the link name for the source file"
set target = $<
link $source $target
echo "file linked successfully"
breaksw
echo "file linked successfully"
breaksw
deefault:
echo " you entered wrong choice"

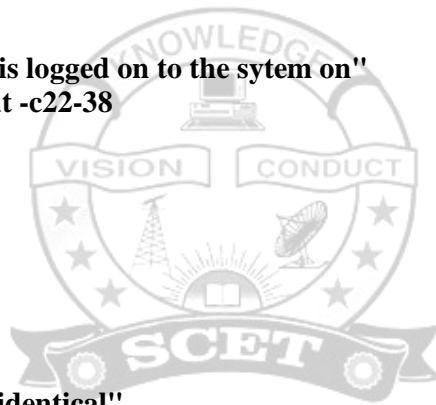
endsw
```

b.

```
#!/bin/csh
echo "the user $argv[1] is logged on to the sytem on"
who|grep "$argv[1]"| cut -c22-38
```

c.

```
#!/bin/ksh
if(cmp $1 $2)
then
print "the two files are identical"
rm $2
print "Now $2 is deleted"
else
print "the two files are different"
fi
```



## WEEK 9

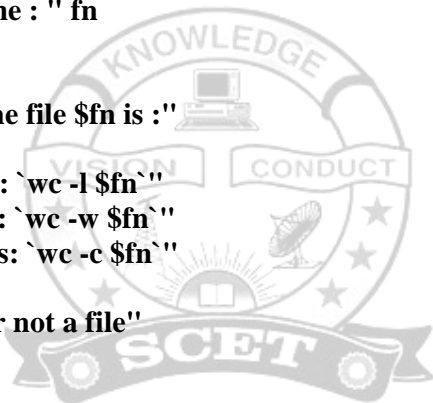
- (a) Write a shell script that displays a list of all files in the current directory to which the user has read write and execute permissions
- (b) Develop an interactive script that asks for a word and file name and then tells how many times that word occurred in the file.
- (c) Write a shell script to perform the following string operations.
  - 1) To extract a sub string from a given string
  - 2) To find the length of a given string

## (a) PROGRAM

```
#!/bin/csh
ls -l|grep '^.rwx'
```

## (b) PROGRAM

```
#!/bin/bash
read -p "Enter a file name : " fn
if test -f $fn
then
echo "The contents of the file $fn is : "
cat $fn
echo "No. of Line      : `wc -l $fn`"
echo "No. of Words     : `wc -w $fn`"
echo "No. of Characters: `wc -c $fn`"
else
echo "$fn is not exists or not a file"
fi
```



## (c) PROGRAM

```
#!/bin/ksh
print "MENU"
print "1.TO EXTRACT A SUBSTRING FROM AGIVEN STRING"
print "2.TO FIND THE LENGTH OF THE STRING"
print "ENTER YOUR CHOICE"
read choice
print "enter a string"
read string
case $choice in
1) print "enter position1"
   read pos1
   print "enter position2"
   read pos2
   print "the extracted substring from $pos1 to $pos2 of $string is"
   print $string|cut -c$pos1-$pos2;;
2) print the length of the string $string is
   c=$(print $string|wc -c)
```

```
((ch=c-1))
print $ch number of characters;;
*) print " wrong choice ";;
esac
```

## ADDITIONAL SHELL PROGRAMMING EXERCISES

---

1. Write a simple shell script that takes any number of arguments on the command line, and prints the arguments with “Hello ” in front. For example, if the name of the script is hello, then you should be able to run it like this:

```
$ hello Nick Urbanik
Hello Nick Urbanik
$ hello Edmund
Hello Edmund
```

2. Write a simple shell script that takes two numbers as parameters and uses a while loop to print all the numbers from the first to the second inclusive, each number separated only by a space from the previous number. Example, if the script is called jot, then

```
$ jot 2 8
2 3 4 5 6 7 8
```

3. Write a script which displays “Good morning”, “Good afternoon” or “Good evening”, on the monitor, depending on the time of running the script.
4. Write a script which reads a number in units of seconds and converts it to the units hours:minutes:seconds and prints the result to standard output. Your script must prompt for re-input if a negative value is input

Enter number of seconds: 12345

Result:

12345 seconds in hours:minutes:seconds is 3:25:45

5. Suppose that the script you wrote for question 2 is called jot. Then run it calling sh yourself. Notice the difference:

```
sh jot 2 5
sh -v jot 2 5
sh -x jot 2 5
```

Do you notice any difference in the output from last two?

6. Write a script calculate, which accepts 4 arguments a, b, c, d and prints the value of  $a \times 20 - b \times 2 + c \div d$  to standard output.

```
$ calculate 2 12 5 2
```

The value of "2\*20 - 12\*2 + 5/2" is 18

## WEEK 10

Write a C program that takes one or more file or directory names as command line input and reports the following information on the file.

1. file type
2. number of links
3. read, write and execute permissions
4. time of last access

(Note: use /fstat system calls)

```
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/mkdev.h>
#include <stdio.h>

char *typeOfFile(mode_t);
char *permOfFile(mode_t);
void outputStatInfo(char *, struct stat *);

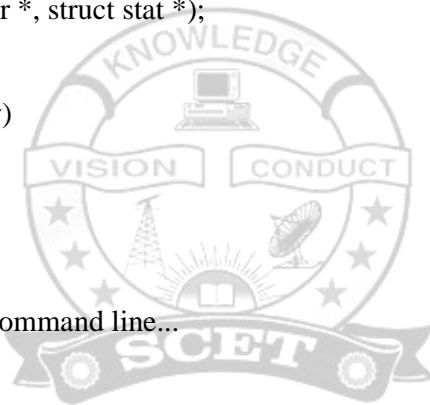
int
main(int argc, char **argv)
{
    char *filename;
    struct stat st;

    /*
     * For each file on the command line...
     */
    while (--argc) {
        filename = *++argv;

        /*
         * Find out about it.
         */
        if (lstat(filename, &st) < 0) {
            perror(filename);
            putchar('\n');
            continue;
        }

        /*
         * Print out the information.
         */
        outputStatInfo(filename, &st);
        putchar('\n');
    }

    exit(0);
}
```



```
}

/*
 * outputStatInfo - print out the contents of the stat structure.
 */
void
outputStatInfo(char *filename, struct stat *st)
{
    printf("File Name:      %s\n", filename);
    printf("File Type:      %s\n", typeOfFile(st->st_mode));

    /*
     * If the file is not a device, print its size and optimal
     * i/o unit; otherwise print its major and minor device
     * numbers.
     */
    if (((st->st_mode & S_IFMT) != S_IFCHR) &&
        ((st->st_mode & S_IFMT) != S_IFBLK)) {
        printf("File Size:      %d bytes, %d blocks\n", st->st_size,
            st->st_blocks);
        printf("Optimum I/O Unit:  %d bytes\n", st->st_blksize);
    }
    else {
        printf("Device Numbers:  Major: %u  Minor: %u\n",
            major(st->st_rdev), minor(st->st_rdev));
    }

    /*
     * Print the permission bits in both "ls" format and
     * octal.
     */
    printf("Permission Bits:  %s (%04o)\n", permOfFile(st->st_mode),
        st->st_mode & 0777);

    printf("Inode Number:      %u\n", st->st_ino);
    printf("Owner User-Id:      %d\n", st->st_uid);
    printf("Owner Group-Id:     %d\n", st->st_gid);
    printf("Link Count:         %d\n", st->st_nlink);

    /*
     * Print the major and minor device numbers of the
     * file system that contains the file.
     */
    printf("File System Device: Major: %u  Minor: %u\n",
        major(st->st_dev), minor(st->st_dev));

    /*
     * Print the access, modification, and change times.
     * The ctime() function converts the time to a human-
     * readable format; it is described in Chapter 7,
```

```

    * "Time of Day Operations."
    */
    printf("Last Access:      %s", ctime(&st->st_atime));
    printf("Last Modification: %s", ctime(&st->st_mtime));
    printf("Last I-Node Change: %s", ctime(&st->st_ctime));
}

/*
 * typeOfFile - return the english description of the file type.
 */
char *
typeOfFile(mode_t mode)
{
    switch (mode & S_IFMT) {
    case S_IFREG:
        return("regular file");
    case S_IFDIR:
        return("directory");
    case S_IFCHR:
        return("character-special device");
    case S_IFBLK:
        return("block-special device");
    case S_IFLNK:
        return("symbolic link");
    case S_FIFO:
        return("FIFO");
    case S_IFSOCK:
        return("UNIX-domain socket");
    }

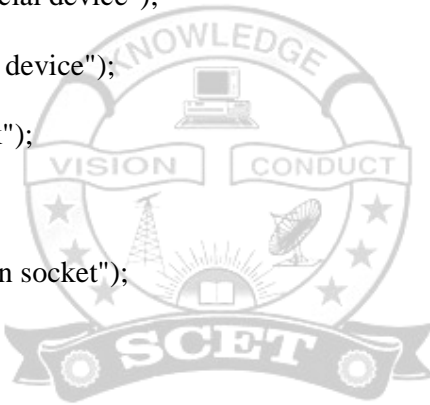
    return("???");
}

/*
 * permOfFile - return the file permissions in an "ls"-like string.
 */
char *
permOfFile(mode_t mode)
{
    int i;
    char *p;
    static char perms[10];

    p = perms;
    strcpy(p, "-----");

    /*
     * The permission bits are three sets of three
     * bits: user read/write/exec, group read/write/exec,
     * other read/write/exec. We deal with each set

```



```
* of three bits in one pass through the loop.
*/
for (i=0; i < 3; i++) {
    if (mode & (S_IREAD >> i*3))
        *p = 'r';
    p++;

    if (mode & (S_IWRITE >> i*3))
        *p = 'w';
    p++;

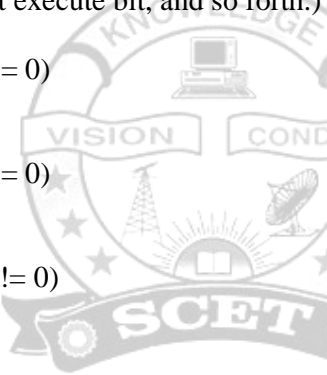
    if (mode & (S_IEXEC >> i*3))
        *p = 'x';
    p++;
}

/*
 * Put special codes in for set-user-id, set-group-id,
 * and the sticky bit. (This part is incomplete; "ls"
 * uses some other letters as well for cases such as
 * set-user-id bit without execute bit, and so forth.)
 */
if ((mode & S_ISUID) != 0)
    perms[2] = 's';

if ((mode & S_ISGID) != 0)
    perms[5] = 's';

if ((mode & S_ISVTX) != 0)
    perms[8] = 't';

return(perms);
}
```

The logo of SCET (Sri Chaitanya Engineering & Technology) is a circular emblem. At the top, the word 'KNOWLEDGE' is written in an arc. Inside the circle, there is a computer monitor, a satellite dish, and a book. Below these symbols, the words 'VISION' and 'CONDUCT' are written on banners. At the bottom of the circle, the letters 'SCET' are prominently displayed in a stylized font, with a banner-like background.



## WEEK 11

Write C program that simulate the following unix commands

- (a) mv
- (b) cp

a.

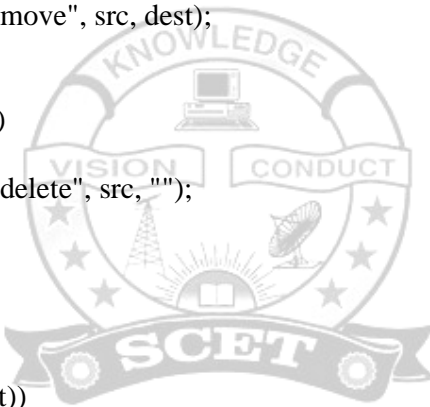
```
int mv(char *src, char *dest)
{
    int errcount = 0;
    char buf[FILENAME_MAX];
    const char *generr = "ERROR: mv - couldn't %s %s %s\n";

    if ('.' == dest[1] && *dest != *getcwd(buf, FILENAME_MAX))
    {
        if (file_copy(src, dest))
        {
            printf(generr, "move", src, dest);
            ++errcount;
        }
        else if (unlink(src))
        {
            printf(generr, "delete", src, "");
            ++errcount;
        }
    }
    else
    {
        if (rename(src, dest))
        {
            printf(generr, "rename", src, dest);
            ++errcount;
        }
    }
    return errcount;
}

/*
** Enter here
*/

int main(int argc, char **argv)
{
    int src, errcount = 0;
    char target[FILENAME_MAX];

    puts("mv 1.3 (4 jun 93) - Ray L. McVay/Bob Stout");
```



```

if (argc < 3)
    help("Not enough parameters");

/*
** Handle cases where target is a directory
*/

else if (isdir(argv[argc - 1]))
{
    for (src = 1; src < argc - 1; src++)
    {
        char termch;

        strcpy(target, argv[argc - 1]);
        termch = target[strlen(target) - 1];
        if ('\ ' != termch && ':' != termch)
            strcat(target, "\ ");

        if (strchr(argv[src], '\ '))
            strcat(target, strchr(argv[src], '\ ') + 1);
        else if (argv[src][1] == ':')
            strcat(target, argv[src] + 2);
        else strcat(target, argv[src]);

        errcount += mv(argv[src], target);
    }
}

/*
** Nothing left except 2 explicit file names
*/

else if (argc == 3)
    errcount += mv(argv[1], argv[2]);

return errcount;
}

```

**b.**

```

#define BSIZE 512
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
struct stat  stbuf1, stbuf2;
char  iobuf[BSIZE];

main(argc, argv)
char *argv[];

```

```

{
    register i, r;

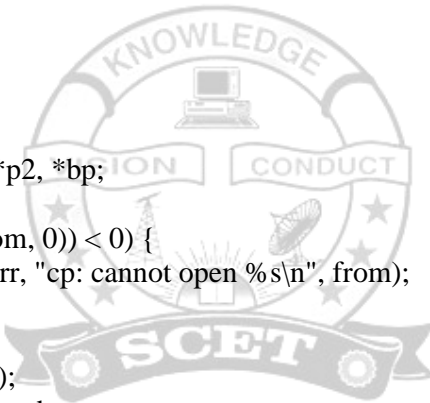
    if (argc < 3)
        goto usage;
    if (argc > 3) {
        if (stat(argv[argc-1], &stbuf2) < 0)
            goto usage;
        if ((stbuf2.st_mode & S_IFMT) != S_IFDIR)
            goto usage;
    }
    r = 0;
    for(i=1; i<argc-1; i++)
        r |= copy(argv[i], argv[argc-1]);
    exit(r);
usage:
    fprintf(stderr, "Usage: cp: f1 f2; or cp f1 ... fn d2\n");
    exit(1);
}

```

```

copy(from, to)
char *from, *to;
{
    int fold, fnew, n;
    register char *p1, *p2, *bp;
    int mode;
    if ((fold = open(from, 0)) < 0) {
        fprintf(stderr, "cp: cannot open %s\n", from);
        return(1);
    }
    fstat(fold, &stbuf1);
    mode = stbuf1.st_mode;
    /* is target a directory? */
    if (stat(to, &stbuf2) >= 0 &&
        (stbuf2.st_mode & S_IFMT) == S_IFDIR) {
        p1 = from;
        p2 = to;
        bp = iobuf;
        while(*bp++ = *p2++)
            ;
        bp[-1] = '/';
        p2 = bp;
        while(*bp = *p1++)
            if (*bp++ == '/')
                bp = p2;
        to = iobuf;
    }
    if (stat(to, &stbuf2) >= 0) {
        if (stbuf1.st_dev == stbuf2.st_dev &&
            stbuf1.st_ino == stbuf2.st_ino) {

```



```
        fprintf(stderr, "cp: cannot copy file to itself.\n");
        return(1);
    }
}
if ((fnew = creat(to, mode)) < 0) {
    fprintf(stderr, "cp: cannot create %s\n", to);
    close(fold);
    return(1);
}
while(n = read(fold, iobuf, BSIZE)) {
    if (n < 0) {
        fprintf(stderr, "cp: read error\n");
        close(fold);
        close(fnew);
        return(1);
    } else
        if (write(fnew, iobuf, n) != n) {
            fprintf(stderr, "cp: write error.\n");
            close(fold);
            close(fnew);
            return(1);
        }
    }
close(fold);
close(fnew);
return(0);
}
```



## WEEK 12

**Write a c program that simulates ls command  
(Use system calls /directory API)**

```
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/mkdev.h>
#include <dirent.h>
#include <stdio.h>

char   typeOfFile(mode_t);
char   *permOfFile(mode_t);
void   outputStatInfo(char *, char *, struct stat *);

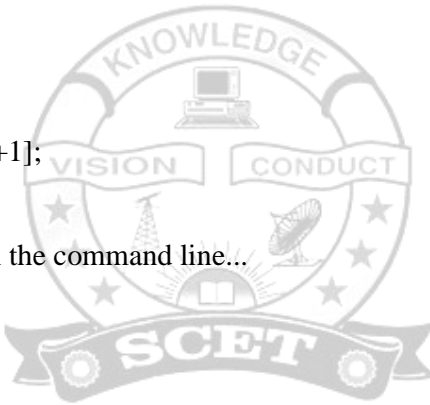
int
main(int argc, char **argv)
{
    DIR *dp;
    char *dirname;
    struct stat st;
    struct dirent *d;
    char filename[BUFSIZ+1];

    /*
     * For each directory on the command line...
     */
    while (--argc) {
        dirname = *++argv;

        /*
         * Open the directory.
         */
        if ((dp = opendir(dirname)) == NULL) {
            perror(dirname);
            continue;
        }

        printf("%s:\n", dirname);

        /*
         * For each file in the directory...
         */
        while ((d = readdir(dp)) != NULL) {
            /*
             * Create the full file name.
             */
            sprintf(filename, "%s/%s", dirname, d->d_name);
```



```
/*
 * Find out about it.
 */
if (lstat(filename, &st) < 0) {
    perror(filename);
    putchar('\n');
    continue;
}

/*
 * Print out the information.
 */
outputStatInfo(filename, d->d_name, &st);
putchar('\n');
}

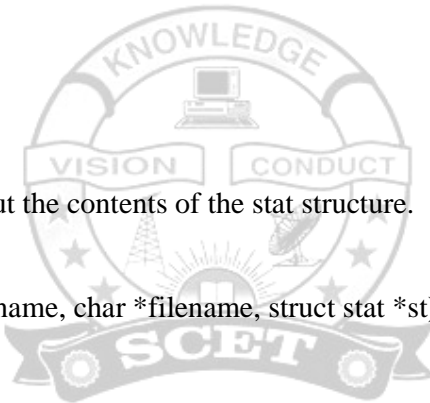
putchar('\n');
closedir(dp);
}

exit(0);
}

/*
 * outputStatInfo - print out the contents of the stat structure.
 */
void
outputStatInfo(char *pathname, char *filename, struct stat *st)
{
    int n;
    char slink[BUFSIZ+1];

    /*
     * Print the number of file system blocks, permission bits,
     * number of links, user-id, and group-id.
     */
    printf("%5d ", st->st_blocks);
    printf("%c%s ", typeOfFile(st->st_mode), permOfFile(st->st_mode));
    printf("%3d ", st->st_nlink);
    printf("%5d/%-5d ", st->st_uid, st->st_gid);

    /*
     * If the file is not a device, print its size; otherwise
     * print its major and minor device numbers.
     */
    if (((st->st_mode & S_IFMT) != S_IFCHR) &&
        ((st->st_mode & S_IFMT) != S_IFBLK))
        printf("%9d ", st->st_size);
    else
```



```

printf("%4d,%4d ", major(st->st_rdev), minor(st->st_rdev));

/*
 * Print the access time. The ctime() function is
 * described in Chapter 7, "Time of Day Operations."
 */
printf("%.12s ", ctime(&st->st_mtime) + 4);

/*
 * Print the file name. If it's a sybolic link, also print
 * what it points to.
 */
printf("%s", filename);

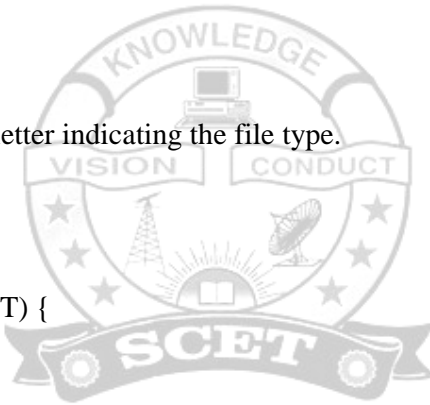
if ((st->st_mode & S_IFMT) == S_IFLNK) {
    if ((n = readlink(pathname, slink, sizeof(slink))) < 0)
        printf(" -> ???");
    else
        printf(" -> %.*s", n, slink);
}
}

/*
 * typeOfFile - return the letter indicating the file type.
 */
char
typeOfFile(mode_t mode)
{
    switch (mode & S_IFMT) {
    case S_IFREG:
        return('-');
    case S_IFDIR:
        return('d');
    case S_IFCHR:
        return('c');
    case S_IFBLK:
        return('b');
    case S_IFLNK:
        return('l');
    case S_IFIFO:
        return('p');
    case S_IFSOCK:
        return('s');
    }

    return('?');
}

/*
 * permOfFile - return the file permissions in an "ls"-like string.

```



```
*/
char *
permOfFile(mode_t mode)
{
    int i;
    char *p;
    static char perms[10];

    p = perms;
    strcpy(perms, "-----");

    /*
     * The permission bits are three sets of three
     * bits: user read/write/exec, group read/write/exec,
     * other read/write/exec. We deal with each set
     * of three bits in one pass through the loop.
     */
    for (i=0; i < 3; i++) {
        if (mode & (S_IREAD >> i*3))
            *p = 'r';
        p++;

        if (mode & (S_IWRITE >> i*3))
            *p = 'w';
        p++;

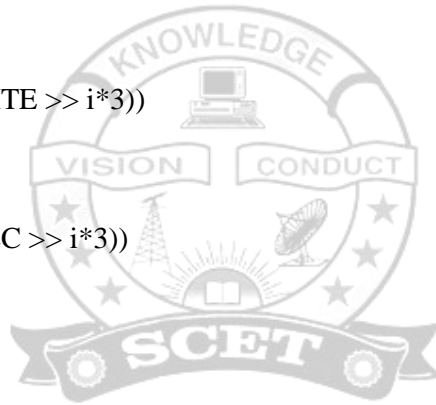
        if (mode & (S_IEXEC >> i*3))
            *p = 'x';
        p++;
    }

    /*
     * Put special codes in for set-user-id, set-group-id,
     * and the sticky bit. (This part is incomplete; "ls"
     * uses some other letters as well for cases such as
     * set-user-id bit without execute bit, and so forth.)
     */
    if ((mode & S_ISUID) != 0)
        perms[2] = 's';

    if ((mode & S_ISGID) != 0)
        perms[5] = 's';

    if ((mode & S_ISVTX) != 0)
        perms[8] = 't';

    return(perms);
}
```





---

## BIBLIOGRAPHY

---

1. David Medinets, *Unix Shell Programming Tools*, McGraw-Hill,
2. Bill Rosenblatt, *Learning the Korn Shell*, O'Reilly and Associates
3. Arnold Robbins, *Effective Awk Programming*, Free Software Foundation / O'Reilly and Associates
4. Anatole Olczak, *Bourne Shell Quick Reference Guide*, ASP, Inc.,
5. UNIX Shell Programming, 4th Edition , Lowell Jay Arthur, Ted Burns
6. Mastering Unix Shell Scripting , by Randal K. Michael
7. The UNIX Operating System, 3rd Edition, by Kaare Christian, Susan Richter

