B.E.

Fifth Semester Examination, Dec.-2007

System Programming & System Administration(IT-303-E)

Note : Attempt any *five* questions. All questions carry equal marks.

Q. 1. (a) What is assembly language? Discuss the advantages & disadvantages of assembly language.

Ans. Programmers found it difficult to write or reach programs in machine language. In their quest for a more convenient language they began to use a mnemonic (symbol) for each machine instruction, which they would subsequently translate into machine language. Such a mnemonic machine language is now called an assembly language.

Assembly language, which is the most machine dependent language used by programmers today.

There are four main advantages to using assembly language rather than machine language.

(i) It is mnemonic; e.g., we write ST instead of the bit configuration 01010000 for the store instruction.

(ii) Address are symbolic, not absolute .

(iii) Reading is easier.

(iv) Introduction of data to program is easier.

A disadvantage of assembly language is that it requires that use of an assembler to translate a source program into object code. Many of the features of 360 or 370 assembly language exist in assembly language for other machines.

Q. 1. (b) Write short notes on the following :

(i) Interpreter **(ii) Assembler**

(iii) Linker **(iv) Loader**

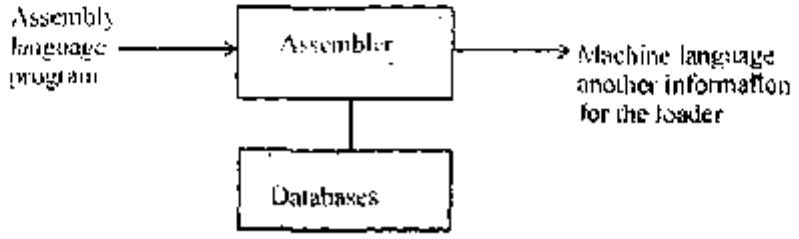
(v) Compiler **(vi) Macros**

Ans. (i) Interpreter : An interpreter is a program that accepts a program written in a high level language and produces an object program which is in machine language. But it translate one statement at a time.

For example, Unix is a high level language which translate one statement at a time by interpreter.

(ii) Assembler : Programmers found it difficult to write or read programs in machine language. In their quest for a more convenient language they began to use a mnemonic (symbol) for each machine instruction, which they would subsequently translate into machine.

An assembler is a program that accepts as input an assembly language program and produces its machine language equivalent along with information for the loader.



(iii) Linker : Usually, a longer program is divided into a number of smaller subprograms called modules. It is easier to develop, test & debug smaller programs. A linker is a program that links (combines) smaller programs to form a single program. While developing a program subroutines are frequently used. The subrou-

tines are stored in a library file. The linker also links subroutines with the main program. The linker links machine codes of the programs. Therefore, it accept user's programs after editor has edited the program and compiler has produced machine codes of the program.

(iv) Loader : Once the assemblers produces an object program, that program must be placed into memory and executed. It is the purpose of the loader to assume that object programs and placed in memory in an executable form.

A loader is a program that places programs into memory and prepares them for execution. The loader program is much smaller than the assembler.

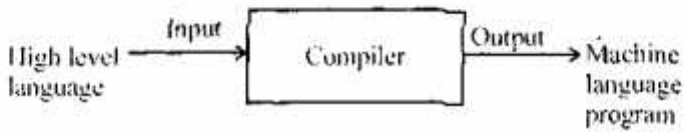
(v) Compiler : A compiler is a program that accepts a program written in a high level language & produces are object program.

Modern compilers must be able to provide the complex facilities that programmers are now demanding. The compiler must furnish complex accessing methods for pointer variables and data structures used in languages like PL I. COBOL & ALGOL 68. Modern compilers must interact closely with the operating system to handler statements concerning the hardware interrupts of a computer.

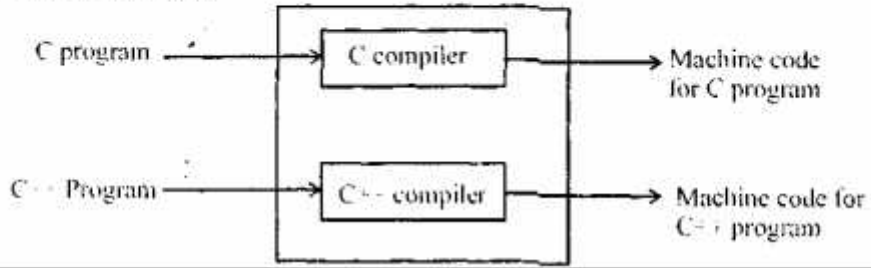
(vi) Macros : To relieve programmers of the need to repeat identical parts of their program, operating systems provide a macro processing facility, which permits the programmer to define an abbreviation of a part of his program & to use the abbreviation in his program. The macro processor treats the identical parts of the program defined by the abbreviation as a macro definition & saves the definition. The macro processor substitutes the definition for all occurrence of the abbreviation in the program.

Q. 2. (a) Discuss compilation process in short.

Ans. The computer understands only machine language. So a program written in a high level language needs to be translated into machine language before it can be executed by the computer. This job of translation is done with help of software program. This program in case of a high level language is known as a compiler. A program written in a high level language is known as source program. The program obtained after compilation is known as object program.



Every language has its own compiler & it can translate only those programs which are written in that particular language. Compilers are large programs which residue permanently on secondary storage. When a program is to be compiled the computer is copied into the main memory of the computer & is executed in the CPU. During compilation the compiler analysis each statement in the source program & generates a sequence of machine instructions. Besides translating the instructions in high level language into machine language instructions the compiler also brings out any errors related to the syntax (grammar) or semantics (meaning) of the specified high level language.



Q. 2. (b) What is meant by single pass & two pass assembler? What features of assembly language required us to build a two pass assembler?

Ans. An assembler is a program that accepts as input an assembly language program & produces its machine language equivalent along with information for the loader.

Source program			First pass		Second pass	
			Relative address	Mnemonic instruction	Relative address	Mnemonic instruction
JOHN	START	0				
	USING	*, 15				
	L	1, FIVE	0	L 1, –(0, 15)	0	L 1, 16 (0, 15)
	A	1, FOUR	4	A 1, –(0, 15)	4	A 1, 12 (0, 15)
	ST	1, TEMP	8	ST	8	ST 1, 20 (0, 15)
FOUR	DC	F'4'	12	4	12	4
FIVE	DC	F'5'	16	5	16	5
TEMP	DS	1F	20	–	20	–
	END					

The first pass has only to define the symbols; the second pass can then generate the instruction & addresses.

An assembler must do the following :

(i) Generate instructions :

(a) Evaluate the mnemonic in the operation field to produced its machine code.

(b) Evaluate the subfields-find the value of each symbol, process literals and assign addresses.

(ii) Process pseudo operations :

We can groups these tasks into two passes or sequential scans over the input; associated with each task are one or more assembler modules.

Pass 1 : Purpose : Define symbols and literals :

(i) Determine length of machine instruction (MOT GET 1).

(ii) Keep track of Location Counter (LC).

(iii) Remember, values of symbols until pass 2 (STSTO).

(iv) Process some pseudo operations, e.g., EQU, DS (POT GET 1).

(v) Remember literals (LITSTO).

Pass 2 : Purpose : Generate object program :

(i) Look up value of symbols (STGET).

(ii) Generate instructions (MOT GET 2).

(iii) Generate data.

(iv) Process pseudo operations.

After all the symbols have been defined by pass 1, it is possible to finish the assembly by processing each card & determining values for its operation code & its operand field. In addition, pass 2 must structure the generated code into the appropriate format for later processing by the loader & print an assembly listing containing the original source & the hexadecimal equivalent of the bytes generated.

Q. 2. (c) Discuss functions of RLD and ESD cards.

Ans. RLD Function :

(i) The location and length of each address constant that needs to be changed for relocation or linking.

(ii) The external symbol by which the address constant should be modified (added or subtracted).

(iii) The operation to be performed (add or subtract).

ESD Function : The ESD cards contain the information necessary to build the external symbol dictionary or symbol table. External symbols that can be referred beyond the subroutines level. The normal labels in the source program are used only by the assembler. & information about them is not included in the object deck.

Q. 3. (a) Describe the input & output of the macro processor. How dependent is it upon the assembler source code format.

Ans. A macro is an abbreviation for a sequence of operations :

e.g. :

```
                    ⋮
A                   1, DATA
A                   2, DATA
A                   3, DATA
                    ⋮
A                   1, DATA
A                   2, DATA
A                   3, DATA
                    ⋮
DATA DC             F'5'
```

In the above program the sequence

```
A                   1, DATA
A                   2, DATA
A                   3, DATA
```

Occurs twice. A macro facility permits us to attach a name to this sequence & to use this name in its place.

A macro processor effectively constitutes a separate language processor with its own language.

We attach a name to a sequence by means of a macro instruction definition, which is formed in the following manner :

```
Start of definition  →   MACRO
Macro name           →   [ ]
                         ⎧ —
Sequence to be abbreviated ⎨ —
                         ⎩ —
End of definition    →   MEND
```

The MACRO pseudo-op is the first line of the definition & identifies the following line as the macro instruction name.

Once the macro has been defined, the use of the macro name as an operation mnemonic in an assembly program is equivalent to the use of the corresponding instruction sequence.

The macro processor can be added as a pre-processor to an assembler, making a complete pass over the

input text before pass 1 of the assembler. The macro processor can be implen.

Q. 3. (b) What are macros? Justify their need. Also define macro instructions with the help of example.

Ans. A macro is an abbreviation for a sequence of operations. A macro facility permits us to attach a name to the sequence and to use this name in its place. Macro instruction are single-line abbreviations for groups of instructions. The programmer essentially defines a single instructions to represent a block of code. For every occurrence of this one-line macro instruction in a program, the macro processing assembler will substitute the entire block.

Micro instructions are usually considered an extension of the basic assembler language & the macro processor is viewed as an extension of the basic assembler algorithm.

We attach a name to a sequence by means of a macro instruction definition, which is formed in the following manner :

Start of definition → MACRO

Macro name → [ ]

Sequence to be abbreviated → [— / — / —] MEND

End of definition →

The MACRO pseudo-op is the first line of the definition & identifies the following line as the macro instruction name. Following the name line is the sequence of instructions being abbreviated the instructions comprising the 'macro' instruction. The definition is terminated by a line with the MEND ('macro end') pseudo-op.

Once the macro has been defined, the use of the macro name as an operation mnemonic in an assembly program is equivalent to the use of the corresponding instruction sequence.

Example : Assigning the name 'INCR' to the repeated sequence.

Source		**Expanded source**		
MACRO				
INCR				
A	1, DATA			
A	2, DATA			
A	3, DATA			
MEND				
		A	1, DATA	
		A	2, DATA	
		A	3, DATA	
INCR				
		A	1, DATA	
		A	2, DATA	
		A	3, DATA	
DATA DC	F'5'	DATA	DC	F'5'

This process of replacement is called expanding the macro. The macro definition itself does not appear in the expanded source code. The definition is saved by the macro processor. The occurrence in the source program of the macro name, as an operation mnemonic to be expanded, is called a macro call.

Q. 3. (c) What are the advantages of incorporating the macro processor into pass of I assembler.

Ans. The major advantages of incorporating the macro processor into pass I are :

(i) Many functions do not have to be implemented twice (e.g., read a card, test for statement type).

(ii) There is less overhead during processing : functions are combined and it is not necessary to create intermediate files as output from the macro processor & input to the assembler.

(iii) More flexibility is available to the programmer in that he may use all the features of the assembler (e.g., EQU statements) in conjunction with macros.

Q. 4. (a) What is Operating System? Discuss basic features of Unix O.S.

Ans. Operating system is the most important program that runs on a computer. Every general purpose computer must have an operating system to run other programs. Operating systems perform basic tasks, such as recognizing input from the keyboard, sending output to the display screen, keeping track of files & directions on the disk & controlling peripheral devices such as disk drives and printers.

"Operating system is an interface between the hardware and the user."

Features of Unix : Unix is such an operating system that can be run on a wide range of machines, from micro computers to mainframes. There are a variety of reasons which have made UNIX an extremely popular operating system.

(a) Portability : The reason behind UNIX's portability is that it is written in a high-level language which has made it easier to read, understand and change. Its code can be changed & compiled on a new machine.

(b) Machine Independent : The UNIX system does not expose the machine architecture to the user. Thus, it becomes very easy to write applications that can run on micros, minis or mainframes.

(c) Multi-User Capability : UNIX is a multi-user system. A multi-user system is a system in which the same computer resource like hard disk, memory etc. can be used or accessed by many users simultaneously. In UNIX terminology, the main computer is called the server or the console.

(d) Multitasking Capability : UNIX has the facility to carry out more than one job at the same time. This feature of UNIX is called multitasking.

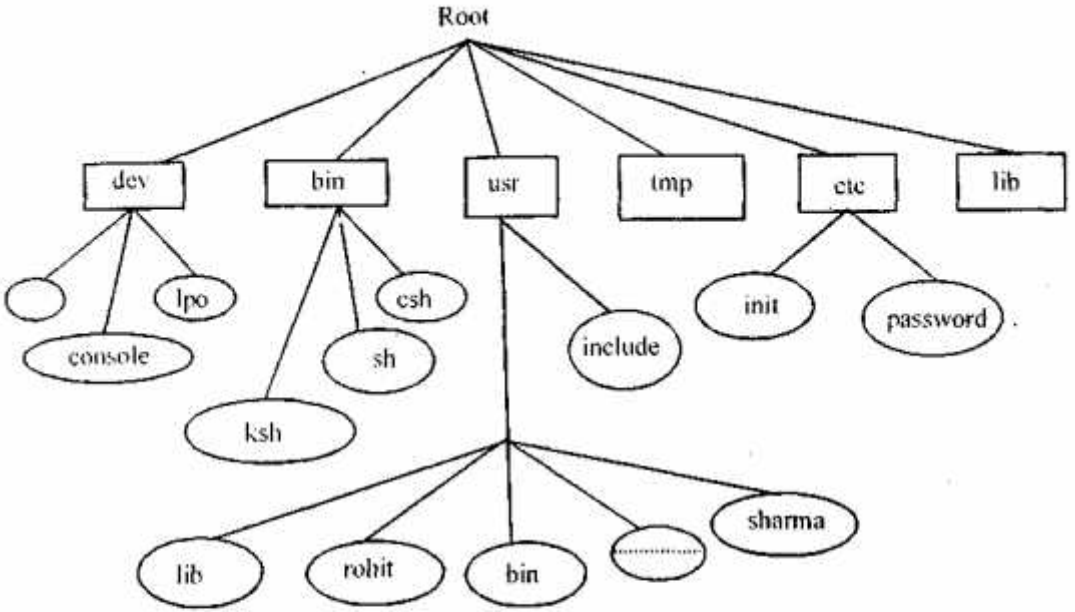
(e) Security : UNIX supports a very strong security them. It enforces security at three levels. Firstly, each user is assigned a login name & a password.

Q. 4. (b) What do you understand by File System? Discuss Unix File System.

Ans. The most important function of an operating system is the effective management of information. We frequently use the expression "information processing system" to refer to a computer system. The modules of the operating system dealing with the management of information are often called the file system. A file system is concerned with the logical organization of information. The file system deals with collections of unstructured and uninterpreted information at the operating system level. Each separately identified collection of information is called a file.

Unix File System : A file in UNIX is sequence of bytes. Different programs expect various levels of structure, but the kernel does not impose any structure on files, & no meaning is attached to its contents the meaning of bytes depends solely on the programs that interpret the file.

The UNIX file system supports two main objects : files and directories. Directories are nothing but files which have a special format.



Unix File System

Figure shows a typical UNIX file system. The file system is organized as a tree with a single root node called the root (written "/"); every non-leaf-node of the file system structure is a directory, & leaf nodes of the tree are either directories or regular files or special devices.

/dev contains devices files. such as /dev/console, /dev /lpo;/ dep/mnt & so on.

Q. 5. Discuss the following :

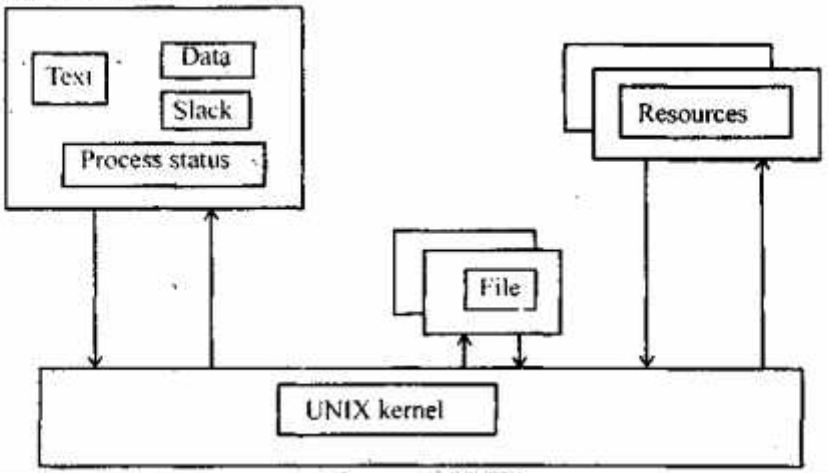
(a) Processes in Unix

(b) Permission settings for a file

(c) Filters and pipelines

(d) Vi editor and its modes of operation.

Ans. (a) Processes in Unix : The behaviour of a UNIX process is defined by its text segment, data segment & stack segment as shown in fig.



The text segment contains the compiled object instructions, the data segment contains static variables, and the stack segment holds the runtime stack used to store temporary variables. A set of source file that is compiled & linked into an executable form is stored in a file with the default name of a.out.

The executable file is created by the compiler and linker. These utilities do not define a process; they define only the program text & a template for the data component that the process will use when it executes the program. When the loader loads a program into the computer's memory. The system creates appropriate data and stack segments, called a process.

A process has a unique process identifier, a PID, that is essentially a pointer–an integer into a table of process descriptors used by the UNIX OS kernel to reference the process is descriptor.

(b) Permission Setting for a File : The programmer wishing to access the file may write the PL/I like statement :

READ FILE (ALPHA) RECORD (6) SIZE (250) LOCATION (BUFF). To read the sixth of the 250-byte logical records from file ALPHA and copy it into main storage at location BUFF. The compiler would translate this READ statement into the appropriate subroutine calls to the file system.

Let us consider the actions that must be performed by a simple file system in order to satisfy the above request :

(i) The physical location of file ALPHA must be determined.

(ii) The physical block containing record 6 must be determined.

(iii) The physical block must be read from the drum into a buffer in main storage.

(iv) Record 6 must be extracted from the block buffer and moved to location BUFF.

(c) Filters and Pipelines :

Filters : Filters are programs that accept the input from the standard input file filters it and then send the output to standard output file (monitor). LINUX has a number of filters.

(a) Grep Filter : This filter searche a file for a regular expression and displays the lines that match the pattern. It cannot be used without specify a regular expression.

Syntax : Grep regular-expression [file name].

(b) The WC Filter : WC filter is used to count the number of lines, words and characters in the standard input or a file.

Syntax : $ WC filename.

(c) Cut Filter : Cut filter is used when specific columns from output of certain commands need to be executed.

Syntax : $ cut [options [filename]].

(d) File tr Filter : This filter is used to translate one set of characters to another. This filter is used where the column n separator is a combination of character or spaces.

Syntax : $ tr ':' ' ' < file_name

(e) The Sort Filter : This filter arranges each line of the standard input in ascending order. Sort filter sorts data based on ASCII values.

Pipeline : Occasionally, we may require the output of one program as the input to other. For this either we can use a temporary file or a pipeline. A pipeline helps us to complete the same thing without using a file. A temporary file if created internally by DOS. For sending output data of first program as an input to the second program. A vertical '|' denotes a pipe. The command,

prog 1 > temp file> prog 2

using pipe can be written as

prog 1 | prog 2.

(d) Vi editor and its Modes of Operation :

Visual Editor (Vi) : This editor is used to display the contents of a file and to add, delete or alter the file. To start vi editor the syntax is :

$ vi [file name]

Vi editor can be used in the following modes :

* **Command Mode :** By default, vi editor exist in command mode. All vi commands are executed in this mode.

* **Insert Mode :** In this mode, text can added to a file. To invoke this mode, the key i is pressed.

* **Append Mode :** This mode is the same as insert mode except that it appends, contents into a file. To invoke this mode, the key 'a' is pressed.

To more back to command mode <Esc> key is pressed.

Q. 6. (a) How would you convert all capital latters in a file to small case latters? Explain with the help of example.

Ans. I'll got a heap (> 50) of *.html files which have most of their hyperlinks referencing files with upper case latter in them, some of which are file. html and other which are like File With Camel-case Name.html'. Now, I want to be able to batch rename the files to lower case, replacing Camel-case with Camel-case where that occurs, & then change all of the hyperlinks in the files to match these new names. Without manually finding each one & changing each reference.

Operation could be done automatically, i.e., with sed or vim or something, but I can't quite get my head around the regular expression.

Usage :

```
// include <ctype.h>
lower = to lower (upper);
```

Where :

int upper;

is the upper case character you want to convert.

int lower;

is the resulting lower case latter. If "upper" is not an upper case latter, the result of "to lower" is the same as the argument value.

Q. 6. (b) Discuss about "Grep" command with the help of example.

Ans. "Grep" Command : The word grep stands for global regular expression printer. The grep command is used for searching one or more files for particular bit patterns globally & displays the line(s) containing it. The syntax of grep command is :

$ grep pattern file[s] ↵

The above command displays the pattern specified from one or more files on the monitor. If more than one file are specified thus each line is preceded by the name of the file. If the pattern is not found then the grep command displays a new prompt.

f grep Command : It is an advanced pattern matching command and can be used for searching two or more strings simultaneously. For example,

$ grep 'marks

pass

fail' data ↵

egrep Command : It is also an advanced pattern matching command & can be used for searching more than one string with more compaction than grep. For example.

$ fgrep 'marks | pars | fail' data ↵

Sort Command : It is used to sort the contents of the specified file in alphabetical order. The syntax of sort command is :

$ sort filename ↵

Q. 6. (c) What do you understand by AWK utility? Justify its significance.

Ans. AWK Utility :

AWK was named after its original developers : Aho, Weinberger and Kernighan. awk scripts are readily partable across all flavors of UNIX/LINUX.

The following shell command scans the contents of file called oldfile, charging all occurrences of the word "UNIX" to "Linux" & writing the resulting text a file called newfile.

$ awk "(grubs (/UNIX/, "LINUX"); print) 'old file/>\ new file.

The most significant characteristic of awk are :

* It views its input as a set of records & fields.
* It offers programming constructs that are similar to the C language.
* It offers built-in function and variables.
* Its variables are typeless.
* It performs patterns matching through regular expressions.

awk scripts can be very expressive and are often several pages in length. The AWK language offers the typical programming constructs expected in any high-level programming language. It has been described as an interpreted version of the C language, but although there are similarities, awk differs from C both semantically and syntactically.

Q. 7. (a) What is shell programming? Justify significance of shell program.

Ans. While using UNIX, we cannot directly use the kernel. It is the shell, one of the utility programs, that starts up the kernel when the user logs in. The shell provides a prompt symbol.

For accessing the hardware, the user requests to shell, the shell requests kernel and then the kernel requests to the hardware. The shell handles the user interaction with the system. The Bourne shell and C shall are most popular used shells. The Bourne shell was developed by Stephen Bourne of Bell Labs for Bell lab releases of UNIX and C shall was designed by Bill Joy at the University of California at Barkeley.

There are basically two types of commands : Simple commands and Complex commands.

Simple Command : A simple command is a sequence of non-blank words separated by blanks. The first words specifies the name of the command to be executed. Except as specified below, the remaining words are passed as arguments to the involved command. The command name is passed as argument o,

For example :

```
$ ls
mail        doc         lib         public_html
news        emacs       man         bin
outgoing    inconning   tmp
```

Shell Programming Include the Following Features :

* **Interactive Processing :** The users & the system communication in the form of an interactive dialogue with the shell.

* **Back Ground Processing :** All time-consuming, non-interactive tasks are performed at the background. Hence, the system can perform many tasks at the same time.

* **Shell Scripts :** Sequence of shell commands that are frequently used in stored in a file.

* **Shell Variables :** Users can control the behaviour of the shell and other programs by storing data in variables that are significant to the shell.

Q. 7. (b) Discuss following with the help of examples :

(i) Read statement

(ii) Echo statement

(iii) Tput command.

Ans. (i) Read Statement : This is the simplest form of reading unformatted data from the standard input channel, for example, if the type declaration are the same as for the PRINT example,

```
READ*, Long_name
READ *, X, Y, Z
READ *, Logical
```

Note : * Each READ statement reads from a newline.

* The read statement can transfer any object of instrinsic type from the standard input.

(ii) Echo Statement : It displays or echoes, whatever we type after the word echo, on the standard output device i.e., the monitor

```
$echo India is Great ↵
India is Great
$
```

If some extra spaces are provided by the user, the Echo command replaces those by single blanks in between, as given below :

```
$ echo Apoorva, a very nice girl ↵
Apoorva, a very nice girl
$
```

If nothing is typed after the word echo & the Enter key is pressed a blank line is displayed as shown below :

```
$ echo ↵
$
```

(iii) Tput Command : There are different type of Tput command :

(a) Tput Clear : This command clears the standard output device, the screen and positions the cursor at the top left corner of the screen.

(b) Tput Cup : This command, followed by the screen coordinates, positions the cursor at the specified row and column.

e.g., $ tput 1540

This will place the cursor at row 15 and column 40.

(c) Tput smso : This sets the screen to reverse video.

(d) Tput rmso : This sets the screen back to normal form reverse video.

(e) Tput blink : This command displays a blinking output. But, this command does not work with a telnet session.

(f) Tput reset : This resets the screen back to the default settings.

Q. 7. (c) What are shell variables? Describe the rules for building shell variables.

Ans. Shell Variables : There is a set of predefined variables in the shell. These variables are used to store values & also to change the behaviour of called programs. A simple example of a variable is the HOME variable, which contains the path to your home directly. If you type cd without any arguments, this is the directory where you will end up. To print the value of the variable HOME, you can write the following :

```
$ echo $ HOME
/users/mat kin
```

The PATH Variable : One of the more important variable is the path variable. The path variable controls where the shell searches for commands, when you type them to at the prompt.

When you type a non-built-in command to the shell, the shell searches for a program to execute. The programs are simply executable file somewhere in the file system. They are executable either because they are compiled programs or because they are scripts that may be executed.

Other Predefined Variables :

HOME : This is the home directory of your account. It is also used as the default when typing cd without any arguments.

PS1 : The primary prompt string, normally set to '$'. This is what the computer prints whenever it is ready to process another command.

PS 2: The secondary prompt string, normally set to '>'. The shell prints this prompt whenever you have type an incomplete time, e.g., if you are mostly a closing quotes. For example :

```
$ echo' hello
> world'
 hello
 world
```

IFS : This is set of the internal field separators, normally set of space, tab, and blank.

Q. 8. (a) Outline the jobs of a system administrator. Also discuss the different ways to carry out these jobs by system administrator.

Ans. System Administrator or sysadmin is a person employed to maintain, and operate a computer system or network. System administrator may be members of an information technology department.

The duties of a system administrator are wide-ranging, and vary widely from one organization to another. Sysadmins are usually charged with installing, supporting and maintaining servers or other computer systems, & planning for and responding to service outages and other problems. Other duties may include scripting or light programming, project management for system related projects, supervising or training computer operators etc. A System Administrator must demonstrate a blend of technical skills and responsibility.

A system administrator's responsibilities might include :

- Performing routine audits of system and software.

- Performing backups.
* Applying operating system updates, patches, & configuration changes.
* Installing and configuring new hardware & software.
* Answering technical queries.
* Responsibilities for security.
* Troubles hooting any reported problems System performance tuning.
* Insuring that the network infrastructure is up and running.

Q. 8. (b) Describe "find" command. What is the noticeable difference between options used by "find" command as compared to other Unix Commands?

Ans. "Find" : The find command is used to find a particle file in a direction that contains large number of files. The following options can also be used along with command for advance searches. The syntax of 'find' command is :

$ find [path] [expression]

The options used with 'find' command are :

(i) The -name option : It is used to specify the filename to be searched.

(ii) The-print option : It is given to display the full path name of the files. The dot is used to specify to search in the current directory.

(iii) The-type option : The -'type' option with the find command can be used to search for a file of a specific type. This option can be used with different parameters b, c, d, f and l.

(iv) The-mtime option: It searches for files that have been modified within a specific number of days before the current date.

Q. 8. (c) Discuss about following :

(i) Formatting a Disk

(ii) Making a File System (Under Unix Environment).

Ans. (i) Formatting a Disk : Formatting or initializing, a disk is the process of preparing the disk so that it can store data or programs on it. For formatting a floppy disk perform the following steps :

(i) Insert unformatted (blank) disk in the floppy disk drive.

(ii) The My Computer, click on the icon for the disk you want to format (the floppy disk).

(iii) On the file menu, click format.

Note :

(i) Formatting of a disk removes (deletes) all information already on the disk, such as previous files stored on it.

(ii) You cannot format a disk if files are already open on that disk.

(ii) Making a File System : The root (/) directory contains the user created as well as standard directories such as /bin, /dev, /etc., lib etc. For storing programs implementing UNIX commands, system libraries, standard devices & so on. The root directory contains a file, namely unix, which is the UNIX kernel.

The /dev directory contains special files for various peripherals such as, keyboard, mouse, printer, monitor etc. which are called the device drivers & contain the address of the device driver & corresponding parameters.

The /bin directory contains the compiled, executable or binaries of the essential UNIX utilities & programs. The /usr directory stores all the user directories. These directories for different users are created by the

system administrator. The subdirectories –/user/lib, /usr/ bin and /usr/temp exist in the/usr directory. The /lib directory contains main parts of (compiler such as C subroutines library, the C preprocessor. The /temp directory stores all the temporary files created by UNIX or users when utility programs are executed. such as editor. These files are deleted automatically when we shut down the system & restart it.