

Dec 2011

Paper Code : CSE - 101 - F

Figures in the bracket indicate full marks.

**Note :** Attempt any five questions in all. Question No. 1 is compulsory and attempt at least one question from each Section.

**Q. 1 (a) Differentiate primary & secondary memory. (2.5)**

**Ans. Primary Memory :** The main memory or the memory on the motherboard is called the primary memory.

This primary memory can be further divided into following types

(i) RAM (Random Access Memory)

(ii) ROM (Read Only Memory)

**Secondary Memory :** Secondary memory is not a memory in conventional terms, it is actually the storage device used to store the program or the data. Floppy disk, hard disk, CD-ROM etc. are examples of secondary memory.

Because of its size, secondary memory is also called mass storage device.

**Q. 1(b) Differentiate between Multiprogramming operating system and Multiprocessing operating system. (2.5)**

**Ans. Difference between Multiprogramming and Multiprocessing :** Multiprocessing is the simultaneous execution of two or more processes by a computer system having more than one CPU. Comparatively, multiprogramming is the interleaved execution of two or more processes by a single, CPU system. Hence, while multiprogramming involves execution of a portion of one program, then a portion of another, etc., in brief consecutive periods, multiprocessing involves simultaneous execution of several program segments of the same or different programs.

**Q. 1 (c) Explain ternary operator in C by providing suitable example. (2.5)**

**Ans. Ternary operator :** C ternary operator is sometimes called conditional operator or trinary operator because it is a shorthand of combination of the *if-else* and *return* statement and has three operands. The syntax of C ternary operator is as follows:

condition ? expression 1 : expression 2

C first evaluates the condition. Based on the return value of the condition the second or third operand is evaluated:

- If the condition is evaluated to true (1), only the second operand (expression 1) is evaluated.

Then the operator return the value of the expression 1.

- If the condition is evaluated to false (0), only the third operand (expression 2) is evaluated. The value of the expression 2 is returned.

The ternary operator can be rewritten as the *if-else* and *return* statement as follows:

```
if(condition)
```

```
return expression 1;
```

```
else
```

```
return expression 2;
```

**Q. 1 (d) Describe Hypertext and its usability.**

**Ans.** Hypertext is text displayed on a computer or other electronic device with references to other text that the reader can immediately access, usually by a mouse click or keypress sequence. Apart from running text, hypertext may contain tables, images and other presentational devices. Hypertext is the underlying concept defining the structure of the World Wide Web. It is an easy-to-use and flexible format to share information over the Internet.

*Types and uses of hypertext :* Hypertext documents can either be static (prepared and stored in advance) or dynamic (continually changing in response to user input). Static hypertext can be used to cross-reference collections of data in documents, software applications, or books on CDs. A well-constructed system can also incorporate other user-interface conventions, such as menus and command lines. Hypertext can develop very complex and dynamic systems of linking and cross-referencing. The most famous implementation of hypertext is the World Wide Web.

**Q. 1 (e) How an array of structures initialized ?**

**(2.5)**

**Ans.** An array of structures can be declared just like an ordinary array. However, the structure has to be defined before an array of its type is declared.

```
struct student
{
    int roll ;
    char name [20] ;
    int age ;
    char class [8] ;
};
struct students [50] ;
```

In this declaration **s** is a 50 - element array of structures.

Individual elements of a structure in an array of structures are accessed by referring to structure variable name, followed by subscript, followed by dot operator, and ending with the structure member desired. Syntax is

sname [subscript].member

**Q. 1 (f) Declare a pointer to a function that accepts three integer arguments and returns a floating point quantity.**

**(2.5)**

**Ans.** `#include <stdio.h>`  
`#include <conio.h>`

```
int add(int a, int b, int c);
int mul(int a, int b, int c);
int (*oper[4])(int a, int b, int c) = {add, mul};
int main()
{
    int i,
    float result;
    int a=10;
    int b=5;
    int c=2
    printf("Enter the value between 0 and 3 : ");
```

```

scanf("%d",&i);
result = oper[i](a,b,c);
printf("result is %f",result);
getch();
}

int add(int i, int j,int k)
{
    return (i+j+k);
}

int mul(int i, int j,int k)
{
    return (i*j*k);
}

```

**Q. 1 (g) Explain break and continue statements.**

**(2.5)**

**Ans. Break statement :** It is used to terminate any type of loop such as *while loop*, *do while loop* and *for loop*. *break* statement terminates the loop body immediately and passes control to the next statement after the loop.

**Continue statement :** It is used to skip over the rest of the current iteration in . After *continue* statement, the control returns to the top of the loop.

**Q. 1 (h) Compare the statements by providing suitable example :  $i++$  and  $++i$ .**

**(2.5)**

**Ans.  $++i$  :** They operate on a single operand.  $++$  The increment operator increments the value of the variable by 1.

**Prefix increment :** Operator is written before the operand.  $++i$  is equivalent to  $i = i + 1$ .

**Example:**

```

#include<stdio.h>
void main
{
    int i=10;
    printf("i=%d/n",i);
    printf("i=%d/n",++i);
    printf("i=%d/n",i);
}

```

**Output :**

```

i = 10
i = 11
i = 11

```

**$i++$  :**

**Postfix increment :** Operator is written after the operand  $i++$ .

**Example:**

```

#include<stdio.h>
void main
{
    int i=10;

```



```
printf("i=%d\n",i);
printf("i=%d\n",i++);
printf("i=%d\n",i);
}
```

**Output:**

```
i = 10
i = 10
i = 11
```

### Section - A

**Q.2.(a) Explain the Generation of microprocessors.**

(8)

not available now go to [studentsuvidha.in](http://studentsuvidha.in)

**Q.2.(b) Differentiate between Windows and linux.**

(8)

**Ans.** Difference between Linux and Windows are as follows :

Topic	Linux	Windows
<b>Price</b>	The majority of linux variants are available for free or at a much lower price than Microsoft Windows.	Microsoft Windows can run between \$50.00-\$150.00 US dollars per each license copy.
<b>Ease</b>	Although the majority Linux variants have improved dramatically in ease of use, windows is still much easier to use for new computer users.	Microsoft has made several advancements and changes that have made it a much easier to use operating system, and although arguably it may not be the easiest operating system, it is still Easier than Linux.
<b>Reliability</b>	The majority of Linux variants and versions are notoriously reliable and can often run for months and years without needing to be rebooted.	Although Microsoft Windows has made great improvements in reliability over the last few versions of Windows, it still cannot match the reliability of Linux.
<b>Software</b>	Linux has a large variety of available software programs, utilities, and games. However, windows has a much larger selection of available software.	Because of the large amount of Microsoft Windows users, there is a much larger selection of available software programs, utilities, and games for Windows.
<b>Software cost</b>	Many of the available software programs, utilities, and games on available on Linux are freeware or open source. Even such complex programs such as Gimp, Open office, staroffice and wine are available for free or at a low cost.	Although Windows does have software programs, utilities, and games for free, the majority of the programs will cost anywhere between \$20.00-\$200.00+US dollars per copy.
<b>Hardware</b>	Linux companies and hardware manufacturers have made great advancements in hardware support	Because of the amount of Microsoft Windows user and the broader driver support, windows has a much larger

<b>Security</b>	for Linux and today Linux will support most hardware devices. However, many companies still do not offer drivers or support for their hardware in Linux.	support for hardware devices and a good majority of hardware manufacturers will support their products in Microsoft windows.
<b>Open Source</b>	Linux is and has always been a very secure operating system. Although it still can be attacked when compared to Windows, it much more secure.	Although Microsoft has made great improvements over the years with security on their operating system, their operating system continues to be the most vulnerable to viruses and other attacks.
<b>Support</b>	Many of the Linux variants and many Linux programs are open source and enable users to customize or modify the code however they wish to.	Microsoft Windows is not open source and the majority of windows programs are not open source.
	Although it may be more difficult to find users familiar with all Linux variants, there are vast amounts of available online documentation and help, available books, and support available for Linux.	Microsoft Windows includes its own help section, has vast amount of available online documentation and help, as well as books on each of the versions of Windows.

#### Q.2. (c) Working of scanner.

(4)

**Ans.** Steps that a scanner goes through when it scans a document:

(i) The document is placed on the *glass plate* and the *cover* is closed. The inside of the cover in most scanners is flat white, although a few are black. The cover provides a uniform background that the scanner software can use as a reference point for determining the size of the document being scanned. Most flatbed scanners allow the cover to be removed for scanning a bulky object, such as a page in a thick book.

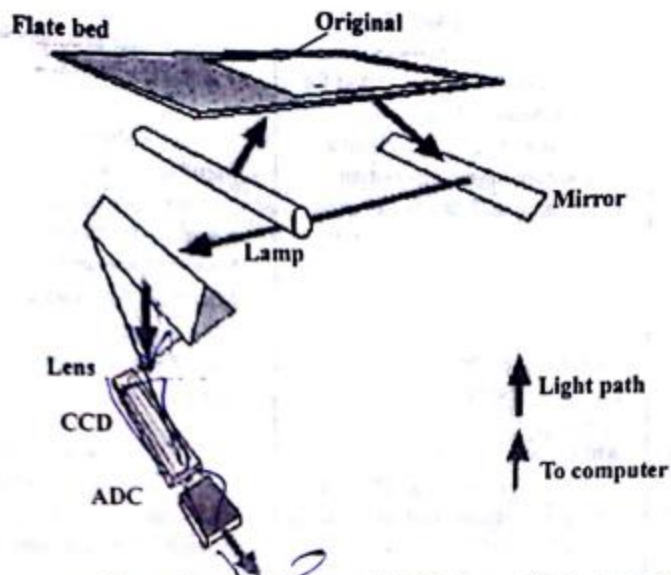
(ii) A lamp is used to illuminate the document. The lamp in newer scanners is either a cold cathode fluorescent lamp (CCFL) or a xenon lamp, while older scanners may have a standard fluorescent lamp.

(iii) The entire mechanism (mirrors, lens, filter and CCD array) make up the *scan head*. The scan head is moved slowly across the document by a *belt* that is attached to a stepper motor. The scan head is attached to a *stabilizer bar* to ensure that there is no wobble or deviation in the *pass*. Pass means that the scan head has completed a single complete scan of the document.

(iv) The image of the document is reflected by an angled *mirror* to another mirror. In some scanners, there are only two mirrors while others use a three mirror approach. Each mirror is slightly curved to focus the image it reflects onto a smaller surface.

(v) The last mirror reflects the image onto a *lens*. The lens focuses the image through a *filter* on the CCD array.

The filter and lens arrangement vary based on the scanner. Some scanners use a *three pass* scanning method. Each pass uses a different color filter (red, green or blue) between the lens and CCD array. After the three passes are completed, the scanner software assembles the three filtered images into a single full-color image.



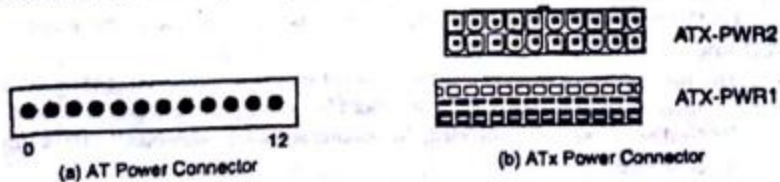
Most scanners today use the *single pass* method. The lens splits the image into three smaller versions of the original. Each smaller version passes through a color filter (either red, green or blue) onto a discrete section of the CCD array. The scanner combines the data from the three parts of the CCD array into a single full-color image.

Another imaging array technology that has become popular in inexpensive flatbed scanners is *contact image sensor* (CIS). CIS replaces the CCD array, mirrors, filters, lamp and lens with rows of red, green and blue *light emitting diodes* (LEDs). The image sensor mechanism, consisting of 300 to 600 sensors spanning the width of the scan area, is placed very close to the glass plate that the document rests upon. When the image is scanned, the LEDs combine to provide white light. The illuminated image is then captured by the row of sensors. CIS scanners are cheaper, lighter and thinner, but do not provide the same level of quality and resolution found in most CCD scanners.

**Q.3.(a) Explain the Various input/output ports and connectors.**

**Ans. Input Output Ports and Connectors :** Input Output ports are used to connect input and output devices to the motherboard. The connectors, on the other hand, are used to connect supply to input/output devices and the motherboard.

(i) **Power Connector :** The power connectors are of two types AT Power connector and ATx power connector. Generally a motherboard has both the connectors so that any of the power supplies (AT or ATx) can be connected to the computer system. Both the connectors are shown in Fig.



**Fig. : Power Connector**



**(a) AT power connector :** It is a 12 pin power connector divided into two parts: pin no. 1-6 and 7-12. The detail of these pins is given in table below :

Pin	Signal Name	Pin	Signal Name
1	Power good signal	7	Ground
2	+5 V	8	Ground
3	+12 V	9	-5 V (DC)
4	-12 V	10	+5 V (DC)
5	Ground	11	+5 V (DC)
6	Ground	12	+5 V (DC)

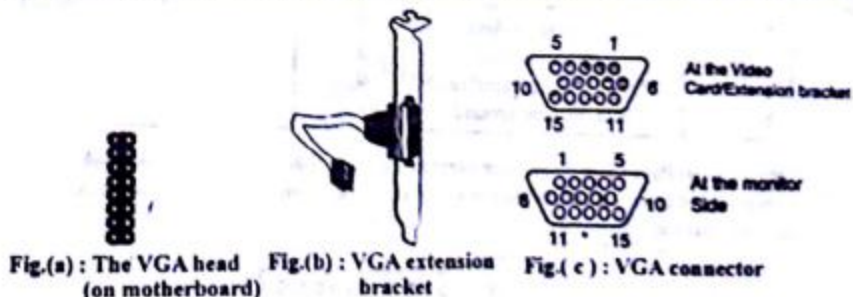
The AT power supply provides two plugs each containing 6 wires. The plugs are inserted into the power connector on the motherboard. However the plugs are so oriented that the black wires on each plug are together i.e. place 2 black wires on each of the plug closed to the centre.

**(b) ATx power connector :** It is a single 20 pin connector having  $\pm 5$  V,  $\pm 12$  V, and optical 3.3 V. The detail of these pins is given in table below :

Pin	Signal Name	Pin	Signal Name
1	+3.3 V	11	+3.3 V
2	+3.3 V	12	-12 V
3	Ground	13	Ground
4	+5 V	14	PW-ON
5	Ground	15	Ground
6	+5 V	16	Ground
7	Ground	17	Ground
8	Power good	18	-5 V
9	+5 VSB	19	+5 V
10	+12 V	20	+5 V

The ATx power supply provides one plug containing 20 wires. ATx supports remote power ON/OFF i.e. the motherboard can turn off the system power through software control. For example, the shut down feature of windows operating system can be used to switch the power off.

**Monitor Socket :** Every motherboard has a VGA socket/header as shown in Fig.(a). VGA extension bracket [See Fig.(b)] is used to connect VGA socket on one and the external monitor cable on the other.



The external monitor cable has a 15 pin male connector and VGA extension bracket has a 15 pin female connector as shown in Fig.

**(ii) Serial (COM) and Parallel (LPT) ports :**

**(a) Serial Port :** Every motherboard has headers for serial and parallel ports for communicating with serial and parallel peripheral devices. The headers for serial port (COM1) and Parallel port (LPT1) are shown in Fig.



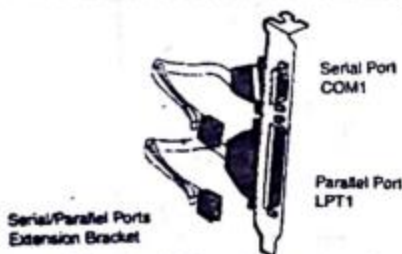
(a) Serial port (on motherboard)



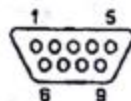
(b) Parallel port (on motherboard)

**Fig.**

Similar to VGA bracket, a serial/parallel ports bracket is used to connect the motherboard on one hand and the devices on other. The bracket is shown in Fig.



**Fig. : Serial/Parallel port bracket**

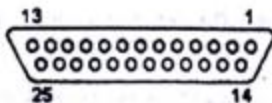


**Fig. : COM connector**

The diagram of a Serial 9 pin male connector, available on the bracket is shown in Fig. and the detail of its various pins is given in Table

Pin	Pin name	Pin	Pin name
1	CD, Carrier detect	6	DSR, Data set ready
2	RxD, Recovery data	7	RTS, Request to send
3	TxD, Transmit data	8	CTS, Clear to send
4	DTR, Data terminal ready	9	RI, Retry intensity
5	GND, System ground		

**(b) Parallel Port :** The diagram of parallel 25 pin D-Sub female connector available on the bracket is shown in Fig. and the detail of its pins is given in Table. This connector is used to connect a parallel device such as a printer.



**Fig. : Parallel port**



Pin	Pin Name	Pin	Pin Name
1	/ stroke	14	/ AUTO FD
2-9	D0 - D7 (Data bits)	15	/ ERROR
10	/ACK	16	/ INIT
11	Busy	17	/ SELIN
12	PE	18-25	GND
13	SEL		

(iii) **Universal Serial Bus (USB) port :** Universal Serial Bus (USB) was developed by Compaq, DEC, IBM, Intel, Microsoft, NEC, and Northern telecom. It is true plug 'n' play connector in the sense that devices can be attached and removed while the computer is on. The operating system can detect a device as soon as it is attached to USB port. On the mother board it is a 5 pin connector as shown in Fig. Its pins detail is given in table.

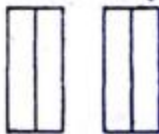


Fig. : USB connector

Pin	Pin Name
1	+5 VDC (Red)
2	Data - (White)
3	Data + (Green)
4	GND (Black)
5	GND (Black)

(iv) **PS-2 port :** This port enables the user to use a keyboard and/ or a mouse with the computer system. The 6-pin port is shown in Fig.

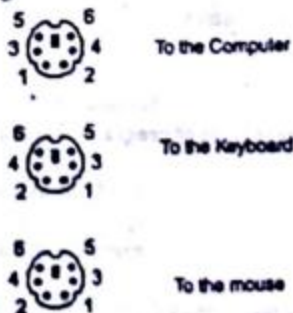


Fig. : PS-2 Port

The detail of pins is given in Table below :

Pin	Pin Name
1	/ DATA
2	n/c
3	GND
4	+ 5 V
5	CLOCK
6	n/c

### Q.3.(b) Classification of Operating Systems.

(5)

**Ans. Classification of Operating Systems:-** Operating systems can be divided broadly into four types, based on the computers they control and applications they support.

**Real time OS :** This type of operating systems are used to control Scientific devices and similar small instruments where memory and resources are crucial. These type of devices have very limited or no end user utilities , so more effort should go into making the OS really memory efficient and fast (less coding), so as to minimize the execution time , in turn saving on power as well.

E x : VHDL, 8086 etc.

**Single User single Task:** This type of OS is just better version of Real time OS , where one User can use the computer to do one thing at a time. which means that doing thing more than one thing at a time is difficult in this type of OS. The handhelds or the palmtop computers are good examples of this type of systems.

E x : Windows mobile etc.

**Single User Multi Task:** This is the most common type of operating system used today: Microsoft windows and Apple Macintosh are the living examples of this Genre. These can perform Multi tasking operations, like for example playing a multimedia file, downloading a file from the internet and editing a Text file simultaneously.

E x : Windows vista, Mac X tiger etc.

**Multi User :** This type of operating systems allow multiple users to use the system resources simultaneously. This should not be confused with the multi user accounts in windows or similar. the main difference being, the network administrator is the only actual user in Windows or Macs and one more difference being that in OS like Unix more than one user can simultaneously login while this is not possible in windows.

E x : Unix, Linux, Solaris etc..

### Q.3.(c) Working of Laser Printers.

(5)

**Ans. Refer Q.2(b) of paper Dec-2009**

### Section - B

**Q. 4 (a) Explain different types of programming language. How we can execute a program written in these languages ?**

(10)

**Ans. A programming language** can be defined as: "A language used for expressing a set of computer instructions (program)". Since this language is responsible for human-system communication, it consists of necessary symbols, characters and grammar rules which allow programmers to communicate with computers.

The programming languages can be divided into two major categories: low level and high level languages. The low level languages can be further divided into machine language and assembly language.

The high level languages, on the other hand, can be categorized in variety of ways. In onesense, the high level languages can be divided into Problem Oriented and Procedure Oriented language

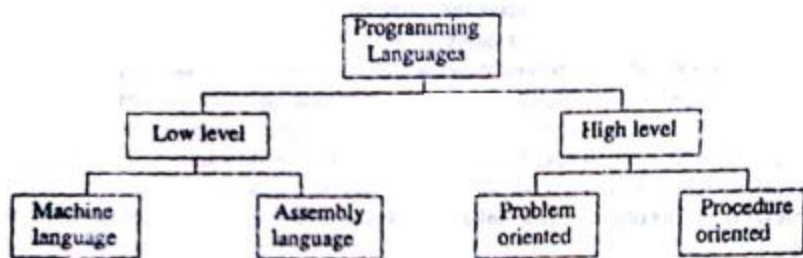


Fig.(a) : Programming languages

**Low Level Programming Languages :** Since the computer hardware works at binary level (i.e. 0s and 1s), it can only understand a binary based language. The binary based language is also called machine language because it is closest to the electronic machine i.e. the computer. An instruction written in this language is simply a string of binary digits.

The Assembly languages were developed in order to reduce the difficulty in writing machine language programs. An assembly language uses symbolic codes in place their equivalent binary codes.

The major difference between machine language and its corresponding assembly language lies in the manner in which both the languages refer to memory locations. The assembly language and the machine language work with symbolic and absolute memory addresses respectively. In fact, there is one to one correspondence between assembly language and machine language in the sense that for every binary code of machine language, there is an equivalent symbol (mnemonic) in assembly language.

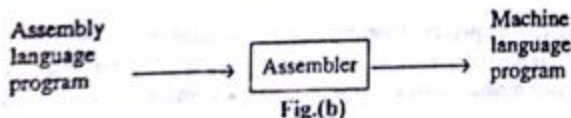


Fig.(b)

**High Level Languages :** It can be appreciated from the previous section, that program development in low level languages is a very slow and tedious process. Programming, especially, in machine language is error prone. Therefore, to overcome the limitations of low level languages, the used to develop programming languages that could make use of familiar English like words became necessary.

The high level languages are generally subdivided into "Procedure-Oriented Languages" and "Problem-Oriented Languages."

**Procedure-Oriented Languages :** A procedure-oriented language is normally computer independent. It is designed to help the programmer in the accurate description of procedures, logic of a problem in the form of a program. It is the responsibility of the programmer to express the logic of the problem into various procedures of the program.

Examples of procedure-oriented languages are : BASIC, FORTRAN, COBOL, PASCAL, 'C++' etc.

**Problem-Oriented Languages :** A problem-oriented language is designed to help the programmer in the accurate description of problems belonging to specific sets of problem types. The programmer does



not provide the procedure to be followed in solving the problem but simply describes the necessary input/output requirements and other parameters. The method of solution is built into the software that interprets the instructions of the Problem-Oriented language. This type of language is very useful in a situation where the programmers or the users can not solve the problems themselves.

Examples of Problem-Oriented Languages are: STRES, GPSS, COGO.

**Programming Language Execution Process :** Since a high level language (HLL) such as FORTRAN, COBOL, PASCAL etc. is English like, it requires a translator that can translate a program written in HLL into the machine language. The translator is a program that translates user's program written in one language (source) to another (target). A program written in an HLL is known as source program. The translator takes the source code and converts it into machine code. The machine code is also known as object code. The object code is linked by a linker to create a machine executable program as shown in Fig.(c).

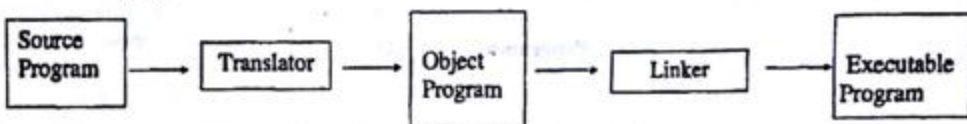


Fig.(c) : Translation process

The source program and executable programs are simply two different forms of the same thing. It may be noted here that the source program does not run in the computer but the translated executable copy of the program runs in the computer.

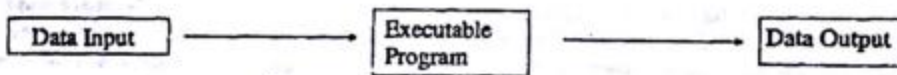


Fig.(d) : Execution process

The compiler can be precisely defined as: A compiler is a system program which translates a program written in a high level language into its equivalent program in a low level language such as assembly language or machine language. Example: Pascal compiler, Turbo C++ compiler etc.

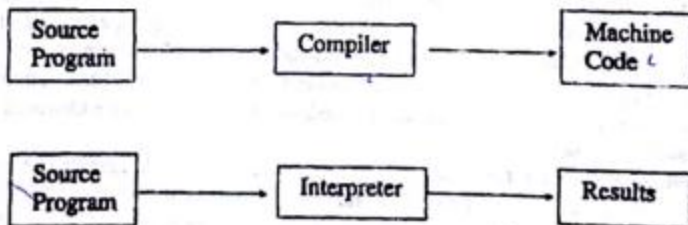


Fig.(e) : Interpreter operation

An interpreter is different from a compiler in the sense that it does not translate but interprets the source language and executes it. It can be defined as "The program execution environment". The operation of an interpreter is shown in fig. (e).

**Q.4.(b) Explain TCP/IP reference model.**

(10)

**Ans.** The TCP/IP model is a description framework for computer network protocols created in the 1970s by DARPA, an agency of the United States Department of Defense. It evolved from ARPANET, which was the world's first wide area network and a predecessor of the Internet. The TCP/IP Model is sometimes called the Internet Model or the DoD Model.

The TCP/IP model, or Internet Protocol Suite, describes a set of general design guidelines and implementations of specific networking protocols to enable computers to communicate over a network. TCP/IP provides end-to-end connectivity specifying how data should be formatted, addressed, transmitted, routed and received at the destination. Protocols exist for a variety of different types of communication services between computers.

Successive encapsulation of application data descending through the protocol stack before transmission on the local network link.

**Link Layer:** The Link Layer (or Network Access Layer) is the networking scope of the local network connection to which a host is attached. This regime is called the link in Internet literature. This is the lowest component layer of the Internet protocols, as TCP/IP is designed to be hardware independent. As a result TCP/IP is able to be implemented on top of virtually any hardware networking technology. The Link Layer is used to move packets between the Internet Layer interfaces of two different hosts on the same link. The processes of transmitting and receiving packets on a given link can be controlled both in the software device driver for the network card, as well as on firmware or specialized chipsets. These will perform data link functions such as adding a packet header to prepare it for transmission, then actually transmit the frame over a physical medium. The TCP/IP model includes specifications of translating the network addressing methods used in the Internet Protocol to data link addressing, such as Media Access Control (MAC), however all other aspects below that level are implicitly assumed to exist in the Link Layer, but are not explicitly defined.

**Internet Layer**

The Internet Layer solves the problem of sending packets across one or more networks. Internetworking requires sending data from the source network to the destination network. This process is called routing.

In the Internet Protocol Suite, the Internet Protocol performs two basic functions:

- Host addressing and identification: This is accomplished with a hierarchical addressing system (see IP address).
- Packet routing: This is the basic task of getting packets of data (datagrams) from source to destination by sending them to the next network node (router) closer to the final destination.

**Transport Layer:** The Transport Layer's responsibilities include end-to-end message transfer capabilities independent of the underlying network, along with error control, segmentation, flow control, congestion control, and application addressing (port numbers). End to end message transmission or connecting applications at the transport layer can be categorized as either connection-oriented, implemented in Transmission Control Protocol (TCP), or connectionless, implemented in User Datagram Protocol (UDP). The Transport Layer can be thought of as a transport mechanism, e.g., a vehicle with the responsibility to make sure that its contents (passengers/goods) reach their destination safely and soundly, unless another protocol layer is responsible for safe delivery.

For example, the Transmission Control Protocol (TCP) is a connection-oriented protocol that addresses numerous reliability issues to provide a reliable byte stream:

- data arrives in-order
- data has minimal error (i.e. correctness)
- duplicate data is discarded
- lost/dropped packets are resent



- includes traffic congestion control

**Application Layer :** The Application Layer refers to the higher-level protocols used by most applications for network communication. Examples of application layer protocols include the File Transfer Protocol (FTP) and the Simple Mail Transfer Protocol (SMTP). Data coded according to application layer protocols are then encapsulated into one or (occasionally) more transport layer protocols (such as the Transmission Control Protocol (TCP) or User Datagram Protocol (UDP)), which in turn use lower layer protocols to effect actual data transfer. Since the IP stack defines no layers between the application and transport layers, the application layer must include any protocols that act like the OSI's presentation and session layer protocols. This is usually done through libraries.

**Q.5.(a) Explain OSI reference model.**

(10)

**Ans.** The OSI, or Open System Interconnection, model defines a networking framework for implementing protocols in seven layers. Control is passed from one layer to the next, starting at the application layer in one station, and proceeding to the bottom layer, over the channel to the next station and back up the hierarchy. The OSI, or Open System Interconnection, model defines a networking framework for implementing protocols in seven layers. Control is passed from one layer to the next, starting at the application layer in one station, and proceeding to the bottom layer, over the channel to the next station and back up the hierarchy.

**Application (Layer 7) :** This layer supports application and end-user processes. Communication partners are identified, quality of service is identified, user authentication and privacy are considered, and any constraints on data syntax are identified. Everything at this layer is application-specific. This layer provides application services for file transfers, e-mail, and other network software services. Telnet and FTP are applications that exist entirely in the application level. Tiered application architectures are part of this layer.

**Presentation (Layer 6) :** This layer provides independence from differences in data representation (e.g., encryption) by translating from application to network format, and vice versa. The presentation layer works to transform data into the form that the application layer can accept. This layer formats and encrypts data to be sent across a network, providing freedom from compatibility problems. It is sometimes called the syntax layer.

**Session (Layer 5) :** This layer establishes, manages and terminates connections between applications. The session layer sets up, coordinates, and terminates conversations, exchanges, and dialogues between the applications at each end. It deals with session and connection coordination.

**Transport (Layer 4) :** This layer provides transparent transfer of data between end systems, or hosts, and is responsible for end-to-end error recovery and flow control. It ensures complete data transfer.

**Network (Layer 3) :** This layer provides switching and routing technologies, creating logical paths, known as virtual circuits, for transmitting data from node to node. Routing and forwarding are functions of this layer, as well as addressing, internetworking, error handling, congestion control and packet sequencing.

**Data Link (Layer 2) :** At this layer, data packets are encoded and decoded into bits. It furnishes transmission protocol knowledge and management and handles errors in the physical layer, flow control and frame synchronization. The data link layer is divided into two sub layers: The Media Access Control (MAC) layer and the Logical Link Control (LLC) layer. The MAC sub layer controls how a computer on the network gains access to the data and permission to transmit it. The LLC layer controls frame synchronization, flow control and error checking.

**Physical (Layer 1) :** This layer conveys the bit stream - electrical impulse, light or radio signal through the network at the electrical and mechanical level. It provides the hardware means of sending and



receiving data on a carrier, including defining cables, cards and physical aspects. Fast Ethernet, RS232, and ATM are protocols with physical layer components.

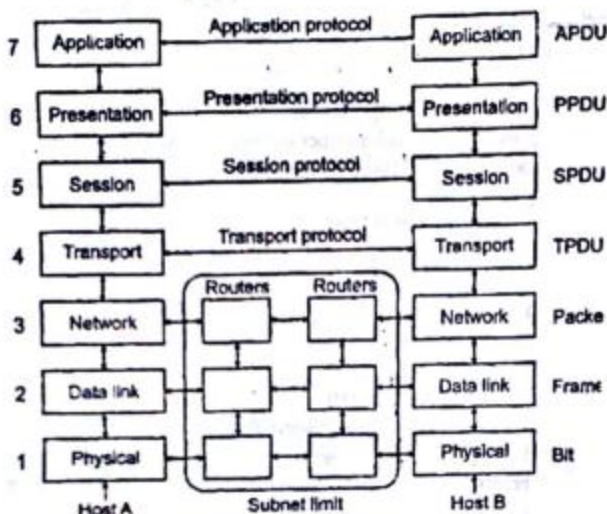


Fig. : Layers of OSI

**Q.5(b) Describe relationship between compiler, assembler, loader and linker.**

(10)

**Ans. Relation Between Compiler, Assembler, Loader and Linker :**

Normally C's program building process involves four stages and utilize different 'tools' such as a preprocessor, compiler, assembler, and linker.

At the end there should be a single executable file. Below are the stages that happen in order regardless of the operating system/compiler and graphically illustrated in Figure.

1. Preprocessing is the first pass of any C compilation. It processes include-files, conditional compilation instructions and macros.

2. Compilation is the second pass. It takes the output of the preprocessor, and the source code, and generates assembler source code.

3. Assembly is the third stage of compilation. It takes the assembly source code and produces an assembly listing with offsets. The assembler output is stored in an object file.

4. Linking is the final stage of compilation. It takes one or more object files or libraries as input and combines them to produce a single (usually executable) file. In doing so, it resolves references to external symbols, assigns final addresses to procedures/functions and variables, and revises code and data to reflect new addresses (a process called relocation).

- Bear in mind that if you use the IDE type compilers, these processes quite transparent.

- Now we are going to examine more details about the process that happen before and after the linking stage. For any given input file, the file name suffix (file extension) determines what kind of compilation is done and the example for GCC is listed in Table 5.6.

- In UNIX/Linux, the executable or binary file doesn't have extension whereas in Windows the executables. For example, may have .exe, .com and .dll.

### Section - C

**Q. 6.(a)** Write a program in 'C' which generate every 3rd integer between 2 and 100 and calculate the sum of those integers that are evenly divisible by 5. (10)

**Ans.**

```
#include <stdio.h>
void main()
{
    int c=0, i, j, d[100], sum=0;
    printf("Every 3rd number between 2 and 100: \n\n");
    for(i=2, j=0; i<=100; i+=3, ++j)
    {
        printf("%d\n", i);
        d[j]=i;
        ++c;
    }
    for(i=0; i<c; ++i)
    {
        if(d[i]%5==0)
            sum=sum+d[i];
    }
    printf("Sum of every 3rd number divisible by 5 is %d", sum);
    getch();
}
```

**Q. 6.(b)** WAP in 'C' to print the total count of all the prime numbers between 100 and 500. (10)

**Ans.**

```
#include <stdio.h>
void main()
{
    int i, j, flag, count=0;
    printf("Enter the range in which you want to count prime numbers: \n\n");
    printf("Lower Bound: 100");
    printf("\nUpper Bound: 500\n\n");
    for(i=100; i<=500; ++i)
    {
        if(i%2==0 || i%5==0 || i%7==0)
            flag=0;
        else
            count++;
    }
    printf("There are %d prime numbers in the range 100-500", count);
    getch();
}
```

**Q. 7.(a)** WAP in 'C' to find out the second smallest and second largest number in the list. (12)

**Ans.**

```
#include <stdio.h>
#define MAXSIZE 500
int elements[MAXSIZE], maxsize;
```

```

int main()
{
    int i, j, k, m, temp;
    printf("\nEnter the number of elements in your list: ");
    scanf("%d", &maxsize);
    printf("\nEnter the values one by one: \n");
    for (i = 0; i < maxsize; i++)
    {
        printf("\nEnter element %i :", i+1);
        scanf("%d", &elements[i]);
    }
    for (i = 0; i < maxsize-1; i++)
    {
        m = i;
        for (j = i+1; j < maxsize; j++)
        {
            if (elements[j] < elements[m])
                m = j;
        }
        temp = elements[i];
        elements[i] = elements[m];
        elements[m] = temp;
    }
    printf("\nSecond largest number is %d\nSecond smallest number is %d", elements[maxsize-2], elements[1]);
    getch();
}

```

**Q.7.(b) How structures within structures are implemented ? How we can access the members of these structures ?**

(8)

**Ans.** The members of a structure can be of any data type including another structure type i.e. we can include a structure within another structure. A structure variable can be a member of another structure. This is called nesting of structures.

```

struct tag1 {
    member1;
    member2;

```

```

.....
struct tag2 {
    member1;
    member2;

```

```

.....
    member m;
} var1;

```

```

.....
    member n;
} var2;

```

For accessing member 1 of inner structure we'll write-



var2.var1.member1

Here is an example of nested structure:

```
struct student {  
    char name [20];  
    int rollno;  
    struct date {  
        int day;  
        int month;  
        int year;  
    } birthdate;  
    float marks;  
} stu1, stu2;
```

Here we have defined a structure data inside the structure student. This structure date has three members day, month, year and birthdate is a variable of type struct date. We can access the members of inner structure as-

```
stu1.birthdate.day    → day of birthdate of stu1  
stu1.birthdate.month → month of birthdate of stu1  
stu1.birthdate.year   → year of birthdate of stu1  
stu2.birthdate.day     → day of birthdate of stu2
```

Here we have defined the template of structure date inside the structure student, we could have defined it outside and declared its variables inside the structure student using the tag. But remember if we define the inner structure outside, then this definition should always be before the definition of outer structure. Here in this case the date structure should be defined before the student structure.

The advantage of defining date structure outside is that we can declare variables of date type anywhere else also.

The nesting of structures can be extended to any level. The following example shows nesting at level three i.e. first structure is nested inside a second structure and second structure is nested inside a third structure.

#### Section - D

Q. 8 Explain the following:

- (a) Array of pointers.
- (b) Pointer of functions
- (c) Dynamic memory allocation
- (d) Command line arguments.

(20)

Ans. (a) Array of pointers : Like any other array, we can also have an array of pointers. Each element of such an array can point to a data item such as : variable, array etc. For example, consider the declaration given below:

Float \*x [20];

This declaration means that x is an array of 20 elements and each element is a pointer to a variable of type float. Let us now, construct an array of pointers to strings:

```
char *item [] = { "Chair",  
                  "Table",  
                  "Stool",  
                  "Desk",
```

The above declaration gives an array of pointer called item. Each element of item points to a string as shown in Fig.

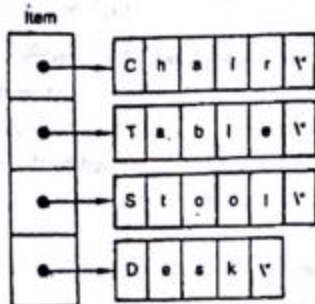


Fig. : Array of pointers to strings

The advantage of having an array of pointer to strings is that manipulation of strings becomes convenient. For example, the string containing "Table" can be copied into a pointer ptr without using strcpy() function by the following set of statements:

```
char *ptr; // declare a pointer to a string
```

```
ptr = item [1]; //assign the appropriate pointer to ptr
```

However, the changes made by the pointer ptr to the string "Table" will definitely disturb the contents of the string pointed by the pointer item[1] and vice versa. Therefore, utmost care should be taken by the programmer while manipulating pointers pointing to same memory locations.

**Ans. (b) Pointer of functions :** A function, like a variable, has a type and an address location in the memory. It is therefore, possible to declare a pointer to a function, which can then be used as an argument in another function. A pointer to a function is declared as follows:

```
type (*fptr) ( );
```

This tells the compiler that fptr is a pointer to a function, which returns type value. The parentheses around \*fptr are necessary. Remember that a statement like

```
type *gptr ( );
```

would declare gptr as a function returning a pointer to type.

We can make a function pointer to point to a specific function by simply assigning the name of the function to the pointer. For example, the statements

```
double mul ( int, int );
```

```
double (*p1) ( );
```

```
p1 = mul;
```

declare p1 as a pointer to a function and mul as a function and then make p1 to point to the function mul. To call the function mul, we may now use the pointer p1 with the list of parameters. That is,

```
(*p1)(x,y) /* function call */
```

is equivalent to

mul(x,y)

Note the parentheses around \*p1.

**Ans. (c) Dynamic memory allocation :** The memory allocation that we have done till now was static memory allocation. The memory that could be used by the program was fixed i.e. we could not increase or decrease the size of memory during the execution of program. In many applications it is not possible to predict how much memory would be needed by the program at run time. For example if we declare an array of integers-

```
int emp_no[200];
```

To overcome these problems we should be able to allocate memory at run time. The process of allocating memory at the time of execution is called dynamic memory allocation. The allocation and release of this memory space can be done with the help of some built-in-functions whose prototypes are found in alloc.h and stdlib.h header files. These functions take memory from a memory area called heap and release this memory whenever not required, so that it can be used again for some other purpose.

Pointers play an important role in dynamic memory allocation because we can access the dynamically allocated memory only through pointers.

**malloc()**

**Declaration:** void \*malloc(size\_t size);

**calloc()**

**Declaration:** void \*calloc(size\_t n, size\_t size);

**realloc()**

**Declaration:** void \*realloc(void \*ptr, size\_t newsize)

**free()**

**Declaration:** void free(void \*p)

**Ans. (d) Command line arguments :** Command line argument is a parameter supplied to a program when the program is invoked. This parameter may represent a filename the program should process. For example, if we want to execute a program to copy the contents of a file named X\_FILE to another one named Y\_FILE, then we may use a command line like

```
C> PROGRAM X_FILE Y_FILE
```

where **PROGRAM** is the filename where the executable code of the program is stored. This eliminates the need for the program to request the user to enter the filenames during execution.

We know that every C program should have one **main** function and that it marks the beginning of the program. But what we have not mentioned so far is that it can also take arguments like other functions. In fact **main** can take two arguments called **argc** and **argv** and the information contained in the command line is passed on to the program through these arguments, when **main** is called up by the system.

For instance, for the command line given above, **argc** is three and **argv** is an array of three pointers to strings as shown below:



argv[0] -> PROGRAM

argv[1] -> X\_FILE

argv[2] -> Y\_FILE

In order to access the command line arguments, we must declare the main function and its parameters as follows:

```
main(int argc, char *argv [])
{
    ....
    ...
}
```

The first parameter in the command line is always the program name and therefore argv[0] always represents the program name.

Q. 9 (a) Write a program to count the number of words, characters and lines in a file. (10)

Ans. #include <stdio.h>

main()

{

char line[81], ctr;

int i, c;

end = 0;

characters = 0;

words = 0;

lines = 0;

printf("KEY IN THE TEXT.\n");

printf("GIVE ONE SPACE AFTER EACH WORD.\n");

printf("WHEN COMPLETED, PRESS 'ENTER'.\n\n");

while( end == 0)

{

/\* Reading a line of text \*/

c = 0;

while((ctr=getchar()) != '\n')

line[c++] = ctr;

line[c] = '\0';

/\* counting the words in a line \*/

if(line[0] == '\0')

break;

else

{

```

        words++;
        for(i=0; line[i] != '\0'; i++)
            if(line[i] == ' ' || line[i] == '\t')
                words++;
    }
    /* counting lines and characters */
    lines = lines + 1;
    characters = characters + strlen(line);
}
printf("\n");
printf("Number of lines = %d\n", lines);
printf("Number of words = %d\n", words);
printf("Number of characters = %d\n", characters);
}

```

**Q.9. (b) WAP using function to clarify the concept of the two parameter passing methods : call by value and call by reference.** (10)

**Ans. Parameter Passing Method:** It is a mechanism through which arguments are passed to the called function for the required processing. There are methods of parameters passing.

1. call by value
2. call by reference

**Call By Value :** When the value of arguments are passed from the calling function to the called function, the values are copied into the called function. If any changes are made to the values in the called function, there is no change in the original values within the calling function.

The program illustrates a call by value parameter passing mechanism.

```

#include<stdio.h>
main ()
{
    int n1, n2, x;
    int call_by_val ( );
    n1 = 6;
    n2 = 9;
    printf("n1 = %d and n2 = %d\n", n1, n2);
    x = call_by_val(n1, n2);
    printf("n1 = %d and n2 = %d\n", n1, n2);
    printf("x = %d", x);
} /* End of main ( ) */

/* Function to illustrate call by value */

call_by_val(p1, p2)

```

```

int p1, p2;
{
    int sum;
    sum = p1 + p2
    p1 += 2;
    p2 += p1;
    printf("p1 = %d and p2 = %d\n", p1, p2);
    return (sum);
}

```

When this program is executed, the following information will be displayed.

```

n1 = 6    and    n2 = 9
p1 = 8    and    p2 = 17
n1 = 6    and    n2 = 9
x = 15

```

Note that there is no change in the values of n1 and n2 before and after the function execution.

**Call By Reference :** In this method, the actual values are not passed, instead their addresses are passed. There is no copying of values since their memory locations are referenced. If any modification is made to the values in the called function, the original values get changed within the calling function. Passing of addresses require the knowledge of pointers. Therefore, the call by reference mechanism will be discussed in later chapter.

This program accepts a one-dimensional array of integers and sorts them in ascending order. This program involves passing of the array to the function.

```

#include<stdio.h>
main ( )
{
    int num [20], i, max;
    printf("Enter the size of the array\n");
    scanf ("%d", &max);
    printf("Start Entering the number\n");
    for (i = 0; i < max; i++)
    {
        scanf ("%d", &num[i]);
    }
    sort_nums (num, max); /*Function Reference*/
    printf("Sorted numbers are as follows\n");
    for (i = 0; i < max; i++)
        printf ("%3d", num [i]);
} /*End of main ( ) */
/* Function to sort list of numbers*/

void sort_nums(a, n)
int a [ ], n;

```



```

{
    int i, j, dummy;
    for (i = 0; i < n-1; i++)
    {
        for (j = i+1; j < n; j++)
        {
            if(a[i] > a[j])
            {
                dummy = a[i];
                a[i] = a[j];
                a[j] = dummy;
            } /*End of if*/
        } /*End of j for loop*/
    } /*End of i for loop*/
    return;
} /* End of function*/

```

output :

```

Enter the size of the array
5
Start entering the numbers
44
64
16
89
8
Sorted number are as follows
8
16
44
64
89

```

Since, an array is passed to the function, only the name of the array (which itself indicates the starting address of an array) must be written in the argument list. Square brackets should not be used with the name of the array at the point of function reference. But, in function definition the name of the array must be specified with an empty square brackets. This is shown in the example program with bold face.

