Building Concepts in COMPUTER SCIENCE ✦ A textbook for class 10

BOARD OF SECONDARY EDUCATION
"तमसो मा ज्योतिर्गमय"
ASSAM

Building Concepts in COMPUTER SCIENCE

A textbook for class

10


PC-WAR
14.31818

Building Concepts in COMPUTER SCIENCE

A textbook for class

10

Authors -

Dr. Manojit Ghose

Mr. Chandan Kalita

Mr. Tarak M Karmmakar

Ms. Abha Goswami

Ms. Jiri Barman

Co-ordinator : Dr. Nityajyoti Kalita, Academic officer, SEBA

Building Concepts in COMPUTER SCIENCE : A textbook of Building Concepts in Computer Science (English Medium) for Class- X, prepared and approved by Board of Secondary Education, Assam, which is introduced from the Acedemic Year 2022-23 and published by the Assam State Textbook Production and Publication Corporation Limited, Guwahati-1 on behalf of Govt. of Assam.

FREE TEXTBOOK

First Publication : 2021

Printed on : 70 GSM Paper

Published by : The Assam State Textbook Production and Publication Corporation Ltd., on behalf of Govt. of Assam for free distribution.

Printed at : Anurag Printers
Garchuk, Guwahati.

Dr. Ranoj Pegu, MBBS.
Minister, Assam

सत्यमेव जयते

Education, Welfare of Plain
Tribe & Backward classes

MESSAGE

Textbooks are the key components of scholastic education. Students gain knowledge that flows through the pages of the textbooks. Students, enriched with knowledge, are the wealth and future of our State as well as our Nation. The Renaissance of human civilization had been influenced by Education only. With this realization and foresight, the present Government has accorded the highest priority to the development of the Educational sector in the State.

The present State Government, in a bid to complement the untiring efforts of the student community to achieve success and fulfil ones' ambition and contribute to the welfare of the State, has been implementing various educational assistance programmes under "Pragyan Bharati". Under this programme, Free Textbooks are being distributed to the students from Pre Primary, Class I to Class XII which is now extended up to degree level since 2020. The benefits have been further augmented by waiving admission fees for the students of Secondary and Degree level. The State Government has also waived examination fees at the Secondary level for the students from the economically weaker section. Free uniforms are also being provided to the students up to the High School level. Under the "Anundoram Borooah Award Scheme", free laptops and at times, substituted with cash rewards, are being provided to meritorious students who excel in the HSLC examinations.

The Free Textbook component of the noble programme -"Pragyan Bharati" comprises printing, publication and distribution and, it is implemented with concerted efforts of the Assam State Textbook Production and Publication Corporation Limited (ASTPPC Ltd.), State Council of Educational Research and Training (SCERT), Board of Secondary Education, Assam (SEBA) and Assam Higher Secondary Education Council (AHSEC). I thank all these wings of the education department and appreciate their sincere hard work to make the scheme successful. My best wishes remain to all the Students - the human resources of the Nation for relentlessly pursuing their studies with all dedication and sincerity to gain a successful foothold to a bright future.

(Dr. Ranoj Pegu)
Education Minister, Assam

Foreword

Board of Secondary Education, Assam has been following the recommendations of National Curriculum Framework (NCF), 2005 for adopting/preparing textbooks of classes IX and X since 2013. Looking at the demand of the present society and also at the course curriculam of higher classes. Several books were prepared and those have been prescribed for class IX and X in Board's curriculum. Recently, it had been decided to bring about some reforms in the existing textbook of Computer Science. As such a new textbook of Computer Science was prepared and introduced from the Academic Session 2021-2022 for class IX students so that they can have the opportunity to study about the latest developments in the subject. As a part of this change this new textbook of class X is prepared and it has been introduced from the academic session 2022-2023. I am thankful to all the authors of this textbook for doing their level best in preparing this textbook which helped us to publish the same in time. Moreover, it is a humble request to all the teachers/subject experts/guardians to put their views and suggestions and point out errors, if any, so that we can take necessary steps for its corrections and present a correct version thereafter. The suggestions may be mailed at **sebatxbsuggest@gmail.com.**

(N.N. Nath, ACS)
Secretary
Board of Secondary Education, Assam
Guwahati

Preface

We live in the era of a digital revolution. Nowadays, computer science has touched almost all aspects of our lives. As a result, basic computer knowledge has become one of the fundamental needs of human lives today. In parallel, computer education has also received enormous importance from the school to the university level. Research and developments in this field also provide a lot of opportunities along with throwing many interesting challenges.

Acquiring sufficient knowledge in this subject will definitely empower our students to grasp several opportunities offered by the ongoing digital revolution. The students can not only take the benefits for themselves, they can also help their parents and others who are not well-equipped with the recent digital technologies. Computer science knowledge will also help the students to draw maximum benefits from several innovative projects of the Govt. such as the Atal Tinkering lab, the Robotics lab, ICT in Schools, etc. Learning computer science will indeed add new skill-sets to our students. This in-tern will surely open new paths for their carriers. Learning and practicing coding at an early age will also train the young minds how to think and how to think logically in particular!

The book “Building Concepts in Computer Science” is written in simple English and the topics are explained in a lucid manner so that students studying in a vernacular medium can understand them without any difficulty. Students have already acquired introductory knowledge about the subject in Class IX. Our primary focus in this book is to give deeper insights into the topics and to build a solid concept about the subject.

The book has 11 chapters in total. Chapter 1 (written by Chandan Kalita) gives an introduction to computer networks where one computer can communicate with another computer. In Chapter 2 (written by Tarak M Karmmakar), we introduce HTML and CSS. Chapter 3 (written by Abha Goswami) deals with databases and it uses MySQL for the practicals. We have introduced the C programming language in class IX. Chapters 4, 5, 6, 7, 8, and 9 of this book (written by Manojit Ghose) give a deeper and in-depth knowledge about C language and coding in general. In Chapter 10 (written by Jiri Barman), we introduce a new programming paradigm - object-oriented programming. At last, Chapter 11 (written by Tarak Karmakar) concludes the book with some case studies related to the daily applications of computer science.

While writing the C programs in the book, we have not explicitly mentioned about the software (compiler, editor, etc.) as we have already discussed the same in the Class IX book. Every C programming chapter starts with a small program and then new features and concepts

of C language are added and the programs are extended. As a result, the program length increases towards the end of a chapter. **We request all the teachers to note the fact that there can be multiple solutions or approaches for a given problem in the field of coding. Thus, they should always keep this in mind while evaluating the answers from the students.**

We understand that students may make silly mistakes when they write programs. Keeping this in mind, we are making all the C programs used in the book available online on ***www.manojitghose.com/seba.*** Students and the teachers can download the C programs along with some additional resources.

Finally, we would like to thank the Honourable Chairman, Secretary and other officials of SEBA for providing us an opportunity to author this book. Suggestions, comments, feedback, and criticism for the authors may kindly be emailed to authors.assam@gmail.com.

- Dr. Manojit Ghose
- Mr. Chandan Kalita
- Mr. Tarak M Karmmakar
- Ms. Abha Goswami
- Ms. Jiri Barman

Contents

CHAPTER 1

Introduction to Computer Networks

In this chapter

We will discuss the basics of Computer Networks. This includes- types of computer networks and their limitations, Computer Addresses, different types of connecting devices, commonly used network protocols, and some basic networking commands.

1.1 COMPUTER NETWORK

A computer network is a set of computers and other units that are connected with each other. In a computer network, information or resources can be shared among connected computers. For example, if there is a printer connected to the network, that printer can be used by all computers of the network. The interconnection of computers in a network is made up of telecommunication network technologies, based on physically wired, or wireless radio-frequency methods. There are different wired and wireless technologies. Coaxial cable, Twisted pair cable, Optical Fiber Cable, etc. are some examples of wired technology. On the other hand, Bluetooth, Wi-Fi, WiMAX, etc. are some examples of wireless technologies.

1.1.1 Types of Computer Network

There are different types of computer networks. Based on the connecting technology, a computer network can be classified as a wired, wireless, or hybrid network. Based on the size of the network, all types of computer networks (wired, wireless, and hybrid) can be further classified into mainly four different types - PAN (Personal Area Network), LAN (Local Area Network), MAN (Metropolitan Area Network), line space and WAN (Wide Area Network).

Personal Area Network (PAN)

Personal Area Network is the simplest computer network. Personal Area Networks are normally set up for individual use within a limited geographical range. In normal circumstances, Personal Area Network offers a network range of about 10 meters from a central device to the other communicating devices. For example, someone can set up a Personal Area Network between his laptop and other electronic gadgets like a smartphone, PDA, tablet, printer, etc. Figure 1.1 gives an idea of a typical Personal Area Network.

Figure 1.1: Personal Area Network (PAN)

Local Area Network (LAN)

Local Area Network is the most commonly used network. In LAN, a number of units are connected to each other in a small area such as a building, office, etc. with the help of a common communication path like a switch or a similar connecting device. You can observe such networks in banks, cyber cafes, or even in your school's laboratory. Most of the time this type of network is set up to share common facilities like internet connection, network printer, etc. For example, if there is a LAN in your school and all computers of your school's laboratory are connected to the LAN, only one internet connection can be shared by all computers of your school. In such a scenario, there is no need for individual internet connections for each computer to access the internet. Similarly, if a network printer is connected to the LAN, you can print a page from any computer at your school. Figure 1.2 gives an idea of a typical Local Area Network.

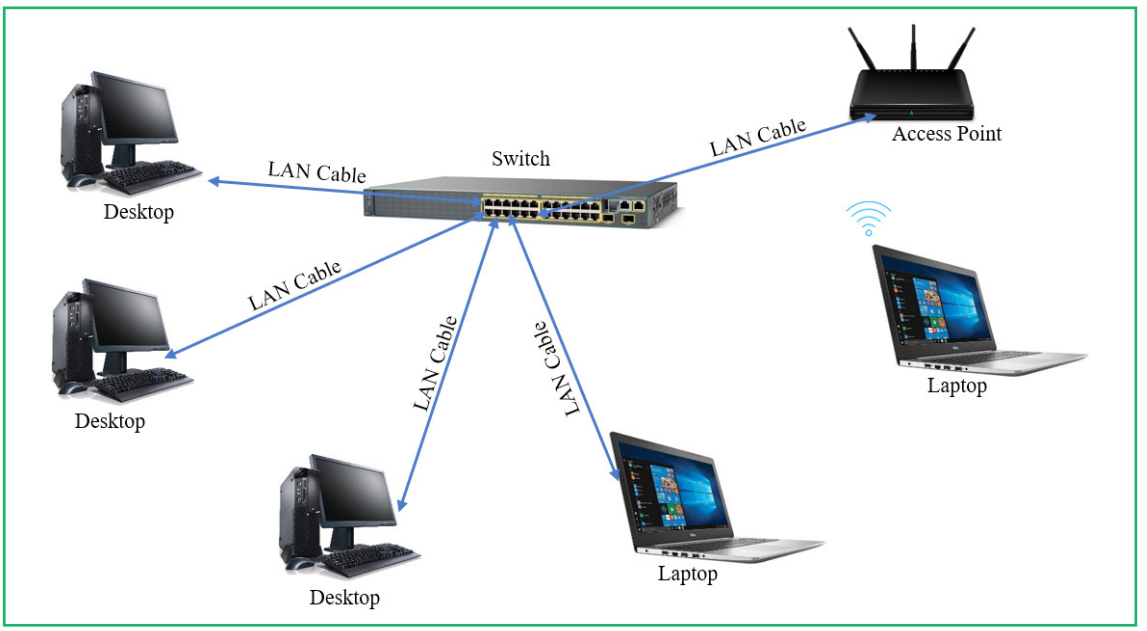


Figure 1.2: Local Area Network (LAN)

Metropolitan Area Network (MAN)

MAN or Metropolitan Area Networks are normally spread over an entire city or a wide geographic location. This type of network covers a larger area than that of a LAN. Although the term Metropolitan refers to urban areas or towns, a Metropolitan Area Network does not have to be in urban areas only; here, the term metropolitan implies the size of the computer network, not the class or type of the area that it serves. A Metropolitan Area Network is formed by interconnecting multiple LANs that are within a limited geographic area. MANs typically merge the networks of multiple organizations to establish communication among them or to share common resources. Most of the time Internet Service Providers set up MAN to provide a high-speed internet connection to multiple organizations. One such scenario is illustrated in Figure 1.3 where, a college LAN, an office LAN, and a hospital LAN are connected by an Internet Service Provider (ISP) to offer internet connection to all three LANs that are at different locations. Most of the time the physical connection from one organization to another organization is established using Optical Fiber Cable to achieve good quality of service.

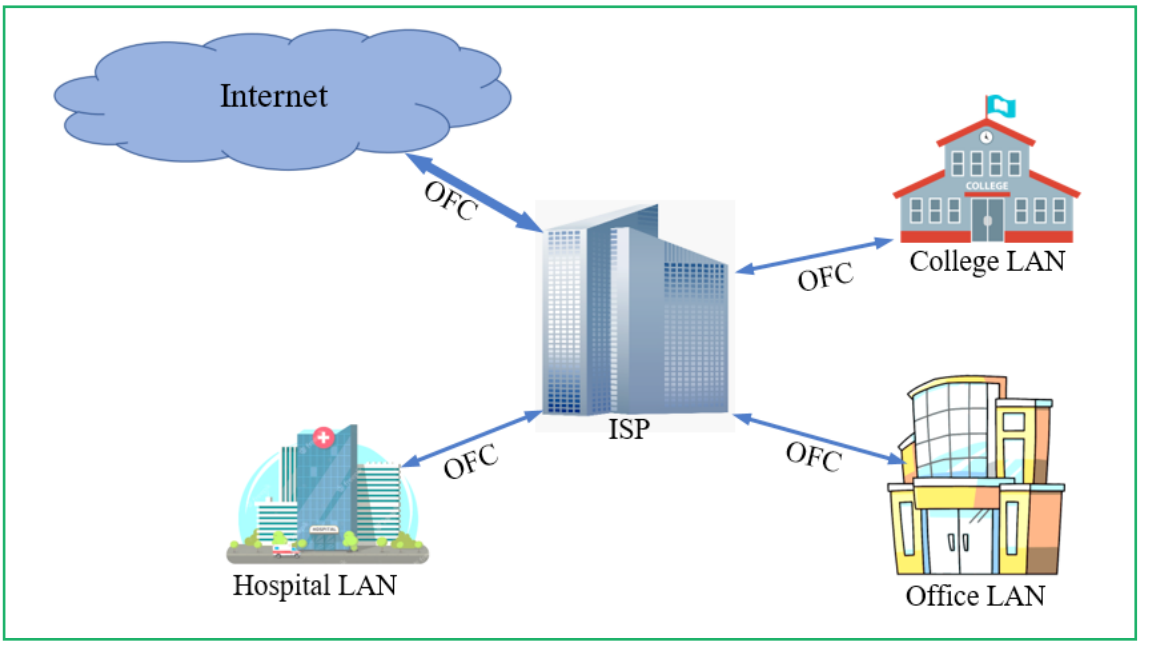


Figure 1.3: Metropolitan Area Network (MAN)

Wide Area Network (WAN)

Wide Area Networks (WAN) are formed by connecting devices from multiple locations across the globe. Wide area networks are often established with Leased Telecommunication Circuits (Leased Telecommunication Circuits is a communication line between two or more locations functioning according to a commercial contract). In comparison to a Metropolitan Area Network, a Wide Area Network is wider with respect to geographical location. The Internet is an example of WAN since it is spared over the world. Figure 1.4 gives an idea of a typical Wide Area Network.

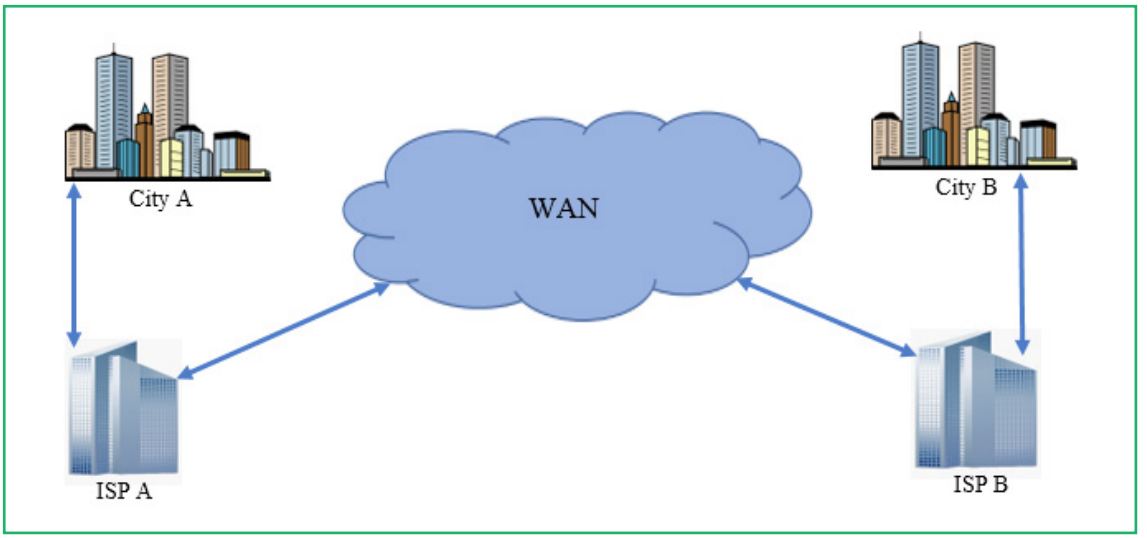


Figure 1.4: Wide Area Network (WAN)

PAN	LAN	MAN	WAN
PAN's ownership is private. Normally, one PAN is owned by a single person.	LAN's ownership is private. i.e. everyone can create their LAN by connecting multiple devices.	MAN's ownership may or may not be private. i.e. MAN may not be owned by one organization.	WAN's ownership is not private. Most of the time, WANs are owned by multiple organizations.
The data transfer rate of a PAN is very high.	The data transfer rate of a LAN is slower than PAN	The data transfer rate of a MAN is slower than LAN	The data transfer rate of a WAN is slower than MAN
There is no congestion in PAN since data lines are not shared.	There may be a little congestion in LAN. The congestion depends upon the number of communicating devices.	Normally the congestion in MAN is more than that of LAN.	In WAN, there is more congestion than MAN, LAN, and PAN
There is no design issue in PAN and maintenance is easy.	LAN's design and maintenance are easy.	MAN's design and maintenance are more difficult than LAN.	WAN's design and maintenance are more difficult than LAN as well MAN.

Table 1.1: A brief comparison between different types of computer networks.

1.2 ADDRESS IN COMPUTER NETWORKS

If there are hundreds of connected computers in a computer network, how to distinctly identify a particular computer? Here identifying a computer does not mean identifying a computer physically. Sometimes you may need to identify a computer A in your laboratory from another

computer B to use A. Let us take an example to understand the idea of identifying a computer in a computer network. Suppose you want to transfer some computer files from computer A to another computer B. You can do that using some external storage device like a pen drive, CD, DVD, etc. But this method needs you to visit both computers physically. Sometimes physical access to the required computer may not be possible. To overcome this problem, you can copy the files from computer A to computer B using the computer network. Since the network may have many computers, you need to know the network addresses of both the computers. A network address should uniquely identify a network node or a device in a computer network. We can refer to a computer in a network by its hostname, IP address, or MAC address.

1.2.1 Hostname

The hostname is the name of a computer or any connected device in a computer network. Some people called it **computer name** also. The hostname can be used to distinguish a device within a computer network. You can find the hostname of your computer in the command prompt in the Windows operating system by the command **"hostname".** The output of the hostname command is shown in Figure 1.5. The hostname command works in Linux as well as in Mac operating system also. There are provisions to change the hostname of a computer. The procedure to change the hostname of a computer varies from OS to OS. In windows10 there is an option "Rename PC". By navigating to this option, we can easily change the hostname of our Windows 10 computer. In Linux operating system, there is a file "hostname" in the "/etc" directory, which contains the hostname of the computer. To change the hostname of such a system we have to modify the content of the file.

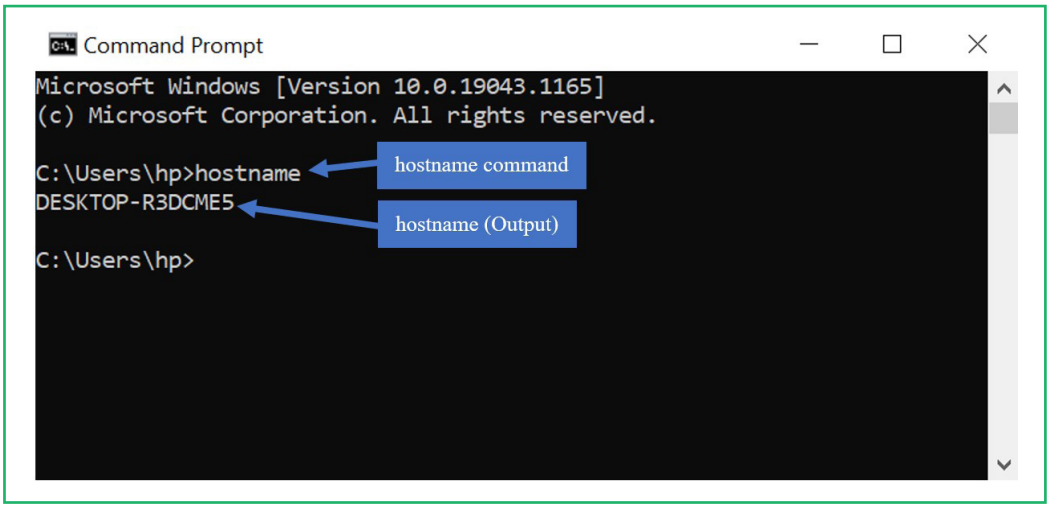


Figure 1.5: Output of the hostname command.

1.2.2 IP Address

Although a hostname can be used to identify a computer in a computer network, hostnames may not be unique always. It means, there is a possibility that two or more computers in a computer network may have the same hostname. Because as mentioned in the previous section,

there are options to change the hostname of a computer independently. If two or more users assign the same hostname to their computers intentionally or unintentionally, it is become difficult to identify a computer. Therefore, if the hostnames are not initialized centrally, the hostname should not be considered as a unique identifier of a computer in a computer network.

An Internet Protocol address (IP address) is a numerical label that is assigned to network devices to identify them uniquely. There are two types of IP address- Internet Protocol Version 4 (IPv4) and Internet Protocol Version 6 (IPv6). The length of an IPv4 address is 32 bit whereas the length of an IPv6 is 128 bit. IPv4 addresses are most often written in dot-decimal notation (See Figure 1.6) which contains four segments of eight-bit each (octets). For example, 172.16.254.2 is an example of an IPv4 address.

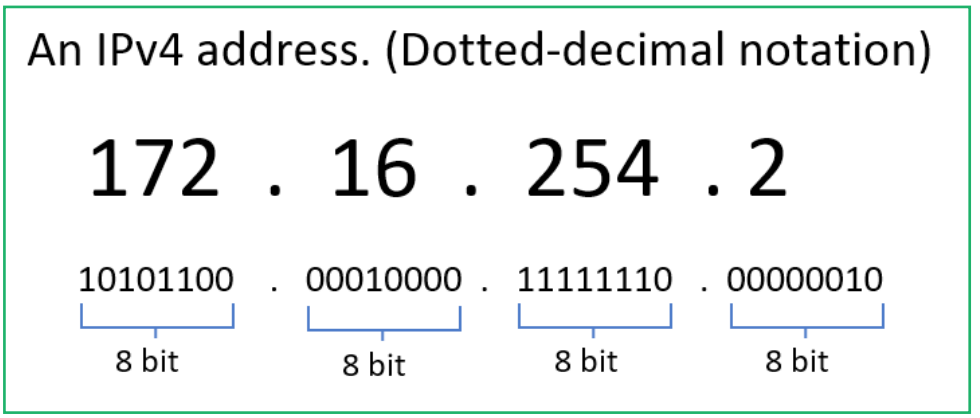


Figure 1.6: An example of an IPv4 address.

Unlike IPv4 addresses, IPv6 addresses are represented as eight groups of four hexadecimal digits of 16 bits each, where groups are separated by colons (See Figure 1.7).

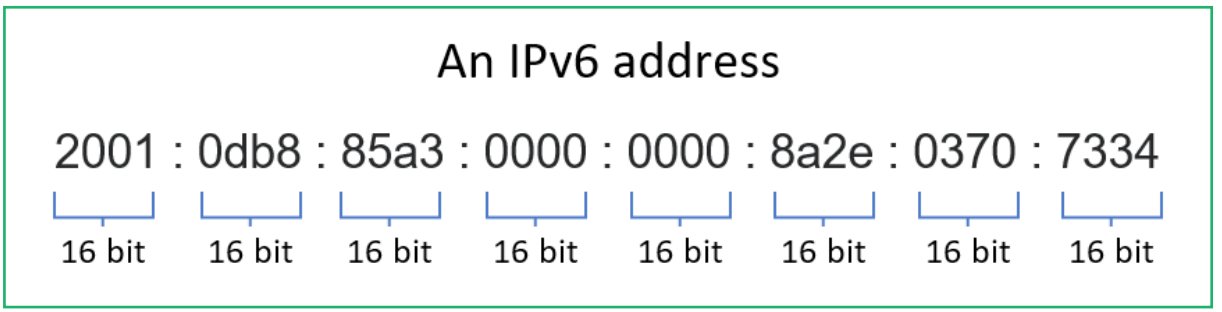


Figure 1.7: An example of IPv6 address.

We have discussed two types of IP addresses and their representation. Now, what is the use of these addresses? Should we always assign an IP address to a computer? To answer this question let us go back to the previous problem of identifying a computer in a computer network. A physical connection to a computer using a network cable does not necessarily connect the computer to a computer network. A computer is considered as connected to a network only if the computer can send or receive data over the network. To transfer data by a computer, besides the

physical connection the computer should have a proper IP address. If one IP address is assigned to more than one computer, only one computer can establish the connection and the remaining computer shows an "IP conflict" error message. It means, to successfully connect a computer to a computer network, each computer must have a unique IP address. Therefore, IP addresses can be considered as unique identifiers of a computer in a computer network.

IP address to a computer can be assigned by a user manually or the computer can request a server to assign an automatic IP address. The procedure to assign an IP address to a computer by a user varies from OS to OS. Most of the operating systems provide a user-friendly graphical user interface (GUI) to assign an IP address. However, in some server-centric operating systems users need to modify some system files to assign an IP address. To acquire an automatic IP address, there should be a server called DHPC (Dynamic Host Configuration Protocol) in the network. DHPC server maintains a pool of IP addresses and based on a request, it assigns an IP address from the pool to a computer. Assignment of IP address using a DHCP server overcomes the problem of IP conflict in a computer network.

Now we know that the IP address of a computer can be changed. Then how can we claim that IP addresses can be considered as a unique identifier of a computer in a computer network? Actually, this statement is true until the IP address is changed by a user. To physically identify a computer, we cannot use the IP address.

1.2.3 MAC Address

A Media Access Control address (MAC address) is a unique identifier assigned to a Network Interface Controller (NIC) (you can consider NIC is the part where the network cable physically connects). MAC addresses are represented as six groups of two hexadecimal digits, separated by hyphens, colons, or some time without any separator. MAC addresses are assigned by hardware manufacturers and cannot be changed by a user. Therefore, MAC addresses are also called burned-in addresses, or Ethernet hardware addresses, or Ethernet physical addresses. Since MAC address is assigned by hardware manufacturers and a user cannot modify it, we can use MAC address as a unique identifier of a computer in a computer network. Now the question is there are many such hardware manufacturers who manufacture NIC. How does different manufacturer maintain the uniqueness of MAC address among them? To understand the answer, we should know the structure of a MAC address. As shown in Figure 1.8, the MAC address is composed of twelve hexadecimal digits. Out of which the group of first six hexadecimal digits is called OUI (Organizational Unique Identifier). A globally recognized committee IEEE Registration Authority Committee assigns unique OUI to hardware manufacturers to use it as a prefix of MAC address. The uniqueness of the remaining six hexadecimal digits is maintained by each manufacturer individually. Thus, MAC addresses become globally unique.

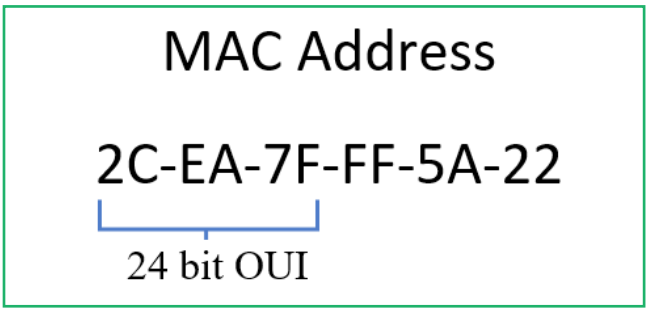


Figure 1.8: An example of MAC address.

1.3 NETWORK DEVICES

Network devices are physical devices that are required to establish a computer network so that multiple computers can communicate or transfer data among them. There are different types of network devices each of them has a specific purpose. In this section, we discuss some commonly used network devices.

1.3.1 Network Interface Card

A Network Interface Card (NIC) is a hardware component that is used to connect a computer to a computer network. Without NIC a computer cannot be connected to a network. Network Interface Cards are also called network adapters or LAN adapters. Different types of NIC are available in the market. Some of them can communicate with a wireless network and some of them can communicate with a wired network. Wireless NIC uses an antenna to provide wireless reception to use Wi-Fi or other wireless technology as a communication medium. Based on the installation type a NIC can be either called external or internal. External NICs are normally connected using the USB port on the other hand internal NICs are installed inside the CPU on a motherboard slot.

Figure 1.9: Different types of Network Interface Cards.

1.3.2 Hub and Switch

A Hub is a networking device that connects other computer networking devices together. When a signal traverses a long distance, the strength of the signal decreases gradually. In such

a situation, a hub can be used to act as a repeater which amplifies signals that deteriorate after traveling long distances over connecting cables. The functionality of a hub is very limited and simplest in comparison to the other networking devices. It simply connects LAN components with identical protocols. Hubs do not perform any filtering or addressing with the incoming data; instead, a hub just sends all incoming data to all other connected devices.

A Switch is also a networking device that connects other computer networking devices, but Switches are generally more intelligent than hubs. A switch is a multiport device that is very much similar to a hub in appearance, but unlike hubs, a switch maintains some routing information about other network devices in the internal network. A switch can read the destination address of incoming data and with the help of the routing information, it can transmit incoming data to the appropriate destination. The appearance of a hub or switch and how it is connected with other networking devices has shown in Figure 1.2.

Hub and Switch both have connecting ports. Computers and other devices are connected to these ports using a connecting cable. Two eight-pin connectors called RJ45 are clipped at both ends of a twisted pair cable to make a connecting cable. Different Switch may have different numbers of connecting ports. Based on the number of connecting ports, a Hub or Switch can connect different numbers of computers. For example, a 16 port Switch can connect 16 computers together. To connect more than 16 computers in a network, we need either a larger switch or more numbers of 16 port Switches.

1.3.3 Router

In the previous section, we have discussed about Hub and Switch. They are basically used to form a computer network by interconnecting many network devices like computers, other switches, etc. but, a Router is not similar to a Hub or a Switch. A router is another type of network device which are used to interconnect two or more heterogeneous computer networks. A router can be considered as a small computer that is usually dedicated to special-purpose, with multiple input and output network interfaces. A Router maintains a chart of network id and output port. When data is received, a Router reads the destination address of incoming data and transmits it to their destinations by following a path through many interconnected networking devices. Routers store information about the networks in which they are connected. This information is used by a Router to transfer incoming data to a destination network. A router also contains some information about all connected networks and devices to which incoming data should be forwarded if the destination address is not known. A router can also be used to divide a computer network into many subnetworks.

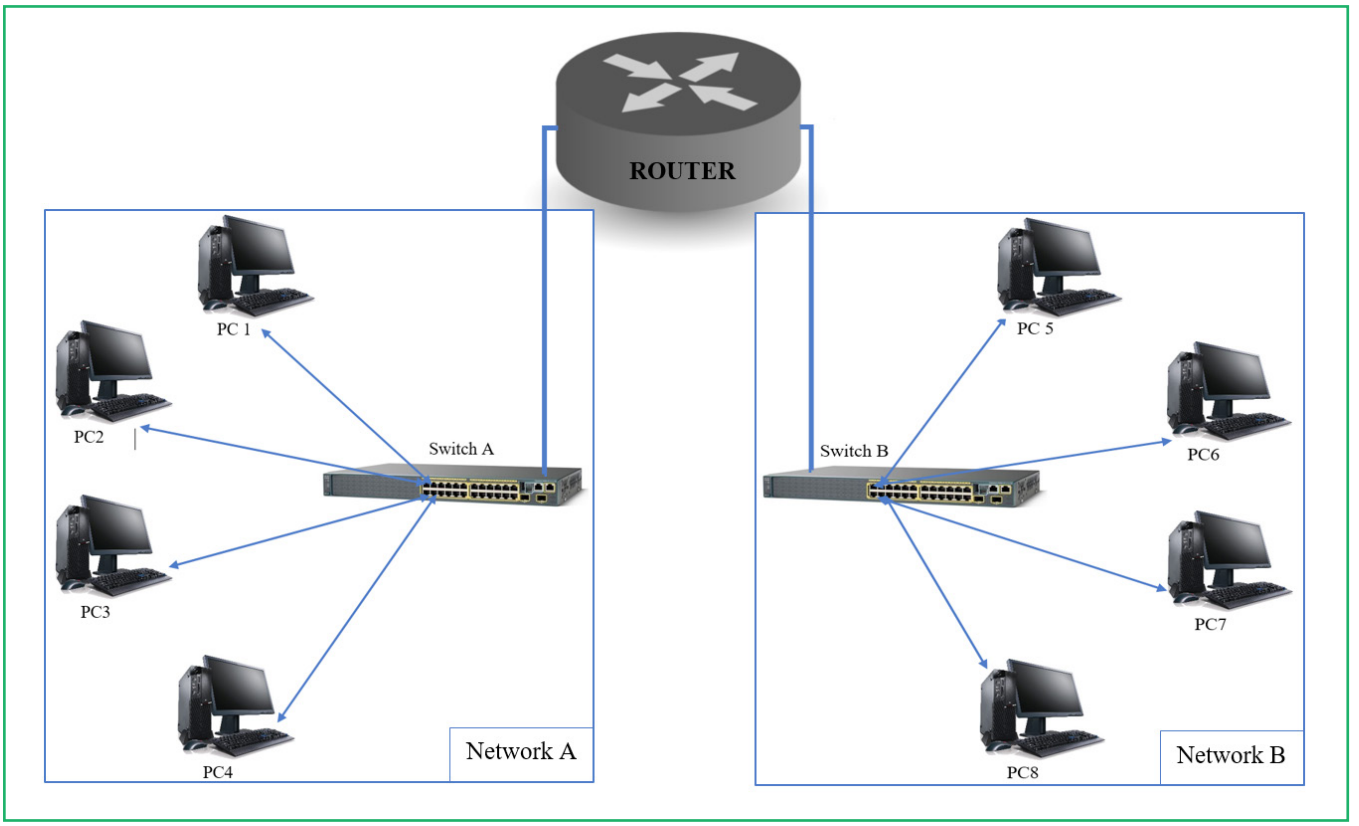


Figure 1.10: Use of a router to connect two computer networks.

1.3.4 Access Point

An access point is a network device that creates a wireless network. Computer or other devices that have a wireless network interface card can connect to such a wireless network. Although the main functionality of the access point is to provide wireless network connectivity, some access point also has several ports to expand the network to support more wireless devices. Each access point has a limitation of its transmission range. The range of an access point depends upon the power of access point antennas, obstructions, and environmental conditions between the wireless device and the Access Point, etc.

When an Access Point is installed in an area, normally the Access Point broadcasts the Service Set Identifier (SSID) name. A service set identifier is a sequence of characters that refers to a wireless local area network. To connect to a wireless network created by an Access Point, the user needs to know the SSID of the Access Point. An Access point can create a wireless network with or without a password-protected SSID. To connect to a password-protected SSID a user should provide the password after selecting the SSID. Access Point also can be configured to create a wireless network with an invisible SSID, which means that a network administrator needs to provide the SSID to a user instead of allowing it to be discovered automatically. In Figure 1.2 you can see how an Access Point is connected to a wired network and a laptop computer can connect to the Access Point using wireless technology.

1.4 COMMUNICATION PROTOCOLS

A network protocol is a set of well-established rules that determine how two computers communicate or transfer data between them. To establish communication between two computers, both computers must understand each other's methods of communication. But in both computers, there may be some differences in their internal processes, structure, or design. This is similar to two persons speaking in different languages. To understand each other, both people must use a common language. Network protocols are also like a common language for computers. Network protocols make it possible for network devices to communicate with each other because of predetermined rules which are built into devices' software and hardware.

1.4.1 TCP

The TCP (Transmission Control Protocol) is a communication protocol that is primarily used for establishing communication between computers over a computer network. When data is sent over a network, the TCP in the source computer divides those data into a series of packets and sends those packets to the destination and the TCP at the destination reassembled them.

1.4.2 IP

The IP (Internet Protocol) is mainly designed for addressing packets. The Internet Protocol is mostly used with TCP. The primary responsibility of Internet Protocol is to deliver packets from a source host to a destination host based on the IP address presents in a packet. TCP/IP is the broadly used protocol connecting computer networks.

1.4.3 POP and SMTP

POP stands for Post Office Protocol and SMTP stands for Simple Mail Transfer Protocol. The POP is designed for receiving incoming E-mails. On the other hand, the SMTP is designed to send outgoing E-Mail.

1.4.4 HTTP and HTTPS

The HTTP (Hypertext Transfer Protocol) protocol is the foundation of the World Wide Web which is used for fetching web resources such as HTML documents or other web pages. HTTP is a client-server protocol which means, a client requests a web resource using HTTP protocol and a web server provides that resource to the client.

The HTTPS (Hypertext Transfer Protocol Secure) is a combination of the Hypertext Transfer Protocol (HTTP) with added security features. By looking at the Uniform Resource Locator (URL)

in the web address bar of your browser you can easily identify whether you are browsing using HTTP or HTTPS. As shown in Figure 1.11, the protocol is mentioned at the beginning of the URL you are using. The main difference between HTTP and HTTPS is- in an HTTP communication data are sent without **encryption**, but in HTTPS, data are sent after **encryption**. The meaning of encryption is the process of converting information or data into a code, to prevent unauthorized access. Encrypted data can be again decrypted to get back the original data.

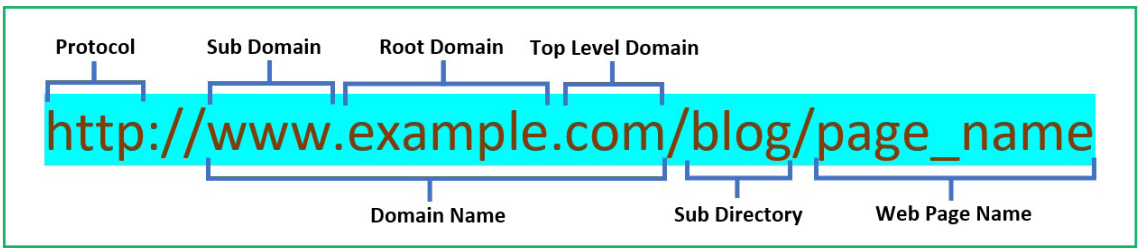


Figure 1.11: Example of URL.

1.4.5 FTP

The FTP (File Transfer Protocol) is a standard internet protocol used for transmitting files from one computer to another over a network. To transfer files using FTP protocol, one computer must allow the other computer to send or receive files. In the early days, FTP client applications were command-line programs. To transfer using such an FTP client, a user had to log into the server and execute FTP commands. Such applications are still available with Microsoft Windows and Linux operating systems. Using FTP commands, a user can move around different directories of the server, download files, upload files, delete files, etc.

The main disadvantage of FTP protocol is that it does not encrypt username, password, and data when logging in to the server and during data transfer. To overcome this disadvantage two new protocols SFTP and FTPS were developed. SFTP and FTPS both protocol uses encryption and therefore it is considered as secure protocols to transfer files.

These file transfer protocols (FTP, SFTP, and FTPS) are now commonly used to store files in **cloud storage**, which is normally a secure and ever-ready location that is held remotely. To transfer files there are many third-party client software. Some examples of such clients that are free to download include FileZilla Client, FTP Voyager, WinSCP, Core FTP, etc.

1.5 BASIC NETWORKING COMMANDS

To set up or diagnosis a computer network, a user should be familiar with some basic networking commands. These commands should be entered in the command prompt of the operating system. In this section, some such commands are mentioned.

- **ping :** This is the most common networking command. This command is used to verify the connectivity between two computers. The syntax of the command is-

ping <target host>

Here the <target host> is the target host's name or IP address. When this command is applied, if the target host is reachable from the source host, the source host receives a reply message from the target host, otherwise the command timeouts. An output of the **ping** command is shown in Figure 1.12.

```
C:\>ping 172.16.0.1

Pinging 172.16.0.1 with 32 bytes of data:
Reply from 172.16.0.1: bytes=32 time<1ms TTL=64
Reply from 172.16.0.1: bytes=32 time<1ms TTL=64
Reply from 172.16.0.1: bytes=32 time<1ms TTL=64
Reply from 172.16.0.1: bytes=32 time<1ms TTL=64

Ping statistics for 172.16.0.1:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

Figure 1.12: Output of ping command.

- **ipconfig :** The ipconfig command is used to determine your computer's networking information, such as the IP address, address of its default gateway, etc. An output of the **ipconfig** command is shown in Figure 1.13.

```
C:\>ipconfig

Windows IP Configuration


Ethernet adapter Ethernet:

   Connection-specific DNS Suffix  . : GU_IT
   Link-local IPv6 Address . . . . . : fe80::4125:c248:1aca:cf7%5
   IPv4 Address. . . . . . . . . . . : 172.16.1.186
   Subnet Mask . . . . . . . . . . . : 255.255.252.0
   Default Gateway . . . . . . . . . : 172.16.0.1
```

Figure 1.13: Output of ipconfig command.

- **hostname :** It is a simple command to display the hostname or computer name of a computer.

- **arp :** This command corresponds to the **Address Resolution Protocol (ARP)**. Although network communications are done with the help of IP addresses, data delivery is ultimately dependent on the MAC address of the computer, because MAC is the only address that is unique in a computer network. ARP maps IP addresses to MAC addresses. The operating system maintains a list of such mapping in memory. Using the **arp –a** command we can display the list of IP addresses and their corresponding MAC addresses. An output of the **arp** command is shown in Figure 1.14.

```
C:\>arp -a

Interface: 172.16.1.186 --- 0x5
  Internet Address      Physical Address      Type
  172.16.0.1            00-23-7d-f3-c1-65     dynamic
  172.16.0.160          8c-ec-4b-c6-81-30     dynamic
  172.16.1.31           00-4e-01-b0-ae-c0     dynamic
  172.16.1.84           e4-b9-7a-e7-64-4e     dynamic
  172.16.2.171          8c-ec-4b-c6-70-d3     dynamic
  172.16.3.139          8c-ec-4b-c9-94-47     dynamic
  172.16.3.179          8c-ec-4b-c6-6f-68     dynamic
  172.16.3.255          ff-ff-ff-ff-ff-ff     static
  224.0.0.22            01-00-5e-00-00-16     static
  224.0.0.251           01-00-5e-00-00-fb     static
  224.0.0.252           01-00-5e-00-00-fc     static
  239.255.255.250       01-00-5e-7f-ff-fa     static
  255.255.255.255       ff-ff-ff-ff-ff-ff     static
```

Figure 1.14: Output of arp -a command.

KEYWORDS LEARNED IN THIS CHAPTER

PAN	LAN	MAN	WAN	NIC
Access Point	Host	IP Address	MAC	Router
Protocol	Encryption	ARP	DHCP	SSID

Exercises

I. FILL IN THE BLANKS

1. The hostname command is used to display _____.
2. The length of a IPv4 and a IPv6 is _____ and _____ bytes respectively.
3. An access point is used to connect _____.
4. The ping command is used to _____.
5. HTTPS transfer _____ data.

II. MULTIPLE CHOICE QUESTIONS:

1. Which device is required to connect multiple heterogeneous networks?

A) Hub
B) Switch
C) Router
D) Access Point

2. Which is the largest type of computer network?

A) PAN
B) LAN
C) MAN
D) WAN

3. Which protocol transfers encrypted data instead of plain data?

A) HTTP
B) HTTPS
C) FTP
D) SMTP

4. How many bytes are reserved for OUI in a MAC address?

A) 3 B) 8 C) 24 D) 12

5. IP address can be automatically assigned if the network is connected with

A) Access Point
B) Mail Server
C) Web Server
D) DHCP Server

III. SHORT ANSWER QUESTIONS

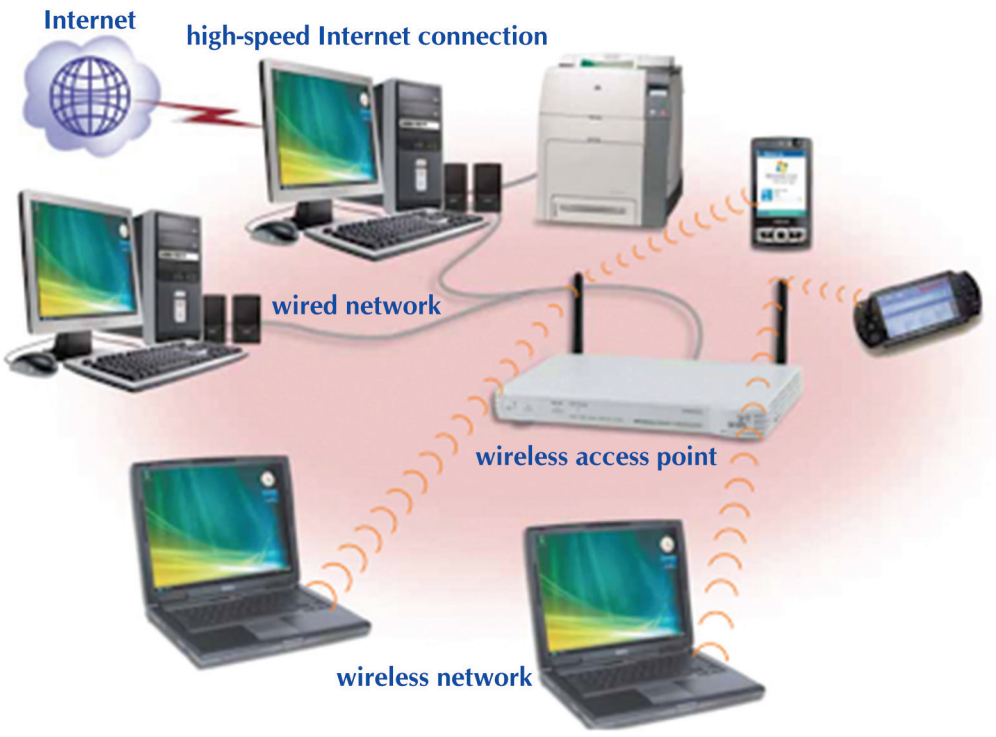
1. Mention the name of components required to set up a MAN network. Draw a block diagram of a MAN network labeling name of each component.
   (Hint: MAN network is a connection of multiple LAN)

2. Why hostname should not be used to identify a computer in a computer network?

3. Give five examples of valid and five examples of invalid IPv4 addresses.
   (Hint: Each segment of an IPv4 address has a fixed length)

4. Can we compare HTTP and FTP protocols? Mention their functionality.

5. How many 16 port switches are required to connect 31 computers in a network?
   (Hint: To connect all computers, all switches also should be connected among themselves)



CHAPTER 2

HTML and CSS3

PART - I Introduction

In this chapter

You will learn all the common HTML tags used to structure HTML pages, the skeleton of all websites. HTML lets you format text, add graphics, create links, input forms, frames and tables, etc., and save it all in a text file that any browser can read and display. You will also learn how to style an HTML document using CSS.

Web programming, also known as web development, is the creation of dynamic web applications. Examples of web applications are social networking sites like Facebook or e-commerce sites like Amazon.

There are two broad divisions of web development – front-end development (also called client-side development) and back-end development (also called server-side development).

Front-end development refers to constructing what a user sees when they load a web application – the content, design and how you interact with it. This is done with three codes – HTML, CSS and JavaScript.

HTML, short for Hyper Text Markup Language, is a special code for 'marking up' text in order to turn it into a web page. Every web page on the net is written in HTML, and it will form the backbone of any web application.

CSS, short for Cascading Style Sheets, is a code for setting style rules for the appearance of web pages. CSS handles the cosmetic side of the web.

JavaScript is a scripting language that's widely used to add functionality and interactivity to web pages.

2.1 WHAT IS HTML?

HTML stands for Hypertext Markup Language, and it is the most widely used language to write Web Pages.

Each word in HTML has its own significance as explained below:

- **Hyper** signifies that the user can view World Wide Web pages by moving anywhere in any direction.
- **Text** is anything written in English or alphanumeric character.
- **Markup** is what HTML tags do to the text inside of them; they mark it as a specific type of text.
- **Language** is a language that a computer system understands and uses to interpret commands.

HTML code includes two kinds of text:

Content: Content is the information to be displayed on the screen like text, pictures, audio, video etc.

Markup: Markup is the information inserted in the HTML script to control the display.

2.2 WHY HTML?

HTML is very useful for students and working professionals to become a great Software Engineer specially when they are working in Web Development.

Some of the key advantages of learning HTML are:

- **Create Web site** - You can create a website or customize an existing web template if you know HTML well.
- **Become a web designer** - If you want to start a career as a professional web designer, HTML and CSS designing is a must skill.
- **Understand web** - If you want to optimize your website, to boost its speed and performance, it is good to know HTML to yield best results.
- **Learn other languages** - Once you understand the basic of HTML then other related technologies like JavaScript, php etc are become easier to understand.

Thus, we can say, HTML is a set of logical codes or tags (markup) that are used to define the Web browser how to present the information in the Web page.

HTML is a platform independent language. It is not a programming language like C, C++ or Java. It is a set of markup tags that tells the browser how to display the Web page content.

Some popular markup languages are DHTML (Dynamic Hypertext Markup Language), SGML (Standard Generalized Markup Language), XHTML (Extended HTML), XML (Extensible Markup Language) etc.

2.3 HISTORY OF HTML

Tim Berners-Lee is known as the father of HTML. The first available description of HTML was a document called "HTML Tags" proposed by Tim in late 1991. The latest version of HTML is HTML5.

2.4 FEATURES OF HTML

1) It is a very easy and simple language. It can be easily understood and modified.

2) It is very easy to make an effective presentation with HTML because it has a lot of formatting tags.

3) It is a markup language, so it provides a flexible way to design web pages along with the text.

4) It facilitates programmers to add a link on the web pages (by html anchor tag), so it enhances the interest of browsing of the user.

5) It is platform-independent because it can be displayed on any platform like Windows, Linux, and Macintosh, etc.

6) It facilitates the programmer to add Graphics, Videos, and Sound to the web pages which makes it more attractive and interactive.

7) HTML is a case-insensitive language, which means we can use tags either in lower-case or upper-case.

2.5 HTML EDITOR

Two types of editor are used. These are WYSIWYG and Text editors.

- **WYSIWYG EDITOR**
  WYSIWYG stands for What You See Is What You Get. This editor allows the developer to see what the end result will look after the document is created.
  Examples of WYSIWYG editors are Adobe Dreamweaver, Amaya and Google Web Designer.

- **TEXT EDITOR**
  We can create HTML documents using text editors like Notepad or WordPad. The user should have the proper knowledge of HTML commands to develop a web page.

2.6 BUILDING BLOCKS OF HTML

An HTML document consist of its basic building blocks which are:

- Tags: An HTML tag surrounds the content and apply meaning to it. It is written between < and > brackets.
- Attribute: An attribute in HTML provides extra information about the element, and it is applied within the start tag. An HTML attribute contains two fields: name & value.
- Elements: An HTML element is an individual component of an HTML file. In an HTML file, everything written within tags are termed as HTML elements.

2.7 HTML TAGS

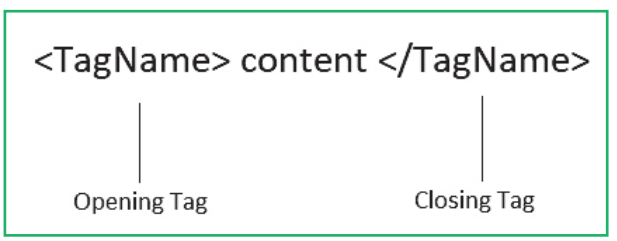
HTML tags are like keywords which define that how web browser will format and display the content. With the help of tags, a web browser can distinguish between an HTML content and a simple content. HTML tags contain three main parts: opening tag, content and closing tag. But some HTML tags are unclosed tags. The text of opening and closing tags is the same but the closing tags contain forward slash (/) character.

When a web browser reads an HTML document, browser reads it from top to bottom and left to right. HTML tags are used to create HTML documents and render their properties. Each HTML tags have different properties.

An HTML file must have some essential tags so that web browser can differentiate between a simple text and HTML text. You can use as many tags you want as per your code requirement.

- All HTML tags must enclosed within < > these brackets.
- Every tag in HTML performs different tasks.
- If you have used an open tag <tag>, then you must use a close tag </tag> (except some tags)

2.7.1 Syntax



The HTML tags can be categorized as:

1. **Container tags:** These tags come in pairs, i.e., they have both opening and closing tags. A tag is said to be paired tag if it along with a closing tag appears at the end.
   Example: <HTML> .... </HTML>, <BODY>....</BODY> etc.

2. **Empty tags:** It is also called singular tag. These type of tags doesn't have closing tag.
   Example:
, <HR> etc.

2.7.2 Nesting of Tags

HTML elements can be nested, meaning that one element can be placed inside another element. Nesting allows you to apply multiple HTML tags to a single piece of content.

Whenever we nest an HTML tag inside of another tag, we indent the inner tag so that the overall tag hierarchy is clear. Take a look at the following example of a well-structured HTML document.

```
<HTML>
        <HEAD>
                <TITLE> Example of indent of tags </TITLE>
        </HEAD>
        <BODY>
                This is the body
        </BODY>
</HTML
```

Keeping your HTML well indented so that every tag and "level" of nesting is aligned will make your code easier to read and maintain.

2.8 HTML ATTRIBUTE

- HTML attributes are special words which provide additional information about the elements or attributes are the modifier of the HTML element.
- Each element or tag can have attributes, which defines the behaviour of that element.
- Attributes should always be applied with start tag.
- The Attribute should always be applied with its name and value pair.
- The Attributes name and values are case sensitive, and it is recommended by W3C that it should be written in Lowercase only.
- You can add multiple attributes in one HTML element, but need to give space between two attributes.

2.8.1 Syntax

<element attribute_name="value">content</element>

2.9 HTML ELEMENTS

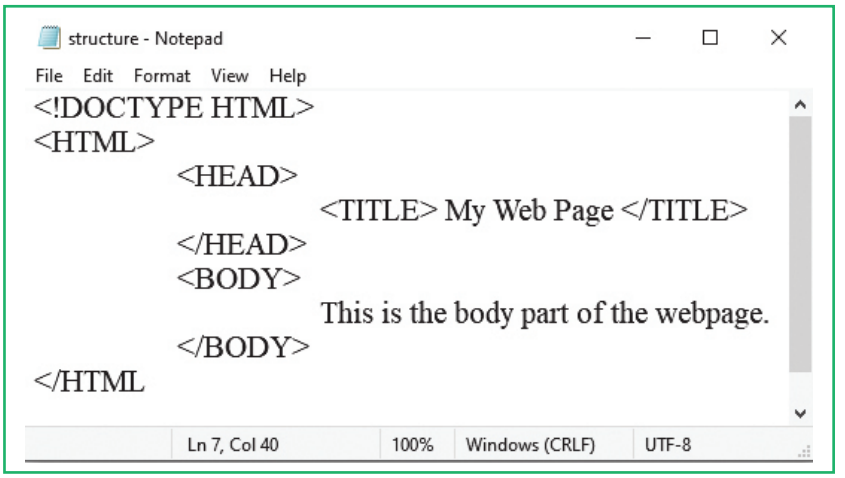
An HTML file is made of elements. These elements are responsible for creating web pages and define content in that webpage. An element in HTML usually consist of a start tag <tag name>, close tag </tag name> and content inserted between them. Thus we can say an element is a collection of start tag, attributes, end tag, content between them.

2.10 BASIC STRUCTURE OF HTML DOCUMENT

The HTML document is mainly divided into two sections. They are HEAD section and BODY section.

- **HEAD:** This contains the information about the HTML document. For Example, Title of the page, version of HTML etc.
- **BODY:** This contains everything you want to display on the Web Page.

Let us now have a look at the basic structure of HTML. That is the code that is a must for every webpage to have:



Explanation:

- The DOCTYPE declaration at the beginning of our page specifies that the document is written in HTML5. In HTML5, the DOCTYPE declaration is required.
- <HTML>: This is called HTML root element and indicates the document has been written in HTML.

- </HTML>: It indicates the end of the HTML document.
- <TITLE>: This tag is used to give the title of the web page. The title is displayed in the title bar of the browser window.
- </TITLE>: It is used to end the title.
  <HEAD>: Head tag contains metadata, title, page CSS etc.
- </HEAD>: It is used to end the heading.
- <BODY>: This tag contains the body of the web page.
- </BODY>: It is used to end the body.

2.11 RULES FOR WRITING HTML CODES

The following rules must be followed while writing HTML codes:

- ✓ Container tags should always be closed properly.
- ✓ Tag name should not contain spaces
- ✓ There should not be any space between < and > in a tag.
- ✓ Tags must be nested correctly.

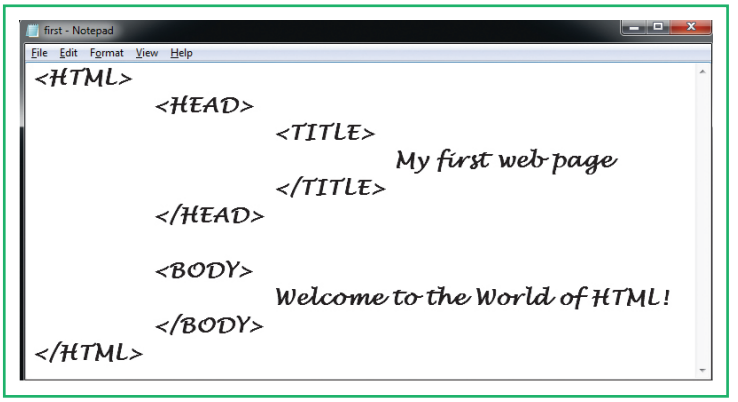
Steps to create a web page

1. Write the HTML code in any HTML editors.
2. Save the page with file extension of .html or .htm
3. Open the page in any Web Browser (like Internet Explorer, Google Chrome, Mozilla Firefox, Opera etc.

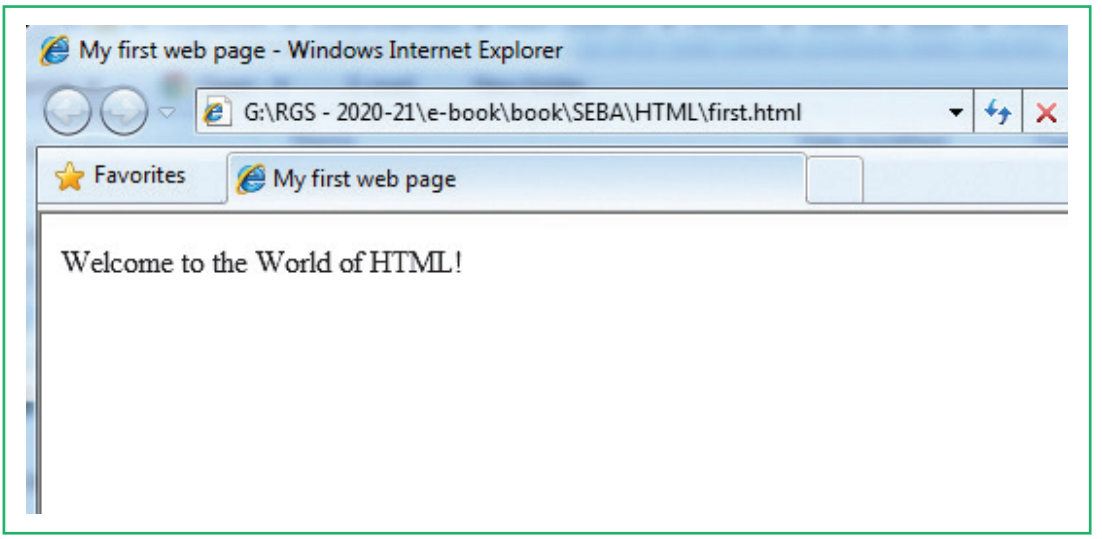
NOTE: HTML is not case sensitive like HTML and html both are same.

2.12 FIRST WEB PAGE

Let us create our first web page which will print "Welcome to the World of HTML!" on the screen.
Open your text editor, and type the below code in it and save it with the name "first.html".



On opening the file in a web browser, you will see the below output.



As HTML is a scripting language, you can simply change your code and hit the refresh button and the changes will be reflected to your Web page immediately.

2.13 BODY TAGS

Body tag is used to enclose all the data which a web page has from texts to links. All the content that you see rendered in the browser is contained within this element.

2.13.1 Attributes of BODY tag

Attribute	Function	Syntax
BGCOLOR	Specify the background color of the web page.	<BODY BGCOLOR=COLOUR NAME> Example: <BODY BGCOLOR=CYAN>
TEXT	Specify the color of the text written on the web page.	<BODY TEXT=COLOUR NAME> Example: <BODY TEXT=BLUE>
BACKGROUND	Specifies an image as the background of the web page.	<BODY BACKGROUND=IMAGE FILE NAME WITH PATH> Example: <BODY BACKGROUND=”D:\flower.jpg”>
TOPMARGIN	Specify the top margin of the web page.	<BODY TOPMARGIN-Value (in pixel)> Example: <BODY TOPMARGIN=20>
LEFTMARGIN	Specify the left margin of the web page.	<BODY LEFTMARGIN-Value (in pixel)> Example: <BODY TOPMARGIN=10>

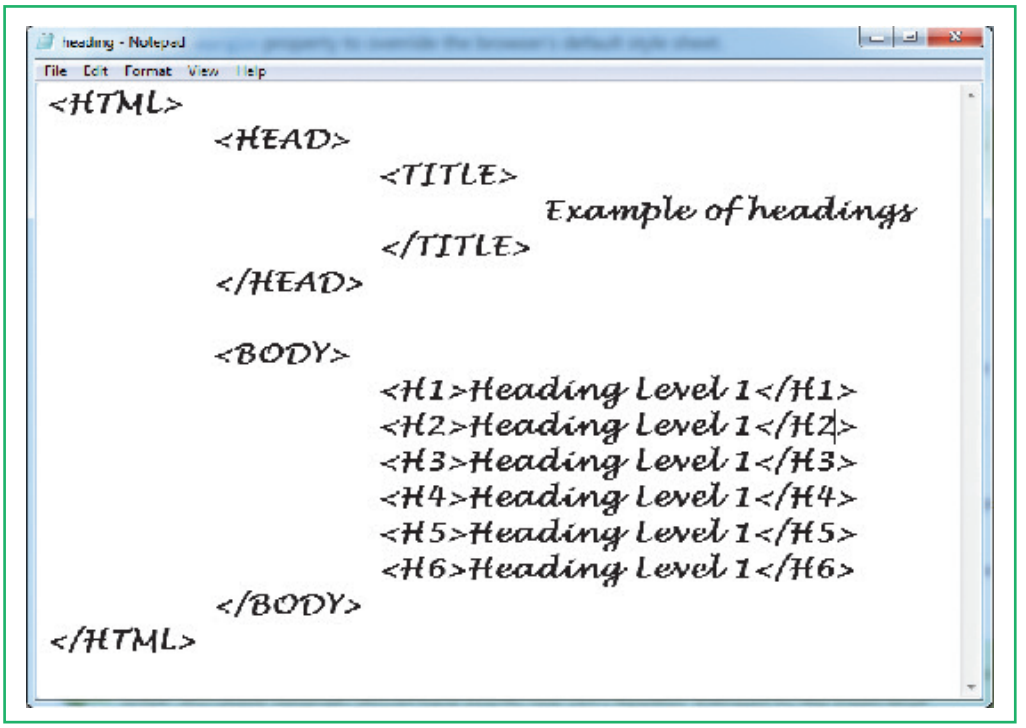
2.14 HEADING TAGS

Headings help in defining the hierarchy and the structure of the web page content.

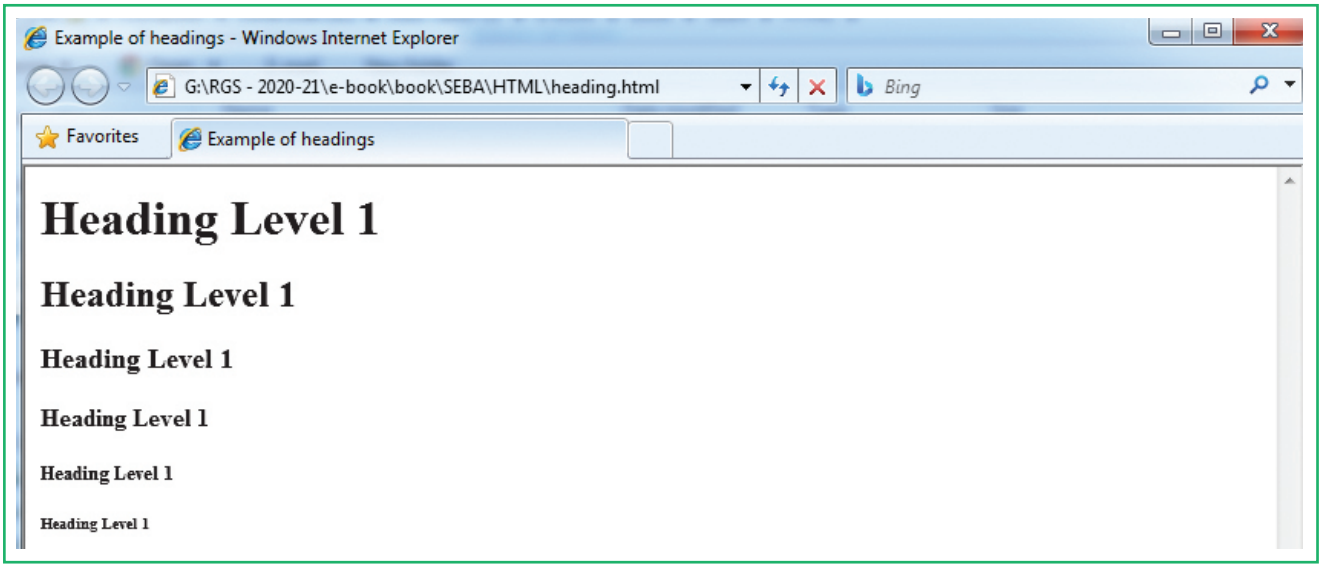
HTML offers six levels of heading tags, <h1> through <h6>; the lower the heading level number, the greater its importance.

By default, browsers display headings in larger and bolder font than normal text. Also, <h1> headings are displayed in largest font, whereas <h6> headings are displayed in smallest font.

Example:



Output



2.14.1 Importance of Headings

- HTML headings provide valuable information by highlighting important topics and the structure of the document, so optimize them carefully to improve user engagement.
- Don't use headings to make your text look BIG or bold. Use them only for highlighting the heading of your document and to show the document structure.
- Since search engines, such as Google, use headings to index the structure and content of the web pages so use them very wisely in your webpage.
- Use the <h1> headings as main headings of your web page, followed by the <h2> headings, then the less important <h3> headings, and so on.

2.14.2 Attributes of Heading

Attribute	Function	Syntax
ALIGN	Controls the alignment of the text on the web page with respect to the margins. The different types of alignments are: **Left:** To align the text to the left. **Right:** To align the text to the right. **Center:** To align the text to the center.	<H1 ALIGN = LEFT/RIGHT/CENTER> Example: <H1 ALIGN=CENTER>

2.15 PARAGRAPH TAG

The paragraph tags are used to define a block of text as a paragraph. In HTML, <P> tag represent a paragraph. The paragraph tag is written as <P> and </P>. An HTML <p> tag indicates starting of new paragraph.

A blank line is inserted before and after the text enclosed between these two tags.

2.15.1 Attributes of Paragraph

Attribute	Function	Syntax
ALIGN	Controls the alignment of the text on the web page with respect to the margins. The different types of alignments are: Left: To align the text to the left. Right: To align the text to the right. Center: To align the text to the center. Justify: Stretches the lines so that each line has an equal width.	<P ALIGN = LEFT/RIGHT/ CENTER/JUSTIFY> Example: <P ALIGN=CENTER>

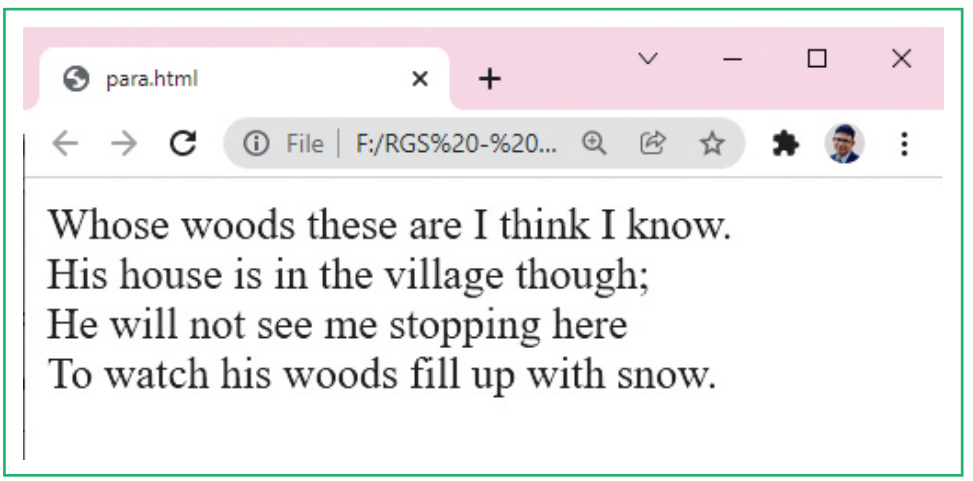
2.16 BREAK TAG

In HTML, to add line breaks, the
 tag is used. It is generally used in poem or address where the division of line is necessary. It is an empty tag which means that it has no end tag.

To add line breaks, write
 where you want the line break to occur. You can insert a line break to shift the text to go to a new line.

Coding

Output



2.17 HORIZONTAL RULE TAG

Horizontal Rule or <HR> tag are horizontal lines spread across the width of the web page or the browser window. It is an empty tag which means it has no closing tag.

<HR> tag is used to separate different parts of the text from one another.

2.17.1 Attributes of Horizontal Rule (<HR> Tag)

Attribute	Function	Syntax
SIZE	Specify the thickness of the line.	<HR SIZE = VALUE IN PIXEL> Example: <HR SIZE = 4>
WIDTH	Specify the width of the line in percentage.	<HR WIDTH = VALUE IN PERCENTAGE> Example: <HR WIDTH = 50%>
COLOR	Specify the color of the line.	COLOR <HR COLOR=COLOR NAME> Example: <HR COLOR=RED>
ALIGN	Controls the alignment of the line on the web page. The different types of alignments are: **Left:** To align the text to the left. **Right:** To align the text to the right. **Center:** To align the text to the center.	<HR ALIGN = LEFT/RIGHT/ CENTER> Example: <HR ALIGN=RIGHT>
NOSHADE	Specifies that a <HR> element should render in one solid color and noshaded.	Example: <HR SIZE=20 NOSHADE>

2.18 COMMENT TAG

Comments are textual content which appear in your HTML code, but are not rendered by user's browser. Comments are given between <! - - and - - >. Browsers ignore the text between comment character sequences.

Example:

```
<! - -    This is a comment - ->
<! - -    This is a
   Multiple line comment - - >
```

NOTE: Most of the attributes of different tags are obsolete in HTML 5; you need to use CSS instead.

2.19 HTML FORMATTING

HTML Formatting is a method of text formatting for a better appearance and look. HTML includes a lot of formatting tags. These tags are used to make bold, underline, and italic text.

For HTML the tags are divided into two categories:

- **Physical tag:** These tags are used to give the text a visual appearance.
- **Logical tag:** Logical or semantic tags are used to append the value. Logical styles render the text according to its meaning.

Physical Tags

Attribute	Function	Syntax
<B> ... </B>	It is used to bold the text between the tags.	<B> Text to be bold </B> Example: <B> Computers </B>
<I> .... </I>	It is used to make italic text.	<I> Text to be italic </I> Example: <I> Computers </I>
<U> ... </U>	This tag is used to underline the text that has been written between them.	<U> Text to be underline </U> Example: <U> Computers </U>

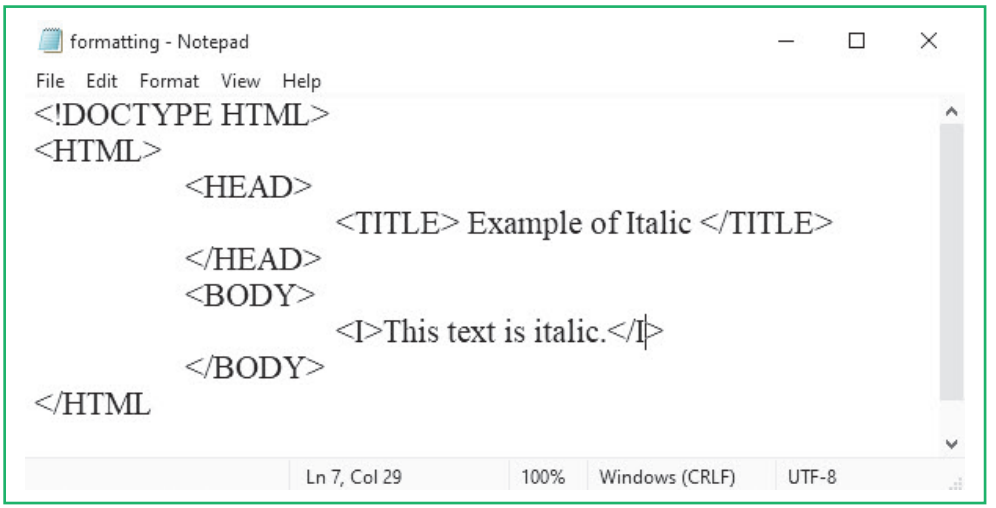
Logical Tags

Attribute	Function	Syntax
<STRONG> ... </STRONG	It is a logical tag that tells the browser that text is important.	<STRONG> Text to be strong</STRONG> Example: <STRONG> Computers </STRONG>
<EM> ... </EM>	It is used to display the content in italics.	<EM> Text to be italic </EM> Example: <EM> Computers </EM>

2.19.1 Bold Text

The HTML <B> element is a physical tag that displays text in bold font without any logical importance. If you write anything within the <B>..... </B> element, it is shown in bold letters.

Code

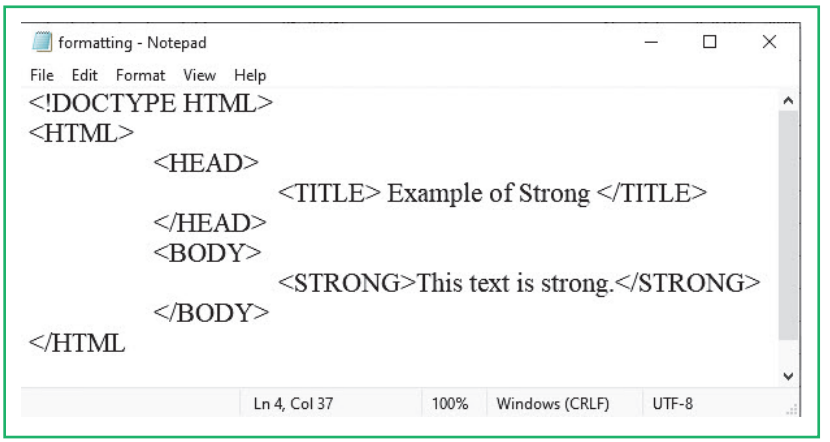


Output

2.19.2 Strong Text

The HTML < strong > tag is a logical tag that displays content in bold font and informs the browser of its logical significance.

Code

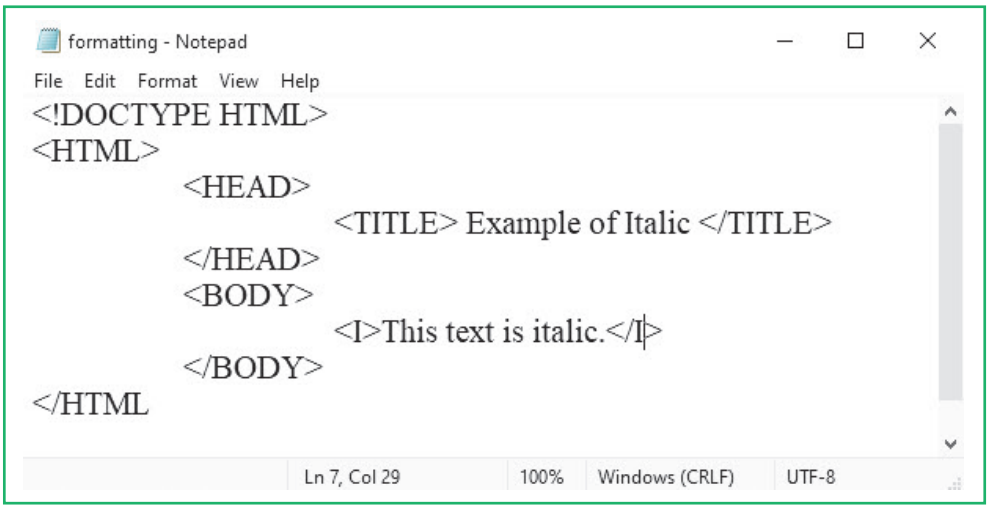


Output

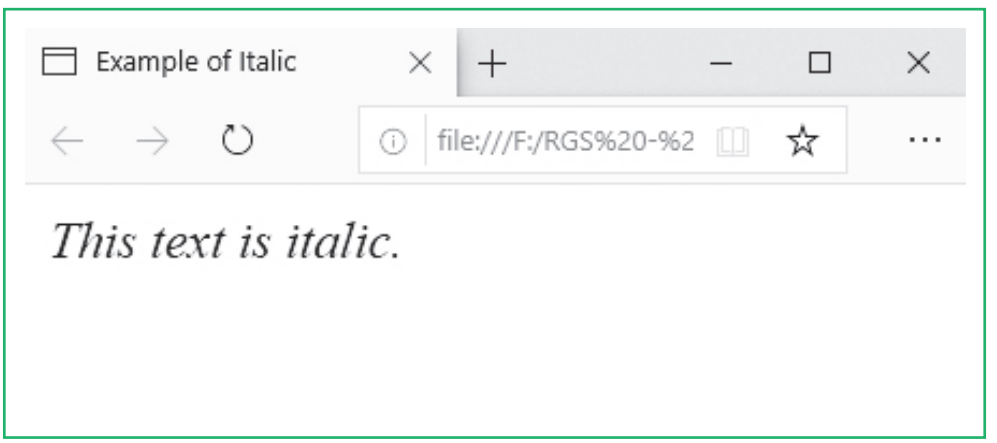
2.19.3 Italic Text

The HTML <I> element is physical element, which display the enclosed content in italic font, without any added importance. If you write anything within <I>.....</I> element, is shown in italic letters.

Code



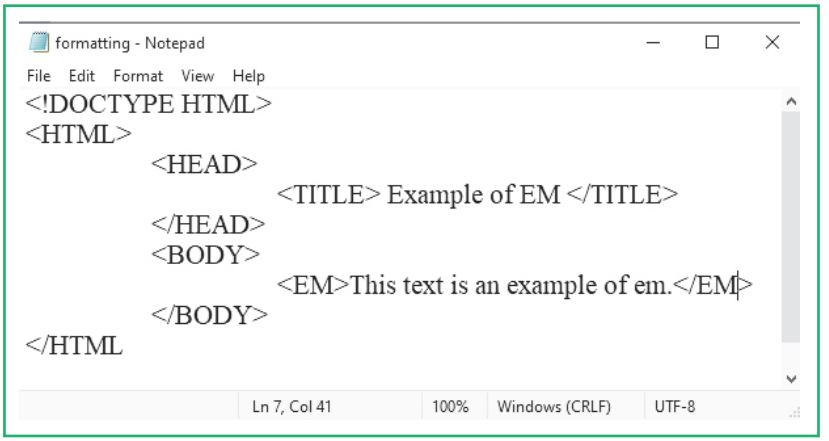
Output



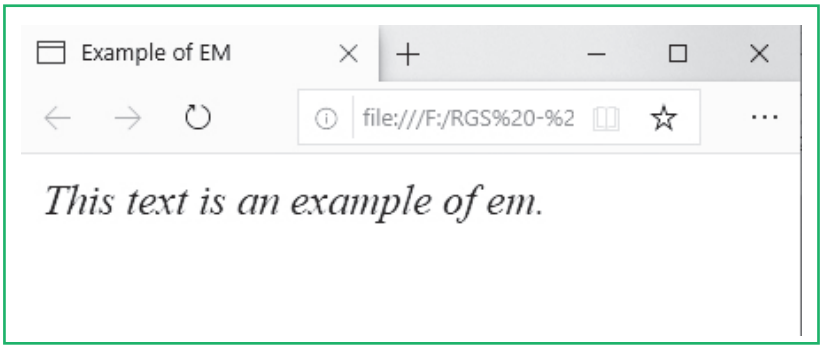
2.19.4 Emphasized Tag

The HTML <EM> tag is a logical element, which will display the enclosed content in italic font, with added semantics importance.

Code



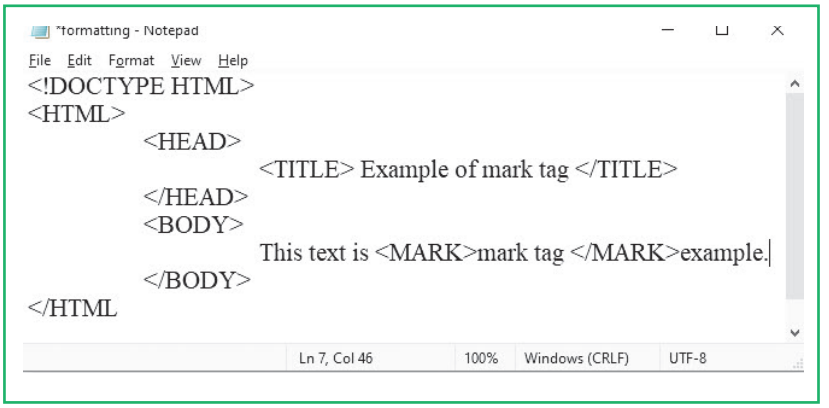
Output



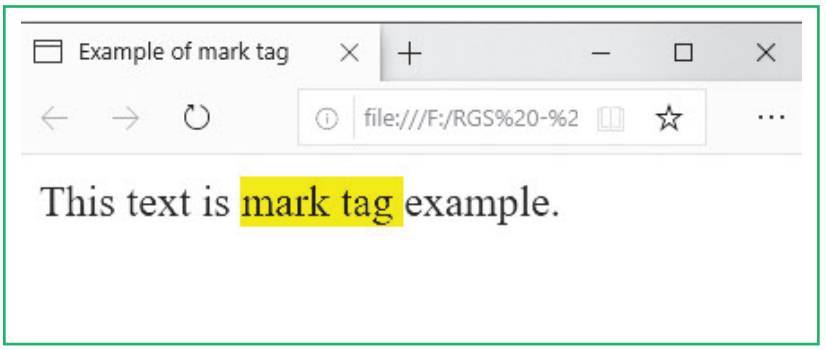
2.19.5 Marked Text

If you want to mark or highlight a text, you should write the content within <MARK>.....</MARK>.

Code



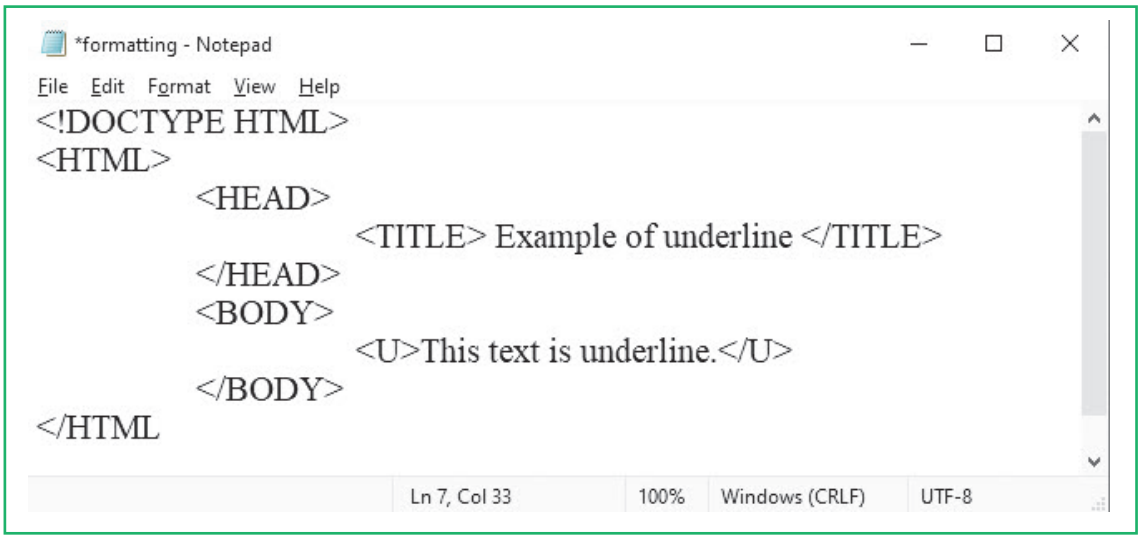
Output



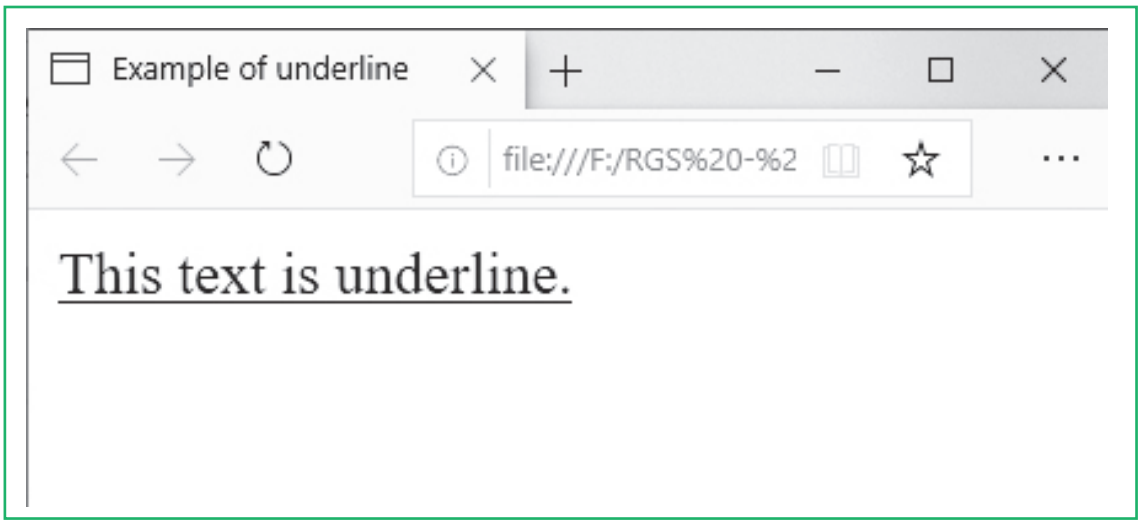
2.19.6 Underlined Text

If you write anything within <U>.....</U> element, is shown in underlined text.

Code



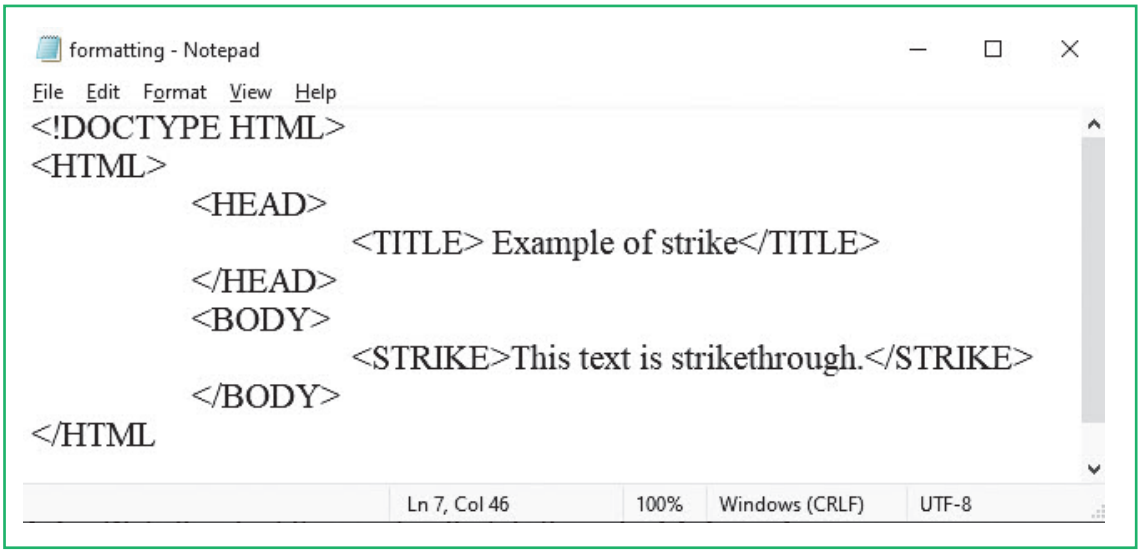
Output



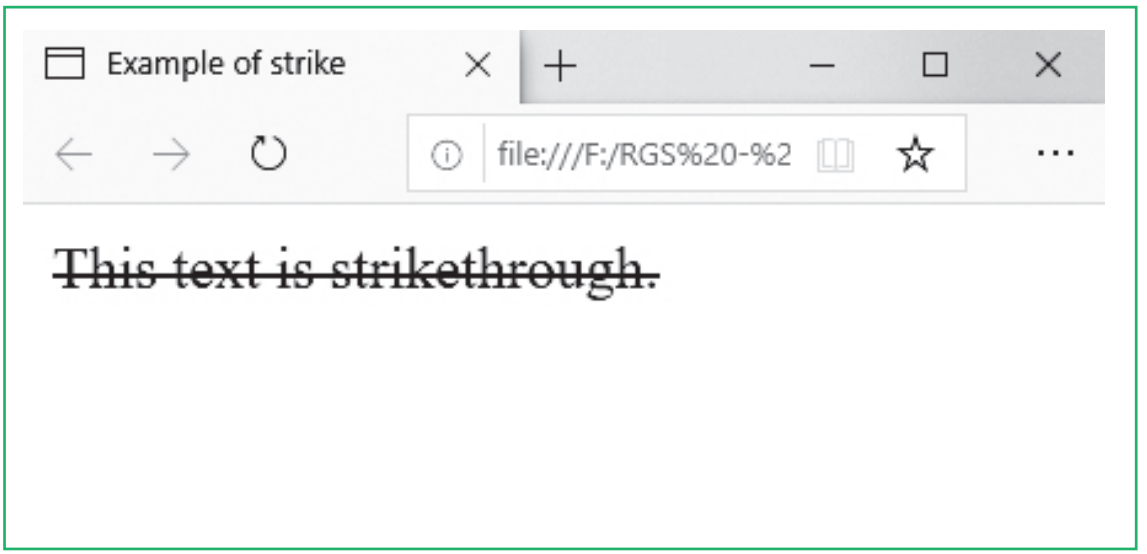
2.19.7 Strike Text

Anything written within <STRIKE>.....</STRIKE> element is displayed with strikethrough. It is a thin line which cross the statement.

Code



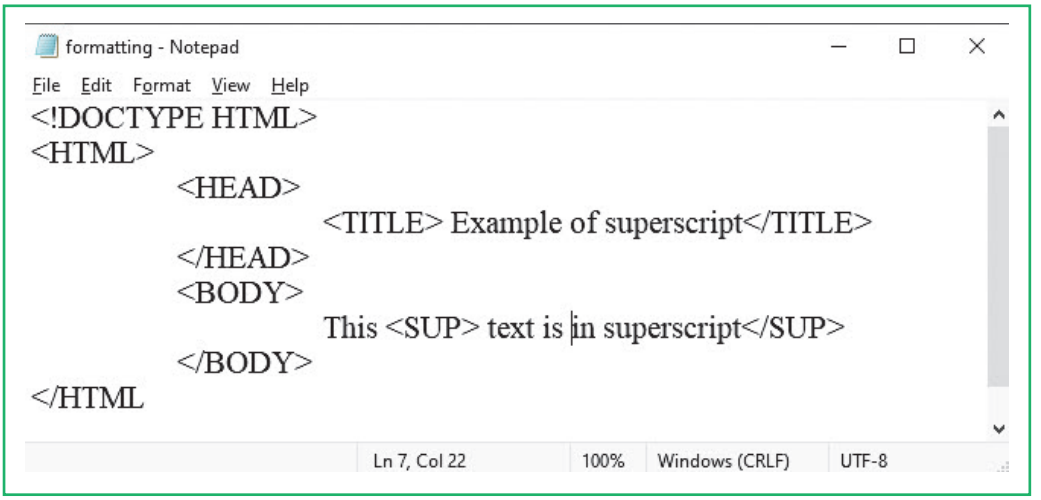
Output



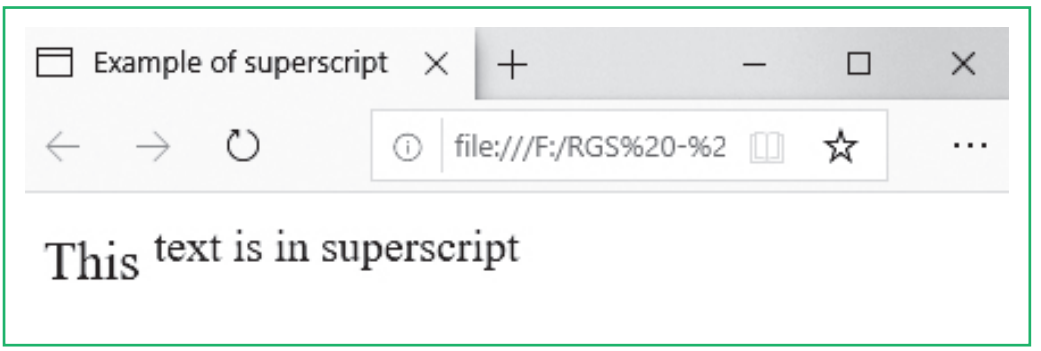
2.19.8 Superscript Text

If you put the content within ^{.....} element, is shown in superscript; means it is displayed half a character's height above the other characters.

Code



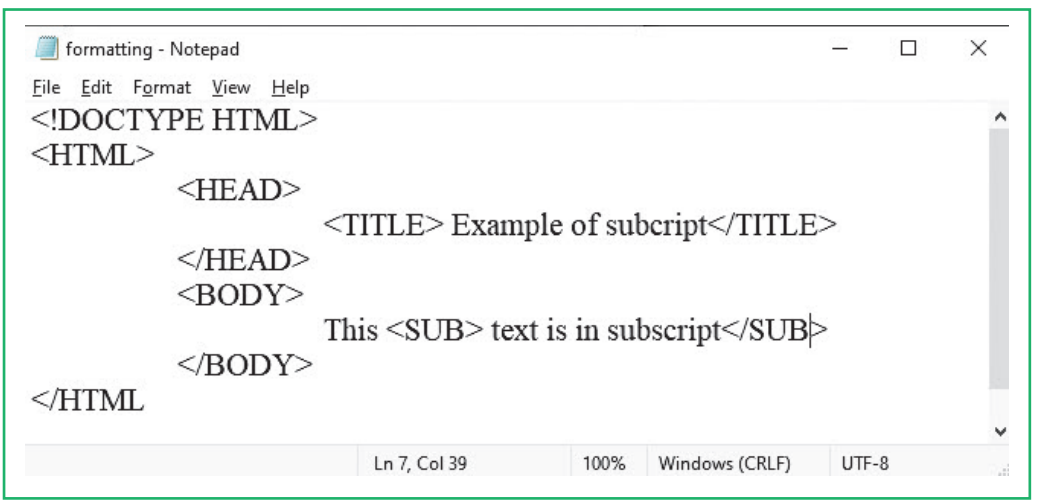
Output



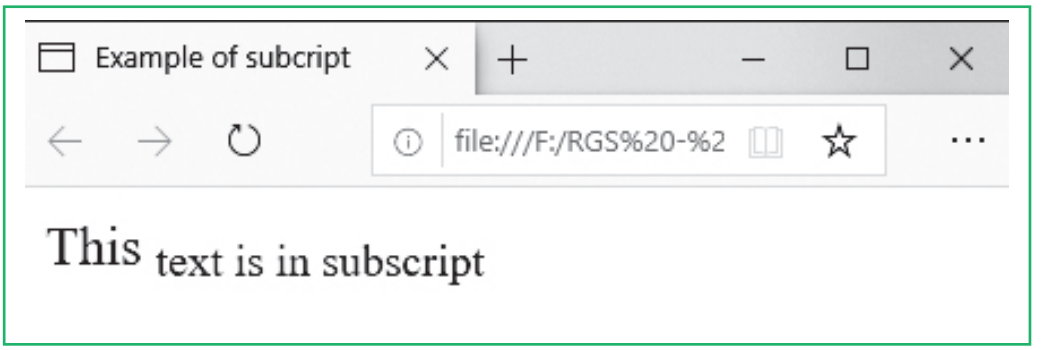
2.19.9 Subscript Text

If you put the content within _{.....} element, is shown in subscript ; means it is displayed half a character's height below the other characters.

Code



Output



2.20 CASCADING STYLE SHEET

Cascading Style Sheets, fondly referred to as CSS, is a simple design language intended to simplify the process of making web pages presentable.

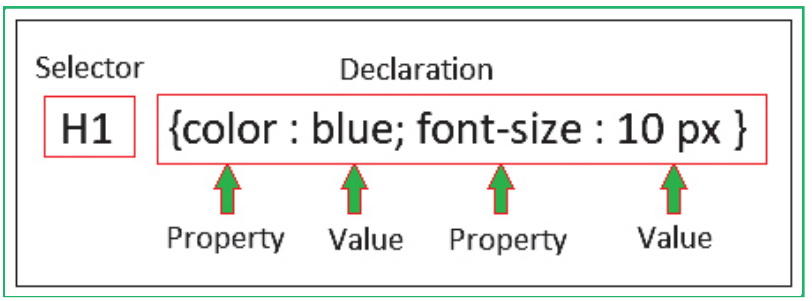
CSS is used to control the style of a web document in a simple and easy way. It provides an additional feature to HTML. It is generally used with HTML to change the style of web pages and user interfaces.

2.20.1 Advantages of CSS

- CSS plays an important role, by using CSS you simply got to specify a repeated style for element once & use it multiple times as because CSS will automatically apply the required styles.
- The main advantage of CSS is that style is applied consistently across variety of sites. One instruction can control several areas which is advantageous.
- If you are using CSS, you do not need to write HTML tag attributes every time. Just write one CSS rule of a tag and apply it to all the occurrences of that tag. So less code means faster download times.
- Cascading sheet not only simplifies website development, but also simplifies the maintenance as a change of one line of code affects the whole web site and maintenance time.
- It is less complex therefore the efforts are significantly reduced.

2.20.2 Syntax of CSS

The Cascading Style Sheets has two parts: selector and declaration.



Selector: Selector indicates the HTML element you want to style. It could be any tag like <h1>, <title> etc.

Declaration Block: The declaration specifies the formatting to the selector. For the above example, there are two declarations:

1. color: blue;

2. font-size: 10 px;

Each declaration contains a **property** name and **value**, separated by a semi-colon.

Property: A Property is a type of attribute of HTML element. It could be color, border etc.

Value: Values are assigned to CSS properties. In the above example, value "blue" is assigned to color property.

Some of the properties and their values used with the style attribute are given below:

Property	Value	Description
Background-color	Name of the color	Specifies background color
Color	Name of the color	Specifies the text color
Text-align	Left, Right, Center, Justified	Specifies the alignment of the text.
Font-family	Name of the font style	Specifies the font style of the text.
Font-size	The size of the font to be given in pixel. (The default font size of the font is 16 px)	Specify the font size of the text

2.20.3 Using CSS

CSS can be added to HTML documents in 3 ways:

Inline CSS

An inline CSS is used to apply a unique style to a single HTML element.

An inline CSS uses the style attribute of an HTML element.

Syntax to use the style attribute with a tag:

<TAGNAME STYLE = “PROPERTY : VALUE;”>

Example:

<BODY STYLE = “BACKGROUND-COLOR : YELLOW; COLOR:BLUE”>

Internal CSS

An internal CSS is used to define a style for a single HTML page.
An internal CSS is defined in the <head> section of an HTML page, within a <style> element.

Example:

int - Notepad

File Edit Format View Help

```
<HTML>
        <HEAD>
        <STYLE>
                BODY {BACKGROUND-COLOR : YELLOW;}
                H1 {COLOR : RED;}
                P {COLOR : BLUE}
        </STYLE>
        </HEAD>
<BODY>
        <H1> THIS IS A HEADING </H1>
        <P> THIS IS PARAGRAPH </P>
</BODY>
</HTML>
```

This code will display the web page background color as yellow, heading as red and paragraph as blue in color.

External CSS

An external style sheet is used to define the style for many HTML pages.
To use an external style sheet, add a link to it in the <head> section of each HTML page:

Example:

File Edit Format View Help

```
<!DOCTYPE html>
<html>
<head>
  <link rel="stylesheet" href="styles.css">
</head>
<body>

<h1>This is a heading</h1>
<p>This is a paragraph.</p>

</body>
</html>
```

Ln 12, Col 8 | 100% | Windows (CRLF) | UTF-8

The external style sheet can be written in any text editor. The file must not contain any HTML code, and must be saved with a .css extension.

KEYWORDS LEARNED IN THIS CHAPTER

HTML	MARKUP	EDITOR	TAGS
ATTRIBUTE	PHYSICAL TAGS	LOGICAL TAGS	CSS

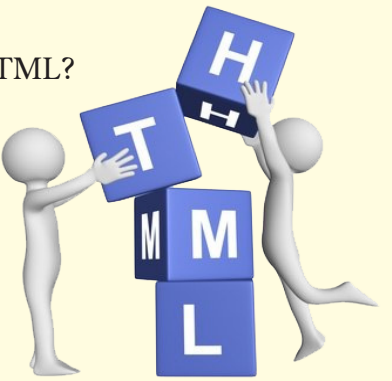
Exercise

I. FILL IN THE BLANKS:

1. _____ documents are made up of text content and special codes.
2. _____ are used to write notes about an HTML document.
3. HTML document is saved with an extension _____
4. _____ are used to view the HTML documents.
5. The _____ element include both on and off tags.
6. The _____ element splits the line and displays the text on a new line.

II. MULTIPLE CHOICE QUESTIONS:

1. Which is the correct way to comment out something in HTML?
   i. Using ## and #
   ii. Using <!-- and -->
   iii. Using </-- and -/>
   iv. Using <!-- and -!>



2. The _____ tag draws a horizontal line across the web page.
   i.
 ii. <hr> iii. <line>
3. _____ provides a set of style rules for defining the layout of the HTML documents.
   i. CSS ii. WSS iii. TSS
4. A property and its value are collectively known as _____
   i. Selector ii. Attribute iii. Declaration

5. Which tag indicates the beginning and end of the HTML documents?

i. <HEAD> ii. <BODY> iii. <HTML>

6. Which of the following is used to define the style for a single HTML page?

i. Inline CSS ii. Internal CSS iii. External CSS

III. APPLICATION BASED QUESTIONS:

1. Yashvi was styling an HTML document using CSS. She wants to add styles directly to an HTML tag using the style attribute with the tag. How can she do this?
2. Rohan wants to divide his web page into different sections so that the content is easily readable. Which tag can he use for this? Which attributes can he use to define some extra properties of this tag?
3. Kirti wants to set the image of a park as the background of her web page but she is unable to do it. Which tag should she use to do so? Give the syntax.

IV. ANSWER THE FOLLOWING:

1. What is a markup language?
2. Write some feature of HTML.
3. Explain the terms tags and attributes with the help of an example.
4. How are comments useful?
5. What are Cascading Style Sheets? Name the different methods available for applying Style rules in an HTML document.
6. Differentiate between Internal CSS and External CSS.

LAB ACTIVITY

1. Create a web page that serves as an invitation card to your birthday party. Use all the HTML tags along with CSS properties you have learnt to make it attractive and lively.
2. Create a web page that serves as a guide for the mathematical formulas using HTML tags along with CSS properties.

PART - II List, Tables And Images

2.21 LIST IN HTML

Whether it is the minutes of a meeting, a list of items, or a table of contents of a document, you will find the use of a list in all.

HTML contain various tags to display items in an organized layout on a web page. You can organize the content on a web page using list. List are used to group related contents together in a structured manner making content easy to read and understand.

HTML supports different elements to create a list for displaying items in a specific order. It can be defined in different styles. The types of list HTML provides are:

- **Ordered list** - To group a set of related items in a specific numbered order.
- **Unordered list** - To group a set of related items in no specific order.
- **Description list** - To group a set of related terms and their definitions.

2.21.1 Ordered List

An ordered list is used to display the list of items in a specific order. An ordered list indents and gives a number to each item in the list.

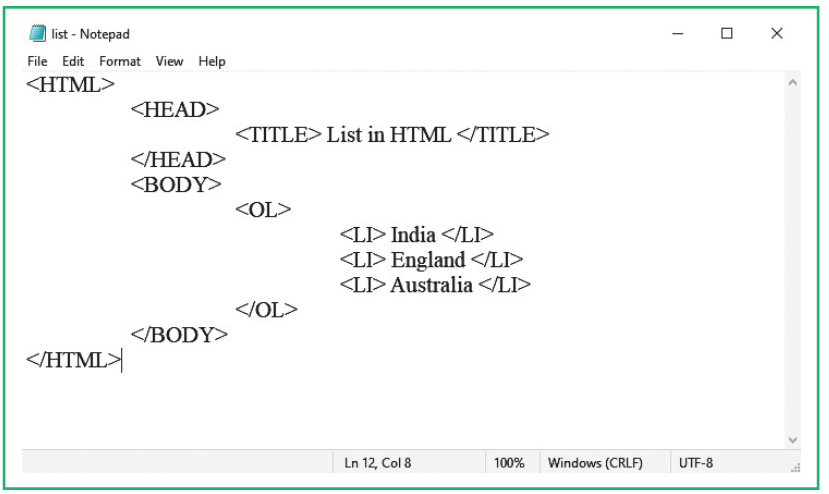
An ordered list is enclosed within the <OL> and </OL> tag. Each item in the list is given an <LI> tag that specifies lite item. The <LI> tag is used to represent individual list items within the <OL> tag.

Format:

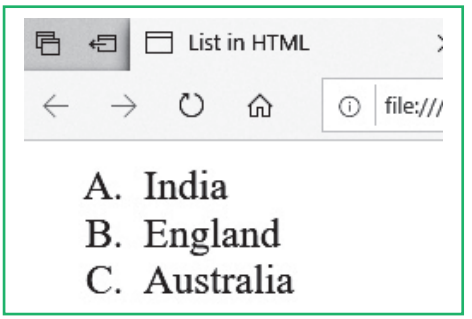
```
<OL>
    <LI> Item 1 </LI>
    <LI> Item 2 </LI>
    <LI> Item 2 </LI>
</OL>
```

Example :

Code:



Output:

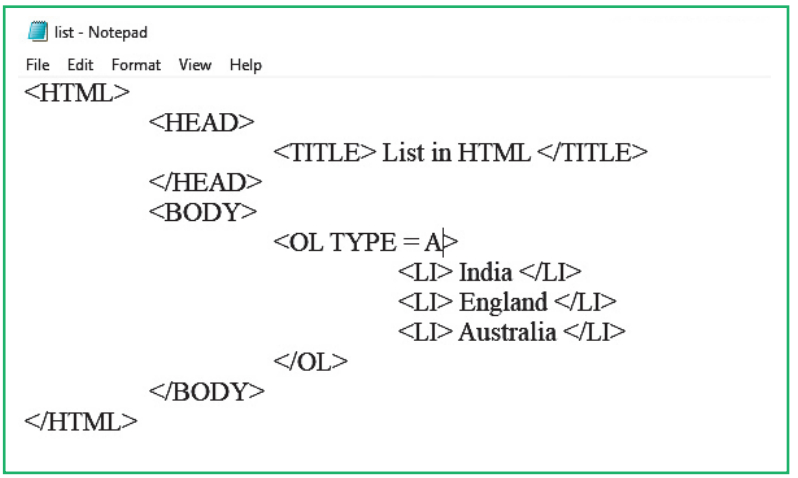


By default, ordered list items are marked with the numbers (1, 2, 3, ...), in ascending order. You can change the number style or the starting of the list using the attribute of the <OL> tag.

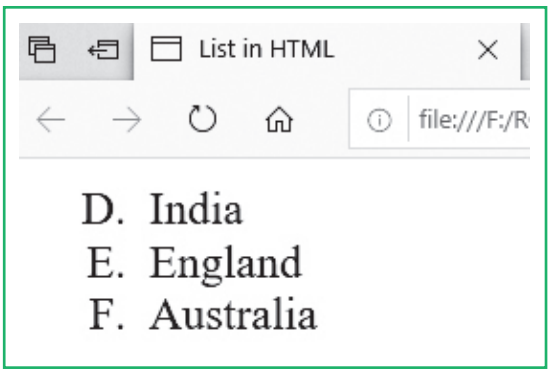
Attribute	Function	Syntax
TYPE	Specifies what kind of numbered list is to be displayed, such as A : Upper case letters a : Lower case letters 1 : Arabic numerals I : Roman numerals (Upper case) i : Roman numerals (Lower case)	<OL TYPE = I / i / 1 / A / a> Example: <OL TYPE = A>
START	Specifies the starting value of the numbered list	<OL START = VALUE> Example: <OL START = 3>
REVERSED	Is used to set reversed ordering of list items in an ordered list.	<OL START = VALUE REVERSED> Example: <OL START = 3 REVERSED>

Example of <OL> tag using TYPE attribute

Code:

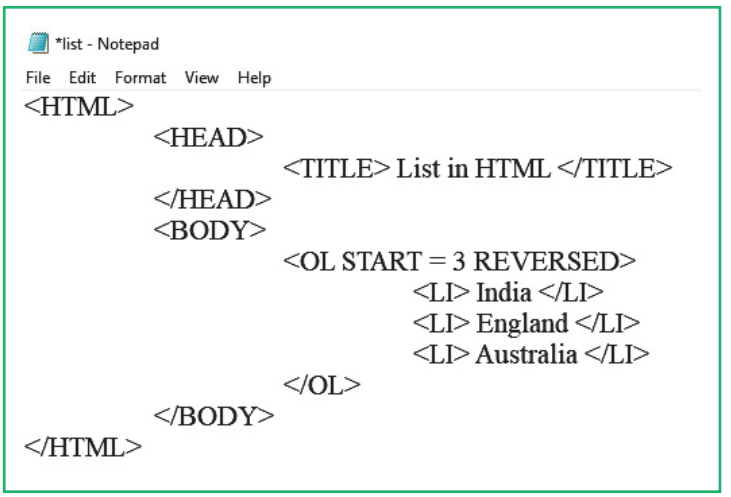


Output:

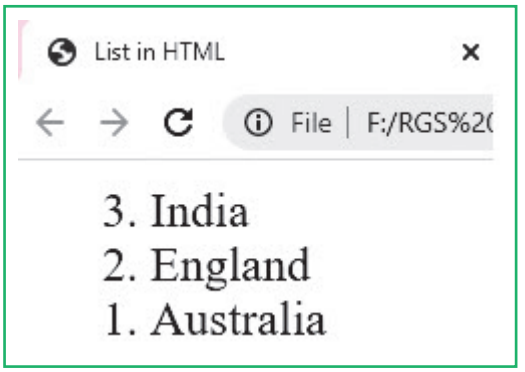


Example of <OL> tag using START attribute

Code:

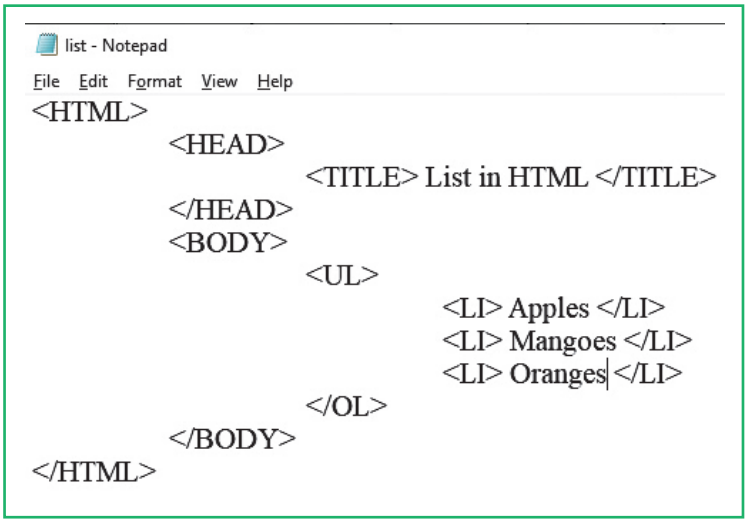


Output:

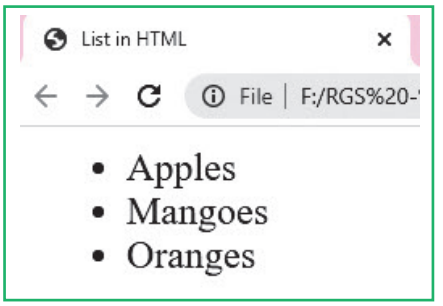


Example of <OL> tag using REVERSED attribute

Code:



Output:



2.21.2 Unordered List

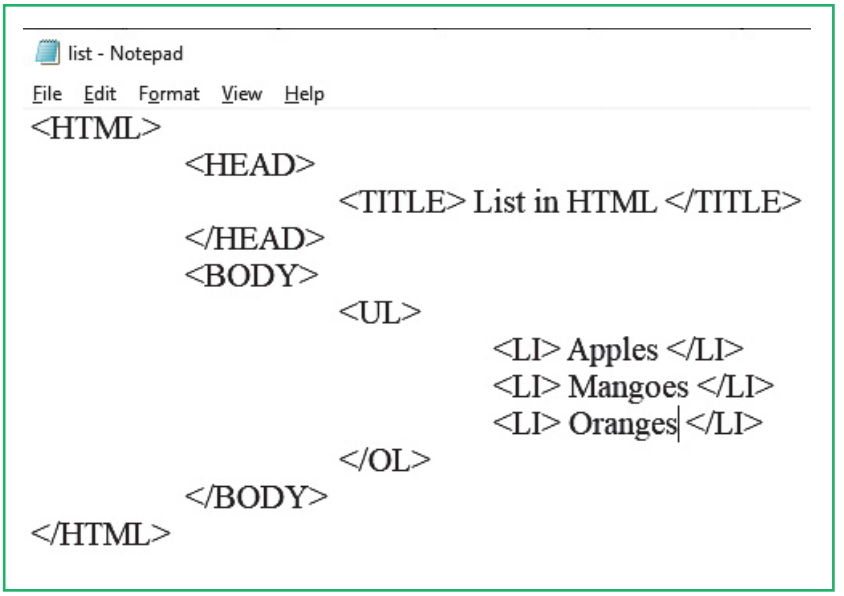
An unordered list is used when the items in the list are not required to be in a specific order. It is also called bulleted list. An unordered list is enclosed within the <UL> ..... </UL> tag. Just like in ordered list, <LI> tag is used to mark the individual list items.

Format:

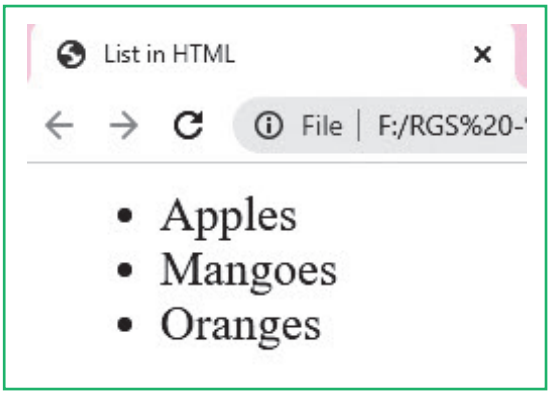
```
<UL>
    <LI> Item 1 </LI>
    <LI> Item 2 </LI>
    <LI> Item 3 </LI>
</<UL>
```

Example:

Code:



Output:



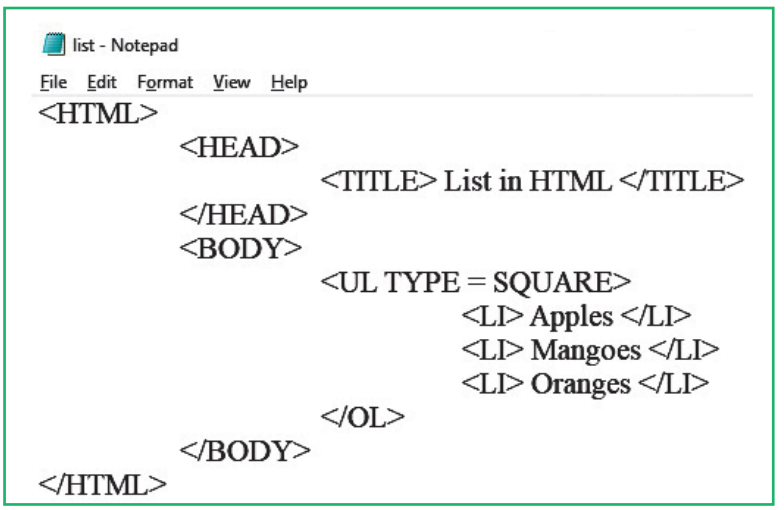
Attribute	Function	Syntax
TYPE	Specifies what kind of bullet used to mark items in the list such as Disc, Circle or Square.	<UL TYPE = Disc/ Circle/ Square> Example: <UL TYPE = Square>

NOTE: *Type Attribute for Unordered List*

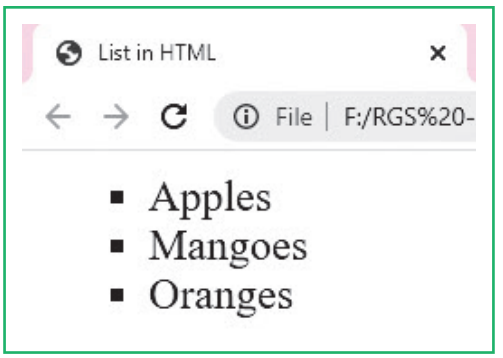
Bullet Style	Output	Description
Disc	●	A filled circle (default)
Circle	○	A non-filled circle
Square	■	A filled square

Example:

Code:



Output:



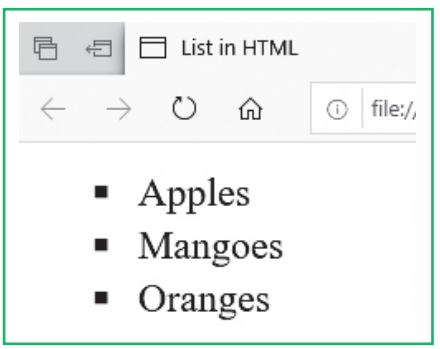
List Properties

Property	Description	Syntax
List-Style-Type	This property specifies the bullet style that will be used as the type of list item marker.	list-style-type: value (where, values for the unordered list is : none | disc | circle | square and value for ordered list is : decimal | lower-roman | upper-roman | lower-alpha | upper-alpha)
List-Style-Image	This property specifies an image as the list item marker.	list-style-image: value (where, value = url ("path of the image")
List-Style-Position	This property specifies the position of the list item marker. (eg – to make them appear inside or outside the content flow)	list-style-position: value (where, value = inside | outside)

Example:

Code:

Output:

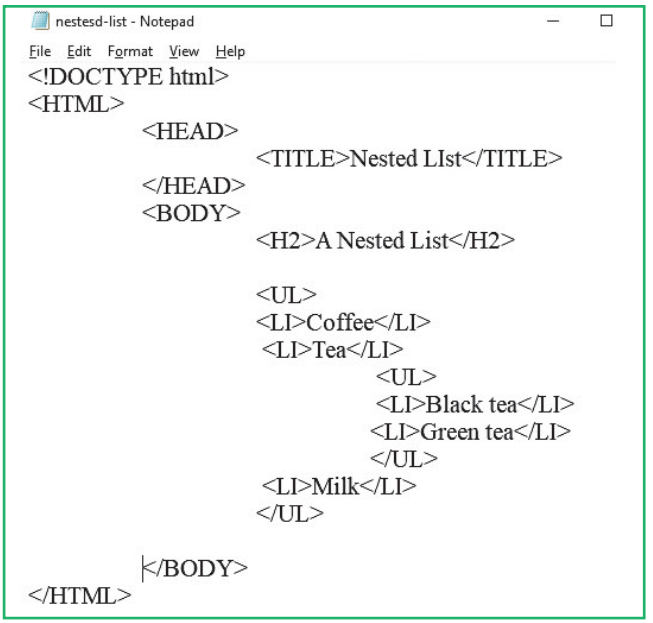


2.21.3 Nested List

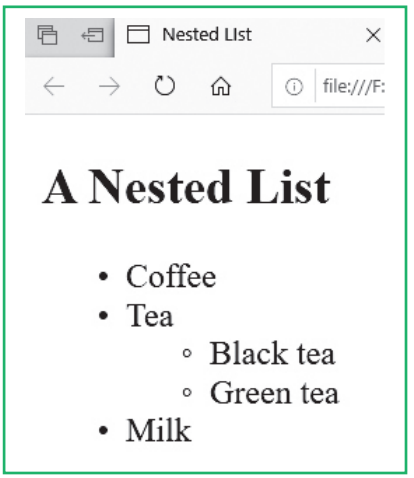
A nested list or a sub-list is a list within a list. It is simply a list that occurs as an element of another list. An ordered and unordered list can be nested within each other to form a multi-level list.

Example:

Code:



Output:



2.21.4 Definition List

A definition list, also called a description list consists of a term followed by its definition. In simple terms, this is a list of items, with a description of each item. It starts and ends with <DL> and </DL> tag.

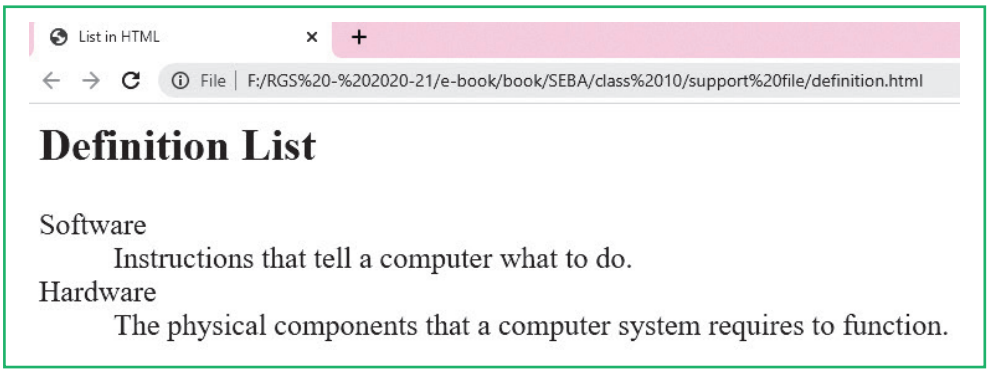
- The <DT> tag is used in defining the terms.
- The <DD> tag is used in describing each term.

Example:

Code:

```
definition - Notepad
File  Edit  Format  View  Help
<HTML>
        <HEAD>
                <TITLE> List in HTML </TITLE>
        </HEAD>
        <BODY>
                <H2> Definition List </H2>
                <DL>
                        <DT>Software</DT>
                                <DD> Instructions that tell a computer what to do.</DD>
                         <DT>Hardware</DT>
                                 <DD>The physical components that a computer system requires to function.</DD>
                </DL>
        </BODY>
</HTML>
```

Output:



2.22 TABLES

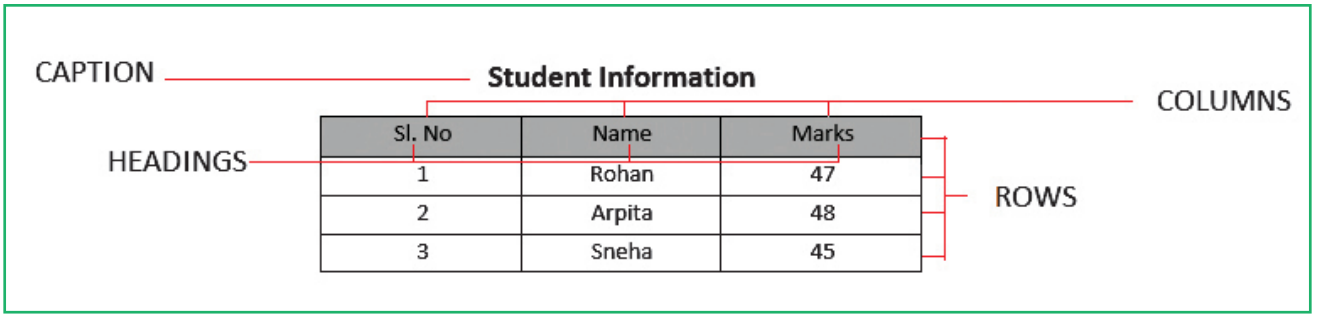
Tables are extremely popular on the web as they are flexible and attractive way of presenting information in the form of rows and columns.

Each table may have an associated caption that provides a short description of the table.

2.22.1 Why tables are useful?

- Tables help in presenting the information or data in a comprehensive manner.
- Tables can be used for comparative analysis of data.
- Information displayed in tables is easier to read and quicker to evaluate.

Let us look at the various terms used in a table.



2.22.2 Various Table Tags

HTML has five core tags.

Tags	Description
<TABLE>	The <TABLE> tag is a basic structural tag that encloses the entire table information. This is a container tag.
<TR>	This is a container tag. This is used to enclose the data of a single row in the table.
<TD>	Within each row of the table, data is represented in the form of individual cells. The <TD> tag is used represent each cell entry of the table.
<TH>	This tag is used to define the headings of the table.
<CAPTION>	This tag defines the title or caption of the table.

2.22.3 Creating a table in HTML

To create a table in HTML, the <TABLE> tag is used. Each table begins with a <TABLE> tag and ends with </TABLE> tag.

Each row in a table begins with the table row <TR> tag and ends with </TR> tag. The rows must always be inside the <TABLE> tag.

To specify a column heading, you use the <TH> tag that is a also a container tag and ends with </TH> tag. It makes the cell content bold format and aligned in the center of the cell.

The columns contain cells, each of which begins with the <TD> tag and ends with </TD> tag. The <TD> tag must always be present inside the row tag <TR>.

Format:

```
<TABLE>
    <CAPTION> The title of the table </CAPTION>
    <TR>
        <TH> Heading 1 </TH>
        <TH> Heading 2 </TH>
        <TH> Heading 3 </TH>
    </TR>
    <TR>
    <TD> Data in Cell 1 </TD>
    <TD> Data in Cell 2 </TD>
    <TD> Data in Cell 3 </TD>
    </TR>
    ------------
    ------------
</TABLE>
```

2.22.4 Attributes of Table Tag

Attribute	Function	Value
WIDTH	Specifies the width of the table.	Pixel or %
ALIGN	Sets the alignment.	Left / right / center
BORDER	Specifies the border width. A value of "0" means no border.	Pixel
BGCOLOR	Specifies the background color of the table.	Color name
CELLPADDING	Specifies the space between the cell borders and their contents.	Pixel or %
CELLSPACING	Specifies the space between cells.	Pixel or %

Example:

Code:

table - Notepad

File Edit Format View Help

```
<BODY>
    <TABLE WIDTH=30% ALIGN=LEFT BORDER=3 BGCOLOR=LIGHTBLUE>
        <CAPTION> Student Details </CAPTION>
        <TR>
            <TH>Sl. No. </TH>
            <TH> Name </TH>
            <TH> Marks </TH>
        </TR>
        <TR>
            <TD> 1 </TD>
            <TD> Rohan </TD>
            <TD> 56</TD>
        </TR>
        <TR>
            <TD> 2 </TD>
            <TD> Arpita </TD>
            <TD> 48</TD>
        </TR>
    </TABLE>
</BODY>
```

Output:

Table in HTML

File | F:/RGS%20-%202020-21/e-book/book/

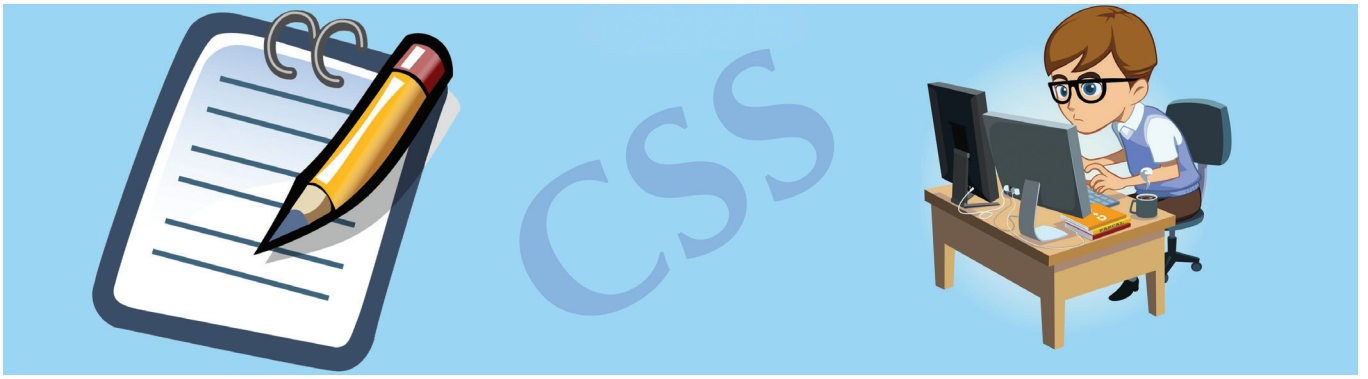
Student Details

Sl. No.	Name	Marks
1	Rohan	56
2	Arpita	48

2.23 STYLING TABLES USING CSS

Internal CSS allows you to style table for a HTML page. It is defined using the <style> tag in the head section of the HTML document.

The following properties can be used while creating a table to enhance its appearance and make it attractive.



2.24 TABLE PROPERTIES

Property	Options	Description	Values
Border	Width	Specify the thickness of the border	Thickness in px, cm or by using one of the three pre-defined values : thin, medium or thick
	Style	Specify the type of border	Dotted | dashed | solid | double | groove | ridge | inset | outset | none | hidden
	Color	Specify the color of the border	Color name
	Radius	Specify rounded borders to an element	In px

Sample of the different type of border is given below:

A dotted border.

A dashed border.

A solid border.

A double border.

A groove border. The effect depends on the border-color value.

A ridge border. The effect depends on the border-color value.

An inset border. The effect depends on the border-color value.

An outset border. The effect depends on the border-color value.

No border.

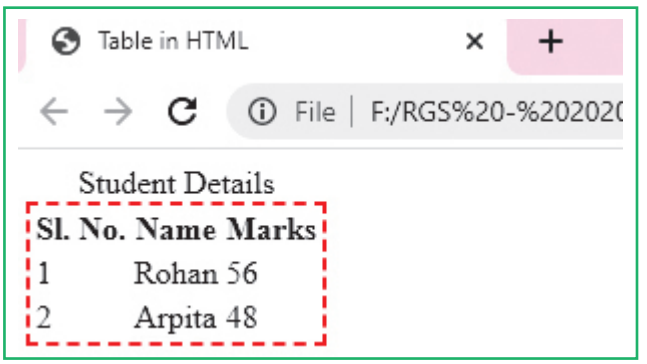
A hidden border.

Example:

HTML code using border property:

```
<HTML>
    <HEAD>
        <TITLE> Table in HTML </TITLE>
        <style type="text/css">
            table
            {
            border: 2px dashed red;
            }
        </style>
    </HEAD>
    <BODY>
        <TABLE>
            <CAPTION> Student Details </CAPTION>
            <TR>
                <TH>Sl. No. </TH>
                <TH> Name </TH>
                <TH> Marks </TH>
            </TR>
            <TR>
                <TD> 1 </TD>
                <TD> Rohan </TD>
                <TD> 56</TD>
            </TR>
            <TR>
                <TD> 2 </TD>
                <TD> Arpita </TD>
                <TD> 48</TD>
            </TR>
        </TABLE>
    </BODY>
</HTML>
```

Output:



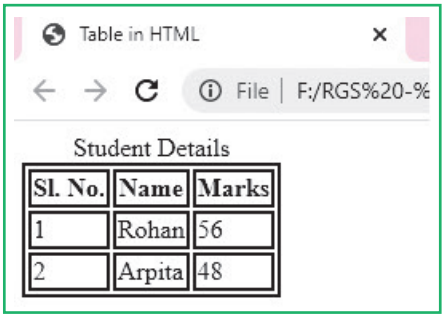
Example:

HTML code for defining a common style for more than one HTML tag.

Code:

```
<HTML>
    <HEAD>
        <TITLE> Table in HTML </TITLE>
        <style type="text/css">
            table, th, td
            {
            border: 2px solid black;
            }
        </style>
    </HEAD>
    <BODY>
        <TABLE>
            <CAPTION> Student Details </CAPTION>
            <TR>
                <TH>Sl. No. </TH>
                <TH> Name </TH>
                <TH> Marks </TH>
            </TR>
            <TR>
                <TD> 1 </TD>
                <TD> Rohan </TD>
                <TD> 56</TD>
            </TR>
            <TR>
                <TD> 2 </TD>
                <TD> Arpita </TD>
                <TD> 48</TD>
            </TR>
        </TABLE>
    </BODY>
</HTML>
```

Output:

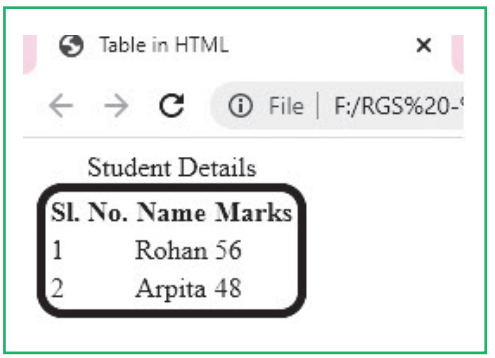


Example:

HTML code defining the rounded border.

```
<HTML>
    <HEAD>
        <TITLE> Table in HTML </TITLE>
        <style type="text/css">
            table
            {
            border: 5px solid black;
            border-radius:15px;
            }
        </style>
    </HEAD>
    <BODY>
        <TABLE>
            <CAPTION> Student Details </CAPTION>
            <TR>
                <TH>Sl. No. </TH>
                <TH> Name </TH>
                <TH> Marks </TH>
            </TR>
            <TR>
                <TD> 1 </TD>
                <TD> Rohan </TD>
                <TD> 56</TD>
            </TR>
            <TR>
                <TD> 2 </TD>
                <TD> Arpita </TD>
                <TD> 48</TD>
            </TR>
        </TABLE>
    </BODY>
</HTML>
```

Output:



2.25 BORDER COLLAPSE PROPERTY

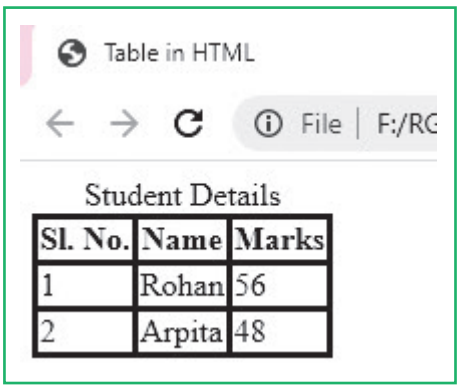
Property	Value	Description
border-collapse	Separate	Applies separate border for each cell
	Collapse	Applies single border for each cell

Example:

HTML code using border-collapse property.

```
<HTML>
    <HEAD>
        <TITLE> Table in HTML </TITLE>
        <style type="text/css">
            table, th, td
            {
            border:3px solid black;
            border-collapse: collapse;
            }
        </style>
    </HEAD>
    <BODY>
        <TABLE>
            <CAPTION> Student Details </CAPTION>
            <TR>
                <TH>Sl. No. </TH>
                <TH> Name </TH>
                <TH> Marks </TH>
            </TR>
            <TR>
                <TD> 1 </TD>
                <TD> Rohan </TD>
                <TD> 56</TD>
            </TR>
            <TR>
                <TD> 2 </TD>
                <TD> Arpita </TD>
                <TD> 48</TD>
            </TR>
        </TABLE>
    </BODY>
</HTML>
```

Output:



2.26 PADDING PROPERTY

Property	Value	Description
padding	length	Specifies the space between cell contents in cm, px or pts.
	%	Specifies the space between cell contents in percentage of the element.

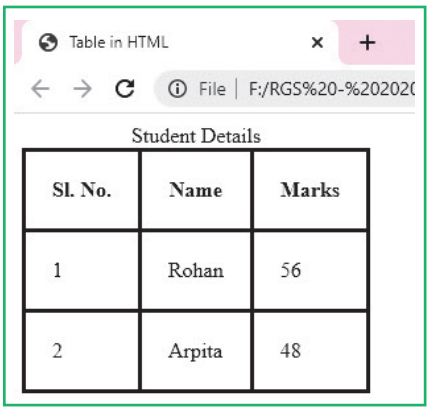
Example:

HTML code using padding property.

```
<HTML>
    <HEAD>
        <TITLE> Table in HTML </TITLE>
        <style type="text/css">
            table, th, td
            {
            border:3px solid black;
            border-collapse: collapse;
            }
            th, td
            {
            padding: 20px;
            }
        </style>
    </HEAD>
    <BODY>
        <TABLE>
            <CAPTION> Student Details </CAPTION>
```

```
                <TR>
                    <TH>Sl. No. </TH>
                    <TH> Name </TH>
                    <TH> Marks </TH>
                </TR>
                <TR>
                    <TD> 1 </TD>
                    <TD> Rohan </TD>
                    <TD> 56</TD>
                </TR>
                <TR>
                    <TD> 2 </TD>
                    <TD> Arpita </TD>
                    <TD> 48</TD>
                </TR>
            </TABLE>
        </BODY>
    </HTML>
```

Output:



2.27 IMAGES IN HTML

Images are very important to beautify as well as to depict many complex concepts in simple way on your web page. Earlier the Web was just text, and it was really quite boring. Fortunately, it wasn't too long before the ability to embed images (and other more interesting types of content) inside web pages was added. There are other types of multimedia to consider, but it is logical to start with <img> element, used to embed a simple image in a webpage.

Images play an important role in any webpage. Though it is not recommended to include a lot of images, but it is still important to use good images wherever required.

The most widely used image formats supported by HTML are GIF, JPEG and PNG. The images on a web page can be inserted using <IMG> tag, which is an empty tag.

Attribute	Function	Syntax
SRC	It stands for source. It specifies the location of the image.	<IMG SRC = “Path of the image”>
ALT	Alternate Text	<IMG SRC = “Path of the image” ALT = “alternate text”>
HEIGHT	Height of the image	<IMG SRC = “Path of the image” HEIGHT= Value in px>
WIDTH	Width of the image	<IMG SRC = “Path of the image” WIDTH = Value in px>
ALIGN	Type of alignment	<IMG SRC = “Path of the image” ALIGN = LEFT (Default) | RIGHT>

Example:

HTML code for inserting an image on a web page.

Code:

*IMAGES - Notepad

File Edit Format View Help

```
<HTML>
        <HEAD>
                <TITLE> Images in HTML </TITLE>

        </HEAD>

        <BODY>
                <H1 ALIGN="CENTER"> Image of Landscape</H1>
                <IMG SRC="LANDSCAPE.JPG" HEIGHT=200 WIDTH=300>
        </BODY>

</HTML>
```

Output:

NOTE: If the image file and the HTML document are in the same folder, then you can write the name of the image file only (without giving the path) in the HTML document to insert an image.

Example:

HTML code to insert an image if the file "Landscape.jpg" lies in a directory other than the one in which the HTML file is placed.

```
<HTML>
        <HEAD>
                <TITLE> Images in HTML </TITLE>

        </HEAD>

        <BODY>
                <H1 ALIGN="CENTER"> Image of Landscape</H1>
                <IMG SRC="C:\Users\user\Documents\LANDSCAPE.JPG" HEIGHT=200 WIDTH=300>
        </BODY>

</HTML>
```

2.28 IMAGES WITH CSS

Property	Description	Value
Border	This is used to set the width of an image border.	in length or in %
Height	This is used to set the height of an image.	in length or in %
Width	This is used to set the width of an image.	in length or in %
Border-radius	This property is used to create rounded imagese	In px

Example:

HTML code to set properties of an image using CSS.

```
*IMAGES - Notepad
File  Edit  Format  View  Help
<HTML>
        <HEAD>
                <TITLE> Images in HTML </TITLE>

        </HEAD>

        <BODY>
                <H1 ALIGN="CENTER"> Image of Landscape using CSS</H1>
                <IMG style="border:2px solid red; height: 500 px; width: 250px; border-radius:50px"
                src="landscape.jpg">
        </BODY>
</HTML>
```

Output

KEYWORDS LEARNED IN THIS CHAPTER

Ordered List	Unordered List	Definition List
Nested List	Tables	Caption

Exercise

I. FILL IN THE BLANKS:

1. By default, the unordered lists items are marked with _____.

2. _____ attribute of list lets you reverse the order of the item list.

3. A list inside another list is called as a _____ list.

4. A collection of related elements is called as _____.

5. _____ property of table defines the space between the content of the table and the border

6. The <img> tag is an _____ tag, that means it has no closing tag.

7. _____ is an attribute of the <img> tag which specifies the location or URL of the image to be displayed.

8. _____ attribute is used to give border to an image.

II. MULTIPLE CHOICE QUESTIONS:

1. Which tag is used for List items?
   a. <OL> b. <LI> c. <UL> d. <DL>

2. Which element contains definition?
   a. <DL> b. <DD> c. <DT> d. <UL>

3. Which of the following can't be the value of list-style-type?
   a. Square b. Circle c. Ellipse d. Disc

4. Which attribute is only used with ?
   a. Value b. type c. compact d. start

5. With the help of which tag, is a row defined in HTML?
   a. <row> c. <row-table>
   b. <tr> d. <tablerow>

6. By using which of the following options, the border of table can be collapsed?
   a. border-collapse: collapse
   b. table-border: collapse
   c. border: collapse
   d. table-border-collapse: collapse

7. Web browsers display images in the following format
   a. XBM c. JPEG
   b. GIF d. All of these

8. The correct HTML code for inserting an image is
   a. <img href=”image.gif”>
   b. <img> image.gif</gif>
   c. <img src = “image.gif”>
   d. <image src = “image.gif” >

9. src attribute used with <img> tag stands for
   a. screen c. screen resolution count
   b. source d. structure

10. alt attribute allows
a. addition of an alternate hyperlink
b. addition of a border to image
c. use of an alternative image in place of the specified image
d. addition of alternative text about an image

III. APPLICATION BASED QUESTIONS:

a. Ruchika was making an ordered list and she noticed that the items of the list by default started with numbers. She wants to use Roman numerals for numbering. How can she do this?

b. Rohan has created a table and he wants that the table border should be collapsed into a single border. Which property should he use?

c. Ashmita has added few images on her web page but she wants to keep some provisions for the visually impaired people or users using text-based browsers so that they get the description for the images. Which attribute should she use to accomplish the task?

IV. ANSWER THE FOLLOWING:

a. Differentiate between the <OL> and <UL> tag.

b. Write the syntax for using list-style-type property.

c. Define Padding property.

d. What is description list? Define the different tags used to create a description.

e. What is the use of type attribute with an unordered list?

f. State the use of any two properties that you use to enhance the appearance of a table.

g. How are images added in an HTML document?

V. ACTIVITY

1. Create a web page to show a list of various colours and their Hex codes (Hexadecimal numerals are widely used by computer system designers and programmers because they provide a human-friendly representation of binary-coded values). Also provide image for the colors you mention in your code.

2. Create a web page to form a tabular representation of different mobile names with their features.

PART - III Link, Frames And Forms

2.29 INTRODUCTION

A website is a collection of web pages which are interlinked with each other and contains related information. HTML renders a powerful feature of linking these web pages together. This feature is called hyperlink. A hyperlink is an underlined text which when clicked will take you to another web page. Generally, hyperlinks are seen in blue color and when you hover the mouse pointer over a link, it will turn into a little hand.

2.30 TYPES OF LINKING

HTML allows you to create two types of linking.

- **Internal linking:** When one part of a web page is linked to another section on the same web page, it is called internal linking. In this case, the hyperlink and the linked section appear on the same web page.

- **External linking:** When one page is linked to another web page of the same website or another website, it is called external linking.

2.31 CREATING LINKS

In HTML, the Anchor tag <A>, is used to mark the text as a hyperlink. An anchor tag is a container tag, which requires closing tag </A> to mark the end of the text or image.

Attribute	Description
HREF	Specifies the URL of a page that the link goes to.
TARGET	Specifies the default browsing context to load the URL into. Only to be used when the href attribute is present. Possible values: • _blank • _self • _top • _parent

The "href" attribute is the most important attribute of the HTML a tag. and which links to destination page or URL.

The href attribute is used to define the address of the file to be linked. In other words, it points out the destination page.

The syntax of HTML anchor tag is given below.
 Text to be linked

Links can be text or images. When a user hovers the mouse pointer over a link, the pointer takes the shape of a pointing hand, indicating the presence of an active link. Clicking on a hyperlink takes you to a web page that the web page is linked. The address of the web page is specified in the HREF attribute.

Example:

HTML code to create a text hyperlink on a web page.

Code:

```
<HTML>
        <HEAD>
                <TITLE> Hyperlink in HTML </TITLE>

        </HEAD>

        <BODY>
                <H1 ALIGN="CENTER"> Text Hyperlink </H1>
                <A HREF = "SECOND.HTML">CLICK TO VIEW THE LINKED WEB PAGE</A>
        </BODY>
</HTML>
```

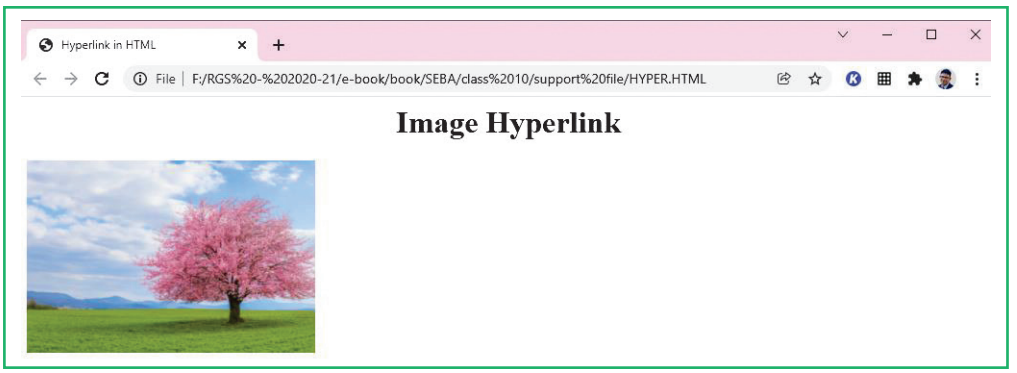
Output

Example:

HTML code to create an image hyperlink on a web page.

```
<HTML>
        <HEAD>
                <TITLE> Hyperlink in HTML </TITLE>

        </HEAD>

        <BODY>
                <H1 ALIGN="CENTER"> Image Hyperlink </H1>
                <A HREF = "SECOND.HTML"><IMG SRC="LANDSCAPE.JPG"
                HEIGHT=200 WIDTH = 300></A>
        </BODY>
</HTML>
```

Output:



NOTE: Remember, if the linked HTML document exists in different folder, you must specify the complete path.

Appearance of HTML anchor tag

- An unvisited link is displayed underlined and blue.
- A visited link displayed underlined and purple.
- An active link is underlined and red.

2.32 HYPERLINK WITH CSS

CSS affects the links differently depending on the state they are in. The following are the properties of a hyperlink using CSS.

Properties	Description
a:link	It signifies the unvisited hyperlinks
a:visited	It signifies the visited hyperlinks
a:hover	It signifies on element on which the user's mouse is hovering over it
a:active	It signifies the element on which the user is currently clicking

Example:

HTML code using hyperlink properties.

```
<HTML>
        <HEAD>
                <TITLE> Hyperlink in HTML </TITLE>
                        <style type = "text/css">
                                a:link{color:black; text-decoration: overline}
                                a:visted{color:yellow;}
                                a:hover{color:purple;}
                                a:active{color:green;}
                        </style>
        </HEAD>
        <BODY>
                <H1 ALIGN="CENTER"> Hyperlink </H1>
                <A HREF = "SECOND.HTML">This is a link</A>
        </BODY>
</HTML>
```

Output:

2.33 LINKS AS BUTTONS

CSS properties allows you to display links as buttons.

Example:

Code:

```
<HTML>
        <HEAD>
                <TITLE> Hyperlink in HTML </TITLE>
                        <style type = "text/css">
                                a:link, a:visited
                                {
                                background-color: yellow;
                                color:red;
                                text-decoration: none;
                                padding: 15px 20px;
                                text-align: center;
                                display: inline-block;
                                }

                        </style>
        </HEAD>
        <BODY>
                <H1 ALIGN="CENTER"> Hyperlink </H1>
                <A HREF = "SECOND.HTML">This is a button link</A>
        </BODY>
</HTML>
```

Output:

2.34 AUDIO AND VIDEO

Now that we are comfortable with adding simple images to a webpage, the next step is to start adding video and audio players to your HTML documents. In this section, you will know about the tags with which you will be able to insert multimedia files in your HTML documents. Multimedia refers to 'multiple mediums' – the ability to add sound and moving objects to a web page.

Inserting Audio

The HTML <audio> element is used to embed sound content in documents. It may contain one or more audio sources, represented using the src attribute or the <source> element: the browser will choose the most suitable one.

HTML5 most commonly used audio formats are ogg, mp3 and wav. You can use <source> tag to specify media along with media type and many other attributes. An audio element allows multiple source elements and browser will use the first recognized format.

The attributes of <audio> tag are given in the following table.

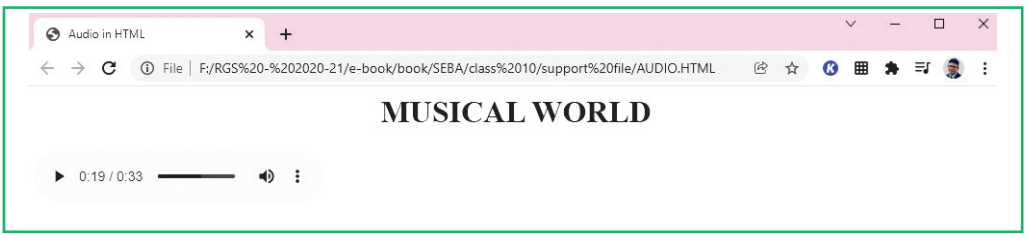
Attribute	Description
src	Specifies the URL (path) of the audio file
controls	Displays the controls on the web page
autoplay	Plays the audio file automatically when the web page is loaded
loop	Replays the audio file

Example:

Code:

```
<HTML>
        <HEAD>
                <TITLE> Audio in HTML </TITLE>

        </HEAD>
        <BODY>
                <H1 ALIGN="CENTER"> MUSICAL WORLD </H1>
                <AUDIO SRC="SAMPLE-AUDIO.WAV" AUTOPLAY CONTROLS>
        </BODY>
</HTML>
```

Output:



Inserting Video

You can insert video file in your HTML tag using <video> tag. Supported file formats include – mp4, webm etc..

The attributes of <video> tag are given in the following table.

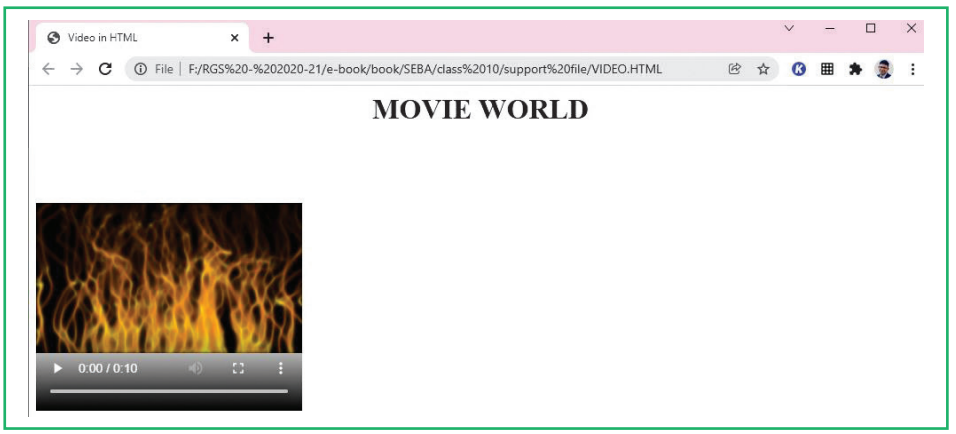
Attribute	Description
src	Specifies the URL (path) of the video file
controls	Displays the controls on the web page
autoplay	Plays the video file automatically when the web page is loaded
height	Specifies the height of the video player displayed
width	Specifies the width of the video player displayed

Example:

Code:

```
<HTML>
        <HEAD>
                <TITLE> Video in HTML </TITLE>

        </HEAD>
        <BODY>
                <H1 ALIGN="CENTER"> MOVIE WORLD </H1>
                <VIDEO SRC="SAMPLE-VIDEO.MP4" WIDTH = 300 HEIGHT = 300
                AUTOPLAY CONTROLS>
        </BODY>
</HTML>
```

Output:



2.35 FRAMES AND iFRAMES

Frames allow you to divide the web page into several independent window allowing multiple views in one time. These also help in making one part static and while other parts to change as per other command.

A collection of frames in a web browser is called a frameset.

In HTML5, frames are created using <iframe> tag. The iframe in HTML stands for Inline Frame. The “iframe ” tag defines a rectangular region within the document in which the browser can display a separate document, including scrollbars and borders.

An inline frame is used to embed another document within the current HTML document. The HTML iframe name attribute is used to specify a reference for an <Iframe> element.

Syntax:

<iframe src="URL" title="description"></iframe>

The ‘ **src** ‘ attribute is used to specify the URL of the document that occupies the iframe and title attribute is used to describe the content of the iframe.

2.35.1 Attributes of iFrame are:

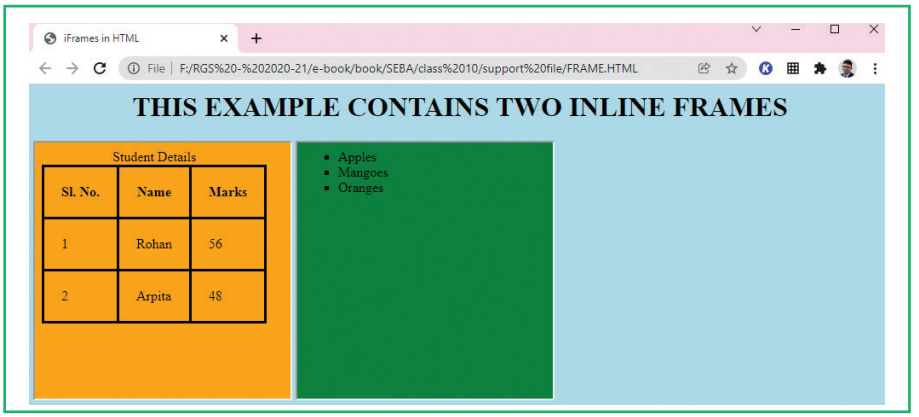
Attribute	Description	Value
height	It is used to control the height of the iframe	Specified in px or %
width	It is used to control the width of the iframe	Specified in px or %

Example:

Code:

```
<HTML>
        <HEAD>
                <TITLE> iFrames in HTML </TITLE>

        </HEAD>
        <BODY BGCOLOR=LIGHTBLUE>
                <H1 ALIGN="CENTER"> THIS EXAMPLE CONTAINS TWO INLINE FRAMES </H1>
                <IFRAME SRC = "TABLE.HTML" WIDTH=300 HEIGHT = 300> </IFRAME>
                <IFRAME SRC = "LIST.HTML" WIDTH=300 HEIGHT = 300> </IFRAME>
        </BODY>
</HTML>
```

Output:



2.36 iFRAMES WITH CSS

Border Property

You can display frames with or without border by applying CSS property – border. Border properties can be applied in the following ways:

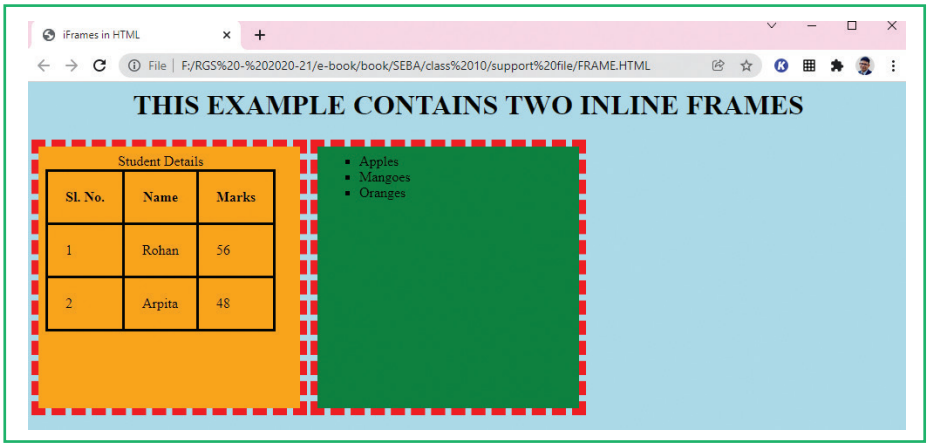
- Border-width : value , where value = thin, thick, medium or numeric values
- Border-style: value, where value = none, hidden, dotted, dashed, solid, double, groove, ridges, inset or outset
- Border-color : value, where value = color name

Example

Code:

```
<HTML>
        <HEAD>
                <TITLE> iFrames in HTML </TITLE>
                <style type = "text/css">
                iframe { border : 8px dashed red}
                </style>
        </HEAD>
        <BODY BGCOLOR=LIGHTBLUE>
                <H1 ALIGN="CENTER"> THIS EXAMPLE CONTAINS TWO INLINE FRAMES </H1>
                <IFRAME SRC = "TABLE.HTML" WIDTH=300 HEIGHT = 300> </IFRAME>
                <IFRAME SRC = "LIST.HTML" WIDTH=300 HEIGHT = 300> </IFRAME>
        </BODY>
</HTML>
```

Output:



2.37 FORMS IN HTML

HTML Forms are required, when you want to collect some data from the site visitor. For example, during user registration you would like to collect information such as name, email address, credit card, etc.

The HTML <form> element provide a document section to take input from user. It provides various interactive controls for submitting information to web server such as text field, text area, password field, etc.

A form will take input from the site visitor and then will post it to a back-end application such as ASP Script or PHP script etc. The back-end application will perform required processing on the passed data based on defined business logic inside the application.

The **form** tag is used to create an HTML form.

2.37.1 HTML Form Controls

There are different types of form controls that you can use to collect data using HTML form -

- Text Input Controls
- Checkboxes Controls
- Radio Box Controls
- Select Box Controls
- File Upload Controls
- Field set Control
- Submit and Reset Button

Text Input Controls

There are three types of text input used on forms -

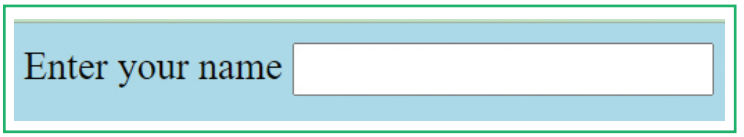
- **Single-line text input controls** - This control is used for items that require only one line of user input, such as search boxes or names. They are created using HTML <input> tag.

Example:

A form with input fields for text

```
<HTML>
        <HEAD>
                <TITLE> Forms in HTML </TITLE>
        </HEAD>
        <BODY BGCOLOR=lightblue>
                <FORM>
                        Enter your name <INPUT TYPE = TEXT NAME = "FNAME">

                </FORM>
        </BODY>
</HTML>
```

Output:



- **Password input controls** - This is also a single-line text input but it masks the character as soon as a user enters it. They are also created using HTML <input> tag.

Example:
A form to enter the password which is not visible to the user in password field control.

```
<HTML>
        <HEAD>
                <TITLE> Forms in HTML </TITLE>
        </HEAD>
        <BODY BGCOLOR=lightblue>
                <FORM>
                        Enter your password <INPUT TYPE = PASSWORD NAME = "PWORD">

                </FORM>
        </BODY>
</HTML>
```

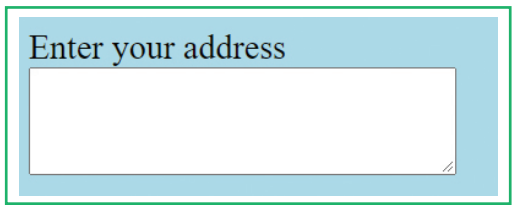
Output:

- **Multi-line text input controls** - This is used when the user is required to give details that may be longer than a single sentence. Multi-line input controls are created using HTML <textarea> tag.

Example:
A form to enter multi-line text.

```
<HTML>
        <HEAD>
                <TITLE> Forms in HTML </TITLE>
        </HEAD>
        <BODY BGCOLOR=lightblue>
                <FORM>
                        Enter your address <BR>
                        <TEXTAREA ROWS = 3 COLS=25>
                        </TEXTAREA>

                </FORM>
        </BODY>
</HTML>
```

Output:



Checkbox Control

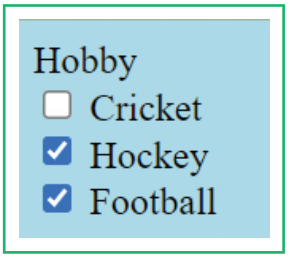
Checkboxes are used to let the user select one or more options from a pre-defined set of options. Checkbox input controls are created using the "input" element with a type attribute having value as "checkbox".

Example

A form to select hobbies.

```
<HTML>
        <HEAD>
                <TITLE> Forms in HTML </TITLE>
        </HEAD>
        <BODY BGCOLOR=lightblue>
                <FORM>
                        Hobby <BR>
                        <INPUT TYPE = CHECKBOX NAME = CRICKET> Cricket  <BR>
                         <INPUT TYPE = CHECKBOX NAME =HOCKEY> Hockey <BR>
                        <INPUT TYPE = CHECKBOX NAME = FOOTBALL> Football <BR>
                </FORM>
        </BODY>
</HTML>
```

Output:



Radio Button Control

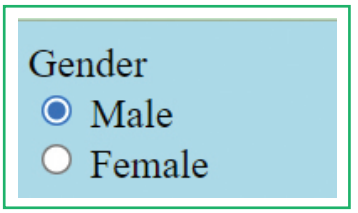
Radio buttons are used when out of many options, just one option is required to be selected. Radio Button input controls are created using the "input" element with a type attribute having value as "radio".

Example:

A form to select gender.

```
<HTML>
        <HEAD>
                <TITLE> Forms in HTML </TITLE>
        </HEAD>
        <BODY BGCOLOR=lightblue>
                <FORM>
                        Gender <BR>
                        <INPUT TYPE = RADIO NAME = GENDER> Male  <BR>
                         <INPUT TYPE = RADIO NAME =GENDER> Female

                </FORM>
        </BODY>
</HTML>
```

Output



NOTE: Radio button can select one button at a time while Checkbox can choose multiple options at a time.

Select Box Control

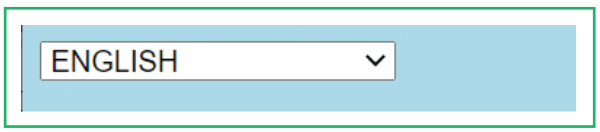
A select box, also called drop down box which provides option to list down various options in the form of drop-down list, from where a user can select one or more options.

Example :

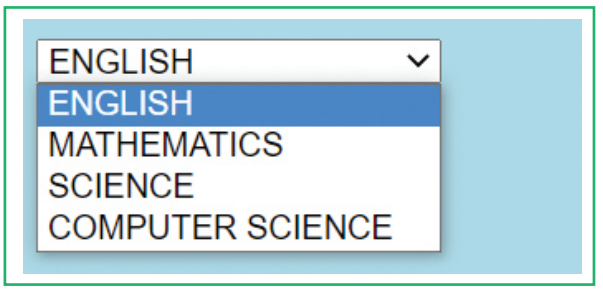
A form to select a subject.

```
<HTML>
        <HEAD>
                <TITLE> Forms in HTML </TITLE>
        </HEAD>
        <BODY BGCOLOR=lightblue>
                <FORM>
                        <SELECT NAME = SUBJECT>
                                <OPTION SELECTED> ENGLISH</OPTION>
                                <OPTION> MATHEMATICS </OPTION>
                                <OPTION> SCIENCE </OPTION>
                                <OPTION> COMPUTER SCIENCE</OPTION>
                        <SELECT>
                </FORM>
        </BODY>
</HTML>
```

Output:



Output – 1



Output – 2

File Upload Box

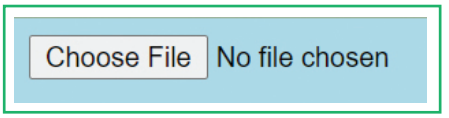
If you want to allow a user to upload a file to your web site, you will need to use a file upload box, also known as a file select box. File Upload Box controls are created using the “input” element with a type attribute having value as “file”.

Example:

A form to choose a file.

```
<HTML>
        <HEAD>
                <TITLE> Forms in HTML </TITLE>
        </HEAD>
        <BODY BGCOLOR=lightblue>
                <FORM>
                        <INPUT TYPE = FILE NAME = "FILE-UPLOAD">
                </FORM>
        </BODY>
</HTML>
```

Output:



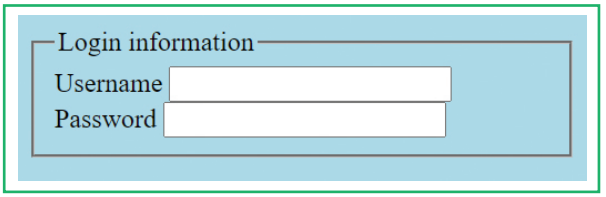
Field Set Control

The <fieldset> element in HTML is used to group the related information of a form. This element is used with <legend> element which provide caption for the grouped elements.

Example:

```
<HTML>
        <HEAD>
                <TITLE> Forms in HTML </TITLE>
        </HEAD>
        <BODY BGCOLOR=lightblue>
                <FORM>
                        <FIELDSET>
                                <LEGEND> Login information </LEGEND>
                                Username <INPUT TYPE = TEXT NAME = "USER-NAME">
                                <BR>
                                Password <INPUT TYPE = PASSWORD NAME = "PWARD">
                        </FIELDSET>
                </FORM>
        </BODY>
</HTML>
```

A form to show the uses of <fieldset> element.



Output:

Button Controls

There are various ways in HTML to create clickable buttons. You can also create a clickable button using <input>tag by setting its type attribute to button. The type attribute can take the following values -

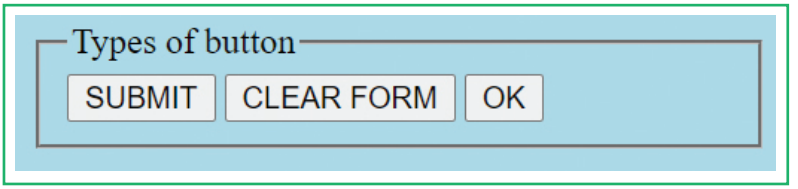
Type	Description
Submit	This creates a button that automatically submits a form.
Reset	This creates a button that automatically resets form controls to their initial values.
Button	This creates a button that is used to trigger a client-side script when the user clicks that button.

Example:

A form with three types of button.

```
<HTML>
        <HEAD>
                <TITLE> Forms in HTML </TITLE>
        </HEAD>
        <BODY BGCOLOR=lightblue>
                <FORM>
                        <FIELDSET>
                                <LEGEND> Types of button</LEGEND>
                                <INPUT TYPE = SUBMIT NAME = "SUBMIT" VALUE="SUBMIT">
                                <INPUT TYPE = RESET NAME = "RESET" VALUE="CLEAR FORM">
                                <INPUT TYPE = BUTTON NAME = "BUTTON" VALUE="OK">
                        </FIELDSET>
                </FORM>
        </BODY>
</HTML>
```

Output:



KEYWORDS LEARNED IN THIS CHAPTER

Linking	Hyperlink	Anchor tag	Frames
Forms	Input	Radio button	Checkbox

Exercise

I. FILL IN THE BLANKS:

1. The web pages of a website are linked to each other using _____.
2. The attribute _____ is used to create a hyperlink between two or more HTML codes.
3. When you move the mouse pointer over a link, the mouse pointer changes its shape from an arrow to a _____.
4. The _____ attribute of the <audio> tag indicates that you can replay the audio file once it is finished.
5. The small rectangular areas created in the main browser window are known as _____
6. The _____ attribute of the frame tag tells the browser which HTML page to load into that frame.
7. _____attribute of the frame tag attaches the default URL.
8. _____ allow multiple HTML documents to be presented as independent windows within one browser window.
9. The _____ tag collects the information from the user.

II. MULTIPLE CHOICE QUESTIONS:

1. A _____ is a word, a group of words, or an image that can be used to jump to another document on the same website or another website.
   i. Hyperlink ii. URL iii. Address iv. none of these
2. The _____ attribute of the <a> tag is used to set the URL of the target resource.
   i. src ii. href iii. Controls iv. none of these
3. Which of the following can be embedded in a web page?
   i. Audio ii. Video iii. Both (i) and (ii) iv. None of these
4. The _____ attribute of the <video> tag plays the video file automatically on loading a web page.
   i. controls ii. Autoplay iii. Height iv. none of these
5. _____ tag is used to create textbox, radio button and checkbox on the web page.
   i. <OPTION> ii. <INPUT> iii. Both of these iv. None of these

III. APPLICATION BASED QUESTIONS:

1. Rohan wants to divide a web page into four sections. Which tag should he use to accomplish this?
2. Rahim is creating a website in which he wants to use different images as links to the web pages. He is also interested in adding some video clips in his website. Can you suggest him the required tags to include the said elements in his website.
3. Ritika was writing an article using HTML. The article contains some external links to other website contains additional information. How can she link these together so that the user can visit the destination of the external links by clicking on them?
4. Priyanka wants to create a form but she has forgotten the tag used to create the form. Can you help her with the solution?

IV. ANSWER THE FOLLOWING:

1. Why do you include hyperlinks in your web page? Give any two reasons.
2. Distinguish between the internal and external linking.
3. What are frames? How are they useful?
4. What are two types of text input in HTML web forms?
5. Which input control is most useful for questions requiring a simple yes or no answer?
6. What is the use of password control in HTML forms?
7. What is the use of <INPUT> tag?
8. What are the uses of Submit and Reset buttons?

V. LAB ACTIVITY

1. Create a web page where a person has top choose a day of the month from a drop – down list. The page should also allow the person to enter two lines on what the person plans to do on.

2. Create an HTML document on the topic HTML list and HTML Tables. The web page should contain two frames where in one frame with HTML List and in other HTML Tables.

CHAPTER 3

Database Part-II MySQL

In this chapter

We will assimilate knowledge about a popular RDBMS which is MySQL. We will learn about MySQL commands, Creating a Database & Tables, Data Types, Inserting Data in the Table, Changing the Structure of a Table, Select statement, Working with Operators, Sorting / Deleting / Updating Data in a Table and Aggregate or Group Functions.

A database is an organized collection of structured information or data stored electronically in a computer system. A database is usually controlled by a DBMS (Data Base Management System).

Nowadays, we use relational database management systems (RDBMS) to store and manage huge volumes of data. This is called relational database because all the data is stored into different tables and relations are established using primary keys or other keys known as Foreign Keys.

You have learnt about Access 2010. Now, you will learn about MySQL, which is the most popular Open-Source Relational Database Management System.

3.1 MySQL & ITS IMPORTANCE

MySQL was created by a Swedish company MySQL AB in 1995. The developers of the platform were Michael Widenius (Monty), David Axmark and Allan Larsson. MySQL is named after co-founder Michael Widenius's daughter, My.

MySQL is becoming so popular because of many good reasons -

- MySQL is released under an open-source license. So you have nothing to pay to use it.
- MySQL is a very powerful program in its own right. It handles a large subset of the functionality of the most expensive and powerful database packages.
- MySQL works on many operating systems and with many languages including PHP, PERL, C, C++, JAVA, etc.
- MySQL works very quickly and supports large databases, up to 50 million rows or more in a table. The default file size limit for a table is 4GB, but you can increase this (if your operating system can handle it).
- MySQL is customizable. The open-source license allows programmers to modify the MySQL software to fit their own specific environments.

It is pronounced as 'My Sequel'. You can download it from the website:

http://dev.mysql.com/downloads.

We will be using MySQL Server 5.6 in this chapter.

3.1.1 Types of SQL Commands

There are five types of SQL commands: DDL, DML, DCL, TCL, and DQL.

1. Data Definition Language (DDL)

DDL changes the structure of the table like creating a table, deleting a table, altering a table, etc. All the commands of DDL are auto-committed, which means it permanently saves all the changes in the database.

Here are some commands that come under DDL:

- CREATE
- ALTER
- DROP
- TRUNCATE

2. Data Manipulation Language

DML commands are used to modify the database. It is responsible for all forms of changes in the database. The command of DML is not auto-committed, which means it can't permanently save all the changes in the database. They can be rollback.

Here are some commands that come under DML:

- INSERT
- UPDATE
- DELETE

3. Data Control Language

DCL commands are used to grant and take back authority from any database user.

Here are some commands that come under DCL:

- GRANT
- REVOKE

4. Transaction Control Languag

TCL commands can only be used with DML commands like INSERT, DELETE and

UPDATE only. These operations are automatically committed in the database that's why they cannot be used while creating tables or dropping them.

Here are some commands that come under TCL:

- COMMIT
- ROLLBACK
- SAVEPOINT

5. Data Query Language

DQL is used to fetch the data from the database.

It uses the commands:

- SELECT
- SHOW
- HELP

3.2 STARTING MYSQL

After installing the database from the given link, you can start working with it in the following manner:

1. Click Start > MySQL5.6 Command Line Client (Fig. 3.1).

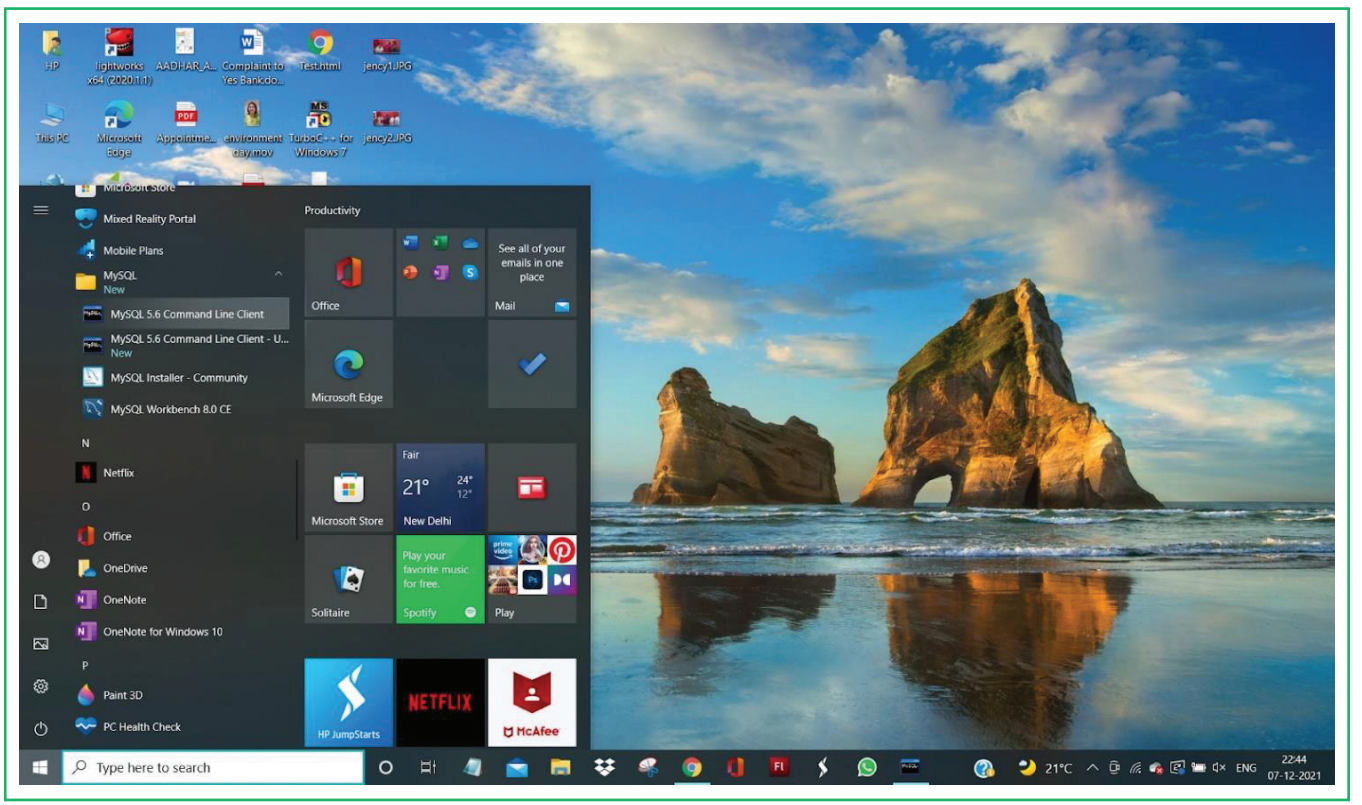


Fig.3.1 MySQL5.6 Command Line Client

2. The MySQL 5.6 Command Line Client window opens (Fig. 3.2).

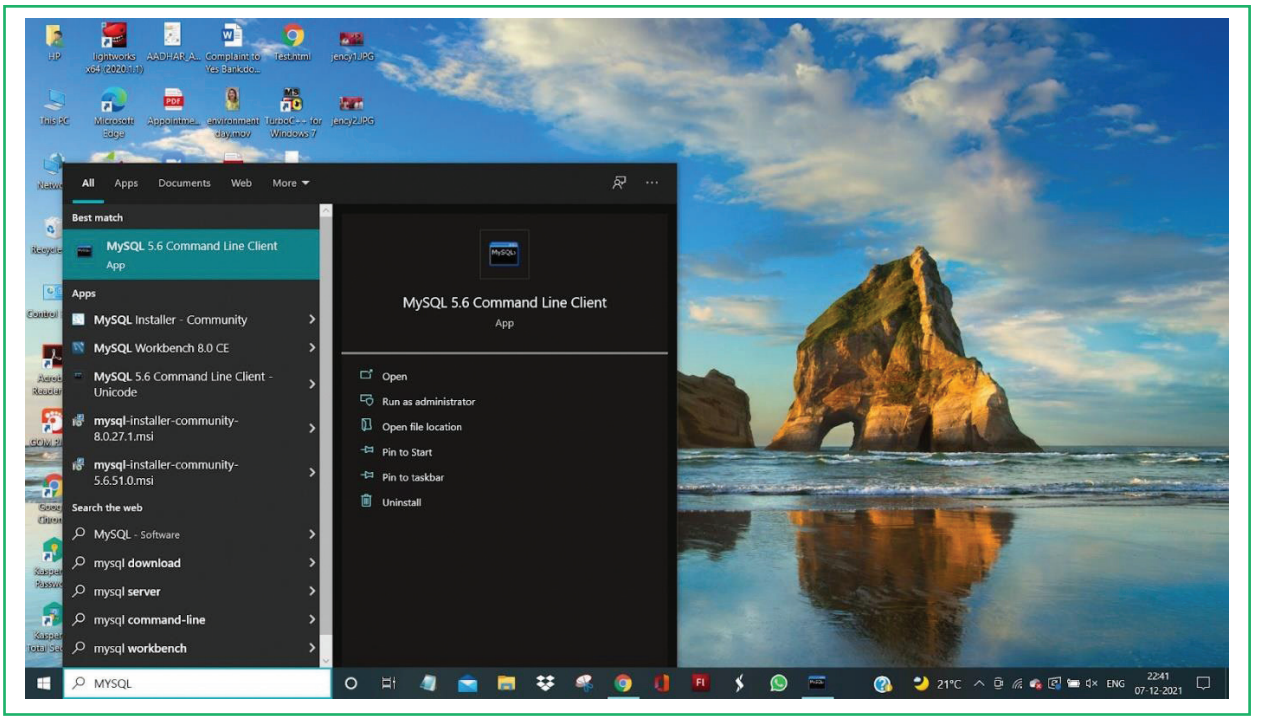


Fig 3.2 Command Line Client Window Opens

3. Click Open. The MySQL 5.6 Command Line Client prompts for a password. Type the password and press Enter. (Fig. 3.3).

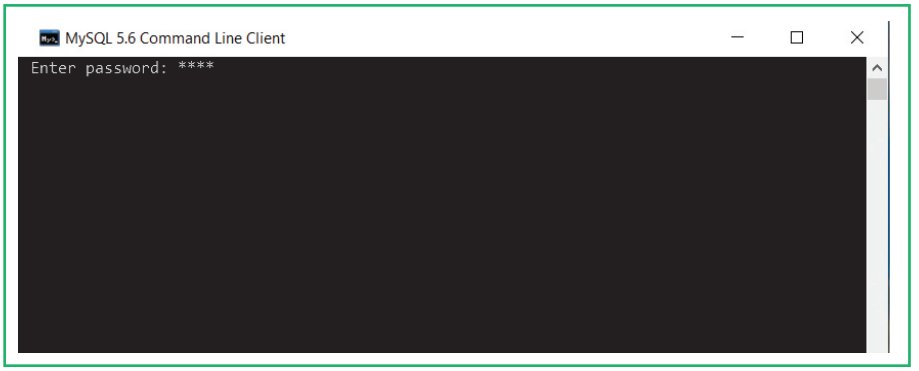


Fig 3.3 Entering Password in MySQL

4. Once the password is verified, the prompt changes to mysql> as shown in (Fig. 3.4)

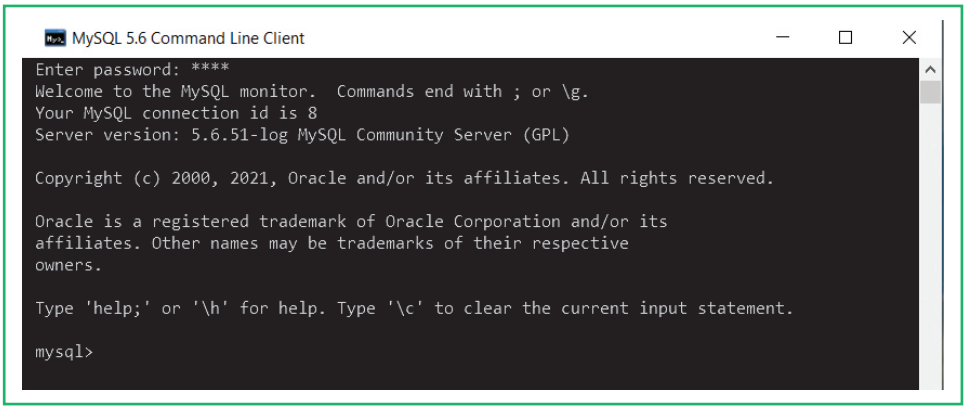


Fig 3.4 MySQL Prompt

We can see the current version of the text that appears before the prompt. However, we can also use the command:

SELECT VERSION();

This command is used to know about the MySQL version we are working with. (Fig.3.5)

Fig 3.5 Select Version

To view the existing databases on the MySQL server, the query is:

SHOW DATABASES;

MySQL displays a list of all the databases, as shown in (Fig.3.6).

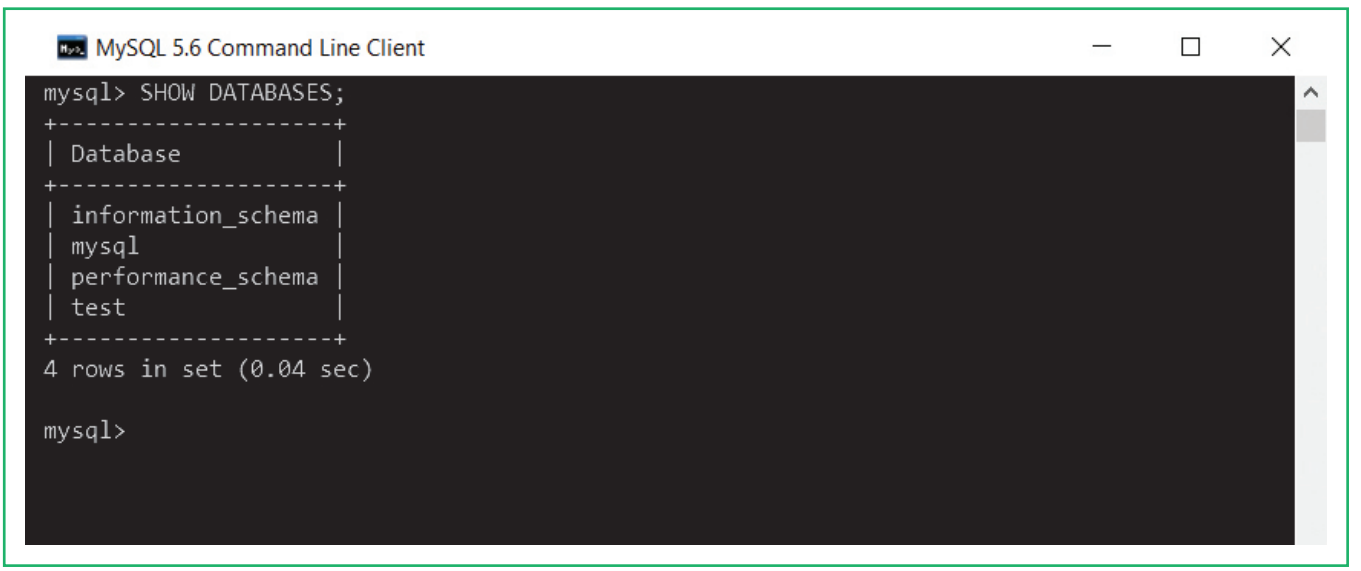


Fig 3.6 Show databases

3.2.1 Creating a Database

To create a new database in MySQL, the query is:
CREATE DATABASE (IF NOT EXISTS] database name;

1. The database name is the name of the database you want to create. The name should be meaningful and descriptive.

2. The IF NOT EXISTS is an optionasl element of the statement. It gives an error while ceating a new database that already exists in the installed MySQL server.
Figure 3.7 shows creation of the database SCHOOL through the query:
CREATE DATABASE SCHOOL;

3.2.2 Selecting a Database For Work

To select a database you want to work with, the query syntax is :

USE database name;

As we are working with the SCHOOL database here, the actual query will be

USE SCHOOL;

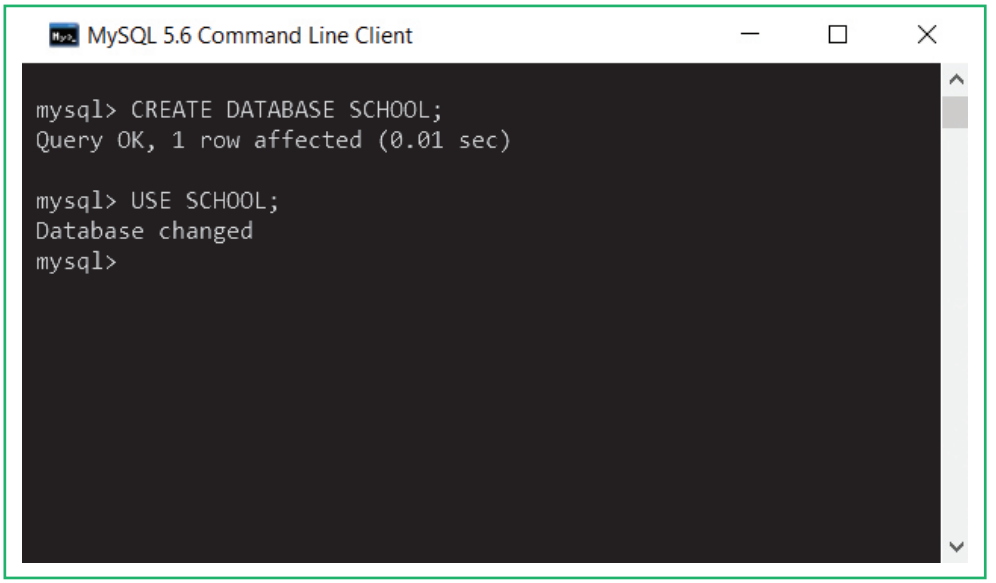


Fig 3.7 Creating & Selecting a database

Now, SCHOOL is the activated database. You can create a table in the SCHOOL database.

3.2.3 Removing a Database

Removing the database means to delete the database physically. The entire data and all related objects inside the database are deleted permanently and this cannot be undone. Therefore, you must delete a database with due care.

DROP DATABASE [IF EXISTS] database_name;

The database_ name is the name of the database that you want to remove. The IF EXISTS is an optional part of the statement and it prevents you from removing the database that does not exist.

Here, the query will be (Fig. 3.8):

DROP DATABASE SCHOOL;

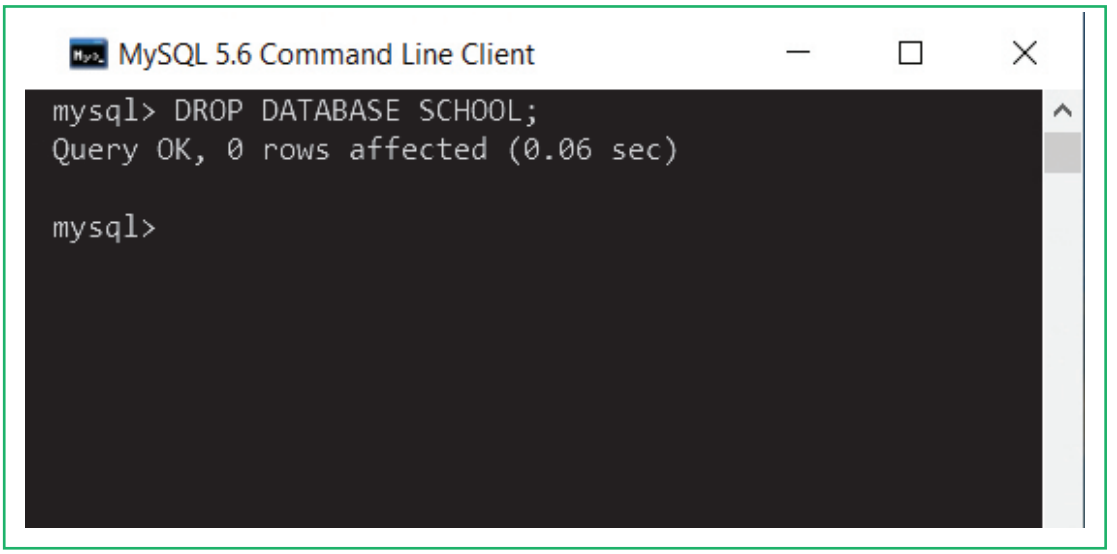


Fig. 3.8 Removing a database

3.3 MYSQL DATA TYPES

A table in a database consists of many fields with specific data types, such as numeric, string or date. The data type can be determined by the: (i) Number of bytes it takes for storage. (ii) Kind of value it represents.

The table below lists the various data types supported by MySQL:

DATA TYPE	SPECIFICATION
CHAR	String (0 – 255)
VARCHAR	String (0 – 255)
TEXT	String (0 – 65535)

SMALLINT	Integer (-32768 to 32767)
INT	Integer (-2147483648 to 2147483647)
FLOAT	Decimal (precise to 23 digits)
DOUBLE	Decimal (24 to 53 digits)
DATE	YYYY-MM-DD
DATETIME	YYYY-MM-DD HH:MM:SS
TIME	HH:MM:SS
YEAR	YYYY

3.3.1 MySQL Tables

A table consists of rows and columns. A Column denotes a field/attribute and rows contain the data for each record/tuple of the fields. The number of rows denotes the **Cardinality** of the table. The number of columns denotes the **Degree** of the table.

Rules for Naming a Table

- A table name can have a maximum of 30 characters.
- It can contain characters A-Z, a-z and numbers 0-9.
- A table name should begin with an alphabet.
- The special character _ (Underscore) is allowed. It is used for joining two words.
- Reserved words are not allowed. For example, CREATE, SELECT, DROP etc.

3.3.2 Creating a Table

In order to create a table, the CREATE TABLE command is used. In the CREATE TABLE command, each field of the table is defined with three attributes - name of the field, data type of the field and size of the field.

Each table field definition is separated by a comma. A MySQL statement is terminated with a semicolon. The syntax of the CREATE TABLE command is:

CREATE TABLE [IF NOT EXISTS] <table name>(<column_name1><data_type> [size] [PRIMARY KEY] [NOT NULL | NULL], <column_name2><data_type> [size] [PRIMARY KEY] [NOT NULL | NULL], _ _ _);

- The column_name specifies the name of the column. Each column_name has a tata type and a size.
- The NOT NULL or NULL indicates that the column accepts NULL values or not.
- The PRIMARY KEY indicates that the particular columns are set as the primary key.

3.3.3 Constraints

In MySQL, data validation can be ensured through Constraints while entering data in a table. Constraints are some rules that help ensure the validity of the data while entering data in a table. The constraints in MySQL are PRIMARY KEY, NOT NULL, FOREIGN KEY, UNIQUE, ENUM, and SET. We will discuss NOT NULL and PRIMARY KEY here.

NOT NULL: This constraint tells us that we cannot store a null value in a column. That is, if a column is specified as NOT NULL then we will not be able to store null in this particular column any more.

PRIMARY KEY: A primary key is a field which can uniquely identify each row in a table. And this constraint is used to specify a field in a table as the primary key.
For example, Create a table Student under the database School in MySQL for the specification shown in the table.

Column Name	Data Type	Size	Constraints
Roll No	INT		PRIMARY KEY
Name	VARCHAR	20	NOT NULL
Class	INT		
Section	CHAR	1	
Date of Admission	DATE		NOT NULL
Total Marks	INT		NOT NULL

Constraints are the rules that we can apply on the type of data in a table. That is, we can specify the limit on the type of data that can be stored in a particular column in a table using constraints.

Create Table student (RollNo INT PRIMARY KEY, Name VARCHAR (20) NOT NULL, Class INT, Section CHAR (1), DateOfAdmission DATE NOT NULL, TotalMarks INT NOT NULL);
Refer to Fig 3.9 to view the command in the database and its result.

```
mysql> Create Table student (
    -> RollNo INT PRIMARY KEY,
    -> Name VARCHAR(20) NOT NULL,
    -> Class INT,
    -> Section CHAR(1),
    -> DateOfAdmission DATE NOT NULL,
    -> TotalMarks INT NOT NULL
    -> );
Query OK, 0 rows affected (0.07 sec)
```

Fig 3.9 Creating a Table

3.3.4 Displaying Table Structure

To display complete information about the fields defined in the table Student, the query is (Fig.3.10):

DESCRIBE student;
OR
DESC student;

```
mysql> DESCRIBE student;
+-----------------+-------------+------+-----+---------+-------+
| Field           | Type        | Null | Key | Default | Extra |
+-----------------+-------------+------+-----+---------+-------+
| RollNo          | int(11)     | NO   | PRI | NULL    |       |
| Name            | varchar(20) | NO   |     | NULL    |       |
| Class           | int(11)     | YES  |     | NULL    |       |
| Section         | char(1)     | YES  |     | NULL    |       |
| DateOfAdmission | date        | NO   |     | NULL    |       |
| TotalMarks      | int(11)     | NO   |     | NULL    |       |
+-----------------+-------------+------+-----+---------+-------+
6 rows in set (0.02 sec)
```

Fig 3.10 Displaying Table Structure

3.3.5 Inserting Data in a Table

When a table is created, it is an empty structure. To add data to the table, the INSERT command is used.

If there are exactly the same number of values as there are fields and the values are sequenced in the same order, there is no need to specify column names. The table columns and their values have a one-to-one correspondence, i.e.,the first value is inserted into the first column, the second value into the second column, and so on. The query for inserting a row in the table is:

INSERT INTO student VALUES (1, 'Anita Saxena', 10, 'A', '2010-01-18', 450);

To view the table with the inserted record, the query is:

SELECT * FROM student;

Fig 3.11 shows both the above commands.

```
mysql> INSERT INTO student
    -> VALUES (1, 'Anita Saxena',10,'A', '2014-01-18', 450);
Query OK, 1 row affected (0.03 sec)

mysql> SELECT * FROM student;
+--------+--------------+-------+---------+-----------------+------------+
| RollNo | Name         | Class | Section | DateOfAdmission | TotalMarks |
+--------+--------------+-------+---------+-----------------+------------+
|      1 | Anita Saxena |    10 | A       | 2014-01-18      |        450 |
+--------+--------------+-------+---------+-----------------+------------+
1 row in set (0.01 sec)
```

Fig 3.11 Inserting & Display Value

You can also insert multiple rows into a table using the INSERT statement. For this, the query will be written as :

INSERT INTO student

VALUES (2, 'Zina Sarma', 10, 'B', '2014-01-18', 365) ,

(3, 'Vinay Singh', 10, 'C','2014-01-18', 420) ,

(4, 'Dina Lekharu', 10, 'A', '2014-01-18', 461) ,

(5, 'Chandan Mukerjee', 10, 'A', '2014-01-18', 390) ,

(6, 'Diya Jain', 10, 'B','2014-01-18', 490) ,

(7, 'Mayank Bansal', 10, 'C', '2014-01-18', 450) ,

(8, 'Fiona Faruq', 10, 'C', '2014-01-18', 380);

```
mysql> INSERT INTO student
    -> VALUES(2,'Zina Sarma',10,'B','2014-01-18',365),
    -> (3,'Vinay Singh',10,'C','2014-01-18',420),
    -> (4,'Dina Lekharu',10,'A','2014-01-18',461),
    -> (5,'Chandan Mukherjee', 10, 'A','2014-01-18',390),
    -> (6,'Diya Jain',10,'B','2014-01-18',490),
    -> (7,'Mayank Bansal',10,'C','2014-01-18',450),
    -> (8,'Fiona Faruq',10,'C','2014-01-18',380);
Query OK, 7 rows affected (0.02 sec)
Records: 7  Duplicates: 0  Warnings: 0
```

Fig 3.12 Inserting Multiple Values

To view the table with the multiple records, the query is:

```
mysql> SELECT * FROM student;
+--------+-------------------+-------+---------+-----------------+------------+
| RollNo | Name              | Class | Section | DateOfAdmission | TotalMarks |
+--------+-------------------+-------+---------+-----------------+------------+
|      1 | Anita Saxena      |    10 | A       | 2014-01-18      |        450 |
|      2 | Zina Sarma        |    10 | B       | 2014-01-18      |        365 |
|      3 | Vinay Singh       |    10 | C       | 2014-01-18      |        420 |
|      4 | Dina Lekharu      |    10 | A       | 2014-01-18      |        461 |
|      5 | Chandan Mukherjee |    10 | A       | 2014-01-18      |        390 |
|      6 | Diya Jain         |    10 | B       | 2014-01-18      |        490 |
|      7 | Mayank Bansal     |    10 | C       | 2014-01-18      |        450 |
|      8 | Fiona Faruq       |    10 | C       | 2014-01-18      |        380 |
+--------+-------------------+-------+---------+-----------------+------------+
8 rows in set (0.01 sec)
```

Fig 3.13 Viewing Multiple Records

3.4 CHANGING THE STRUCTURE OF A TABLE

The ALTER TABLE command is used to modify the structure of an existing table. With this command, you can:

- add or delete columns
- change the data type of existing columns
- rename columns or the table itself
- delete table
- add/delete PRIMARY KEY and NOT NULL constraints

3.4.1 Creating New Columns

The syntax for creating a new column in a table is;
ALTER TABLE <TableName> ADD (<NewColumnName> <DataType> (<size>), <NewColumnName> <DataType> (<size>), ...);

```
mysql> ALTER TABLE student ADD Grade CHAR(2);
Query OK, 0 rows affected (0.06 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESCRIBE student;
+-----------------+------------+------+-----+---------+-------+
| Field           | Type       | Null | Key | Default | Extra |
+-----------------+------------+------+-----+---------+-------+
| RollNo          | int(11)    | NO   | PRI | NULL    |       |
| Name            | varchar(20)| NO   |     | NULL    |       |
| Class           | int(11)    | YES  |     | NULL    |       |
| Section         | char(1)    | YES  |     | NULL    |       |
| DateOfAdmission | date       | NO   |     | NULL    |       |
| TotalMarks      | int(11)    | NO   |     | NULL    |       |
| Grade           | char(2)    | YES  |     | NULL    |       |
+-----------------+------------+------+-----+---------+-------+
7 rows in set (0.06 sec)
```

Fig 3.14 Adding New Column

3.4.2 Adding New Columns at a Specific Location

To add new columns to the table at specific locations in the table structure, the command syntax is:

ALTER TABLE <TableName> ADD <NewColumnName> ‹Datatype> (‹size>)
AFTER <OldColumnName

```
mysql> ALTER TABLE student ADD Percentage FLOAT(3,2) AFTER TotalMarks;
Query OK, 0 rows affected (0.05 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM student;
+--------+------------------+-------+---------+-----------------+------------+------------+-------+
| RollNo | Name             | Class | Section | DateOfAdmission | TotalMarks | Percentage | Grade |
+--------+------------------+-------+---------+-----------------+------------+------------+-------+
|      1 | Anita Saxena     |    10 | A       | 2014-01-18      |        450 |       NULL | NULL  |
|      2 | Zina Sarma       |    10 | B       | 2014-01-18      |        365 |       NULL | NULL  |
|      3 | Vinay Singh      |    10 | C       | 2014-01-18      |        420 |       NULL | NULL  |
|      4 | Dina Lekharu     |    10 | A       | 2014-01-18      |        461 |       NULL | NULL  |
|      5 | Chandan Mukherjee |    10 | A       | 2014-01-18      |        390 |       NULL | NULL  |
|      6 | Diya Jain        |    10 | B       | 2014-01-18      |        490 |       NULL | NULL  |
|      7 | Mayank Bansal    |    10 | C       | 2014-01-18      |        450 |       NULL | NULL  |
|      8 | Fiona Faruq      |    10 | C       | 2014-01-18      |        380 |       NULL | NULL  |
+--------+------------------+-------+---------+-----------------+------------+------------+-------+
8 rows in set (0.00 sec)
```

Fig 3.15 Adding Column in Specific Location

3.4.3 Modifying Existing Columns

You can change the data type of an existing column by using the ALTER TABLE command in the format shown below:
ALTER TABLE <TableName> MODIFY (<ColumnName> <NewDataType> (<NewSize>);

```
mysql> ALTER TABLE student MODIFY Name VARCHAR(30) ;
Query OK, 8 rows affected (0.08 sec)
Records: 8  Duplicates: 0  Warnings: 0

mysql> DESCRIBE student;
+-----------------+-------------+------+-----+---------+-------+
| Field           | Type        | Null | Key | Default | Extra |
+-----------------+-------------+------+-----+---------+-------+
| RollNo          | int(11)     | NO   | PRI | NULL    |       |
| Name            | varchar(30) | YES  |     | NULL    |       |
| Class           | int(11)     | YES  |     | NULL    |       |
| Section         | char(1)     | YES  |     | NULL    |       |
| DateOfAdmission | date        | NO   |     | NULL    |       |
| TotalMarks      | int(11)     | NO   |     | NULL    |       |
| Percentage      | float(3,2)  | YES  |     | NULL    |       |
| Grade           | char(2)     | YES  |     | NULL    |       |
+-----------------+-------------+------+-----+---------+-------+
8 rows in set (0.01 sec)
```

Fig 3.16 Modified Column Name

You can also change the data type of an existing column and shift it to a specific location by using the ALTER TABLE command as given below:

ALTER TABLE ‹TableName> MODIFY (<ColumnName> <NewDataType>(<NewSize>) AFTER <ColumnName>);

```
mysql> ALTER TABLE student MODIFY Grade CHAR(2) AFTER TotalMarks;
Query OK, 0 rows affected (0.05 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESCRIBE student;
+-----------------+------------+------+-----+---------+-------+
| Field           | Type       | Null | Key | Default | Extra |
+-----------------+------------+------+-----+---------+-------+
| RollNo          | int(11)    | NO   | PRI | NULL    |       |
| Name            | varchar(30)| YES  |     | NULL    |       |
| Class           | int(11)    | YES  |     | NULL    |       |
| Section         | char(1)    | YES  |     | NULL    |       |
| DateOfAdmission | date       | NO   |     | NULL    |       |
| TotalMarks      | int(11)    | NO   |     | NULL    |       |
| Grade           | char(2)    | YES  |     | NULL    |       |
| Percentage      | float(3,2) | YES  |     | NULL    |       |
+-----------------+------------+------+-----+---------+-------+
8 rows in set (0.01 sec)
```

Fig. 3.17 Modified Grade After TotalMarks

Or

ALTER TABLE <TableName> MODIFY (<ColumnName> <NewDataType> (<NewSize>) FIRST);

```
mysql> ALTER TABLE student MODIFY Percentage FLOAT(3,2) FIRST;
Query OK, 0 rows affected (0.04 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESCRIBE student;
+-----------------+------------+------+-----+---------+-------+
| Field           | Type       | Null | Key | Default | Extra |
+-----------------+------------+------+-----+---------+-------+
| Percentage      | float(3,2) | YES  |     | NULL    |       |
| RollNo          | int(11)    | NO   | PRI | NULL    |       |
| Name            | varchar(30)| YES  |     | NULL    |       |
| Class           | int(11)    | YES  |     | NULL    |       |
| Section         | char(1)    | YES  |     | NULL    |       |
| DateOfAdmission | date       | NO   |     | NULL    |       |
| TotalMarks      | int(11)    | NO   |     | NULL    |       |
| Grade           | char(2)    | YES  |     | NULL    |       |
+-----------------+------------+------+-----+---------+-------+
8 rows in set (0.01 sec)
```

Fig. 3.18 Modified Percentage as First Field

3.4.4 Renaming Existing Columns

You can rename an existing column by using the following form of the ALTER TABLE command:

ALTER TABLE <TableName> CHANGE <OldColumnName> <NewColumnName> <Datatype> (<size>) ;

```
mysql> ALTER TABLE student CHANGE Name StudentName VARCHAR(20);
Query OK, 8 rows affected (0.05 sec)
Records: 8  Duplicates: 0  Warnings: 0

mysql> DESCRIBE student;
+-----------------+-------------+------+-----+---------+-------+
| Field           | Type        | Null | Key | Default | Extra |
+-----------------+-------------+------+-----+---------+-------+
| Percentage      | float(3,2)  | YES  |     | NULL    |       |
| RollNo          | int(11)     | NO   | PRI | NULL    |       |
| StudentName     | varchar(20) | YES  |     | NULL    |       |
| Class           | int(11)     | YES  |     | NULL    |       |
| Section         | char(1)     | YES  |     | NULL    |       |
| DateOfAdmission | date        | NO   |     | NULL    |       |
| TotalMarks      | int(11)     | NO   |     | NULL    |       |
| Grade           | char(2)     | YES  |     | NULL    |       |
+-----------------+-------------+------+-----+---------+-------+
8 rows in set (0.01 sec)
```

Fig. 3.19 Changed Column Name

You can also rename the existing column and shift it to a specific location within the table.

ALTER TABLE <TableName> CHANGE <OldColumnName> <NewColumnName> <Datatype> (<size>) AFTER <ColumnName>;

```
mysql> ALTER TABLE student CHANGE Percentage Percent FLOAT(3,2) AFTER Grade;
Query OK, 0 rows affected (0.05 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESCRIBE student;
+-----------------+-------------+------+-----+---------+-------+
| Field           | Type        | Null | Key | Default | Extra |
+-----------------+-------------+------+-----+---------+-------+
| RollNo          | int(11)     | NO   | PRI | NULL    |       |
| StudentName     | varchar(20) | YES  |     | NULL    |       |
| Class           | int(11)     | YES  |     | NULL    |       |
| Section         | char(1)     | YES  |     | NULL    |       |
| DateOfAdmission | date        | NO   |     | NULL    |       |
| TotalMarks      | int(11)     | NO   |     | NULL    |       |
| Grade           | char(2)     | YES  |     | NULL    |       |
| Percent         | float(3,2)  | YES  |     | NULL    |       |
+-----------------+-------------+------+-----+---------+-------+
8 rows in set (0.01 sec)
```

Fig. 3.20 Changed Column Name Shift Position

Or

ALTER TABLE <TableName> CHANGE <OldColumnName> <NewColumnName><Datatype> (<size>)
FIRST;

3.4.5 Dropping a Column

This means to delete a column.

The syntax of this command is:

ALTER TABLE <TableName> DROP <ColumnName>;

```
mysql> ALTER TABLE student DROP Percent;
Query OK, 0 rows affected (0.04 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESCRIBE student;
+-----------------+-------------+------+-----+---------+-------+
| Field           | Type        | Null | Key | Default | Extra |
+-----------------+-------------+------+-----+---------+-------+
| RollNo          | int(11)     | NO   | PRI | NULL    |       |
| StudentName     | varchar(20) | YES  |     | NULL    |       |
| Class           | int(11)     | YES  |     | NULL    |       |
| Section         | char(1)     | YES  |     | NULL    |       |
| DateOfAdmission | date        | NO   |     | NULL    |       |
| TotalMarks      | int(11)     | NO   |     | NULL    |       |
| Grade           | char(2)     | YES  |     | NULL    |       |
+-----------------+-------------+------+-----+---------+-------+
7 rows in set (0.01 sec)
```

Fig. 3.21 Dropping Column Percent

Or

ALTER TABLE <TableName> DROP <ColumnName1>, <ColumnName2>, - - -;

3.4.6 Renaming a Table

To rename a table, the syntax is:

ALTER TABLE <OldTableName> RENAME [TO] <NewTableName>;

```
mysql> ALTER TABLE student RENAME studentdata;
Query OK, 0 rows affected (0.02 sec)

mysql> SHOW TABLES;
+------------------+
| Tables_in_school |
+------------------+
| studentdata      |
+------------------+
1 row in set (0.01 sec)
```

Fig.3.22 Renaming Table

3.4.7 Destroying Tables

Sometimes, we are required to permanently delete a table along with its structure. The syntax for this command is:
DROP TABLE (IF EXISTS] <TableName>;

The keyword IF EXISTS can be used to prevent the occurrence of an error while deleting a table that does not exist.
The query to destroy the table 'student' along with its data is:
DROP TABLE student;

3.4.8 Add / Delete Constraint

Suppose you have created the table 'student' without specifying any primary key, in such a case, the following query:
ALTER TABLE student ADD PRIMARY KEY (RolINo);
will set the RollNo field as the primary key of the table.

To delete the primary key constraint, the query is:
ALTER TABLE <TableName> DROP PRIMARY KEY;

3.5 SELECT STATEMENT

The SELECT statement is used to retrieve zero or more rows from a table. Its syntax is:

SELECT col1, col2, ... FROM <TableName>
WHERE conditions
GROUP BY group
HAVING group conditions
ORDER BY col1 [ASC I DESC];

There are three ways to retrieve data from a table:

- Selected columns and all rows
- Selected rows and all the columns
- Selected columns and selected rows

3.5.1 Selected Columns and all Rows

The retrieval of specific columns from a table can be done as shown below:
SELECT <ColumnName1>, <ColumnName2> FROM <TableName>;

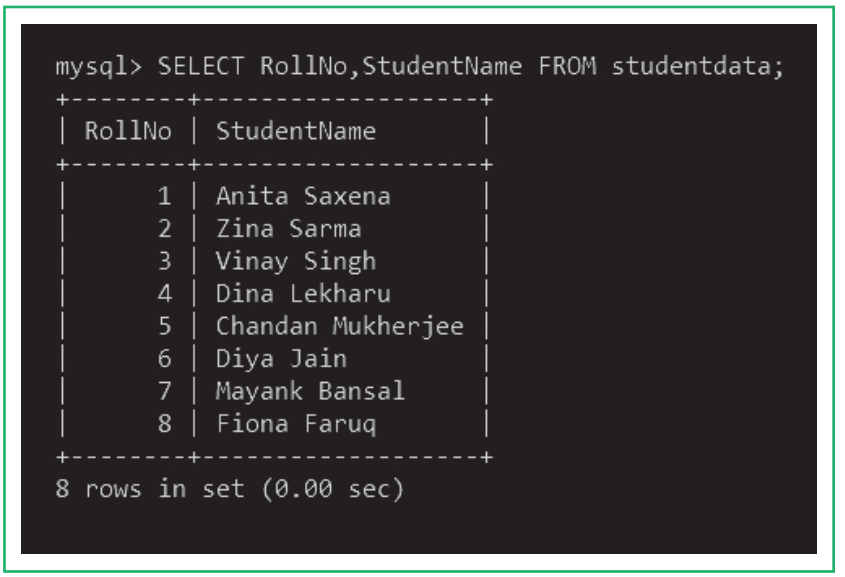


Fig. 3.23 Data From Specified Columns

3.5.2 Selected Rows and all the Columns

Suppose that the information for a particular student has to be retrieved from a table. This retrieval would be based on specific conditions. For this, WHERE clause is used in the SELECT statement.

The syntax for such a query is:

SELECT * FROM <TableName> WHERE <Condition>;

```
mysql> SELECT * FROM studentdata WHERE Section ='A';
+--------+------------------+-------+---------+-----------------+------------+-------+
| RollNo | StudentName      | Class | Section | DateOfAdmission | TotalMarks | Grade |
+--------+------------------+-------+---------+-----------------+------------+-------+
|      1 | Anita Saxena     |    10 | A       | 2014-01-18      |        450 | NULL  |
|      4 | Dina Lekharu     |    10 | A       | 2014-01-18      |        461 | NULL  |
|      5 | Chandan Mukherjee |    10 | A       | 2014-01-18      |        390 | NULL  |
+--------+------------------+-------+---------+-----------------+------------+-------+
3 rows in set (0.01 sec)
```

Fig.3.24 Data of all columns and specific rows

3.5.3 Selected Columns and Selected Rows

To display the selected set of columns of selected rows from a table, the query syntax is;

SELECT <ColumnName1>, <ColumnName2> FROM <TableName> WHERE Condition>;

```
mysql> SELECT StudentName, Class FROM studentdata WHERE Section ='B';
+-------------+-------+
| StudentName | Class |
+-------------+-------+
| Zina Sarma  |    10 |
| Diya Jain   |    10 |
+-------------+-------+
2 rows in set (0.00 sec)
```

Fig.3.25 Data from Selected columns and rows

3.5.4 Eliminating Duplicate Values

To display only unique values, the DISTINCT clause can be used. It can only be used in a SELECT statement. The DISTINCT clause scans through the values of the specified table columns and displays only the unique values among them. The syntax is:

SELECT DISTINCT <ColumnName> FROM <TableName>;

```
mysql> ALTER TABLE student RENAME studentdata;
Query OK, 0 rows affected (0.02 sec)

mysql> SHOW TABLES;
+------------------+
| Tables_in_school |
+------------------+
| studentdata      |
+------------------+
1 row in set (0.01 sec)
```

Fig.3.26 Selecting Distinct Column

3.5.5 Retrieving All Data Values

The keyword ALL in place of DISTINCT will display the data from all the rows.

3.5.6 Pattern Matching

The LIKE predicate allows the comparison of one string value with another string value. It uses two wildcard characters:

% (percentage sign) It allows finding a match for any string of any length, including zero length.

_ (underscore sign) It allows finding a match for any single character.

The query:

SELECT * FROM <TableName> WHERE <ColumnName> LIKE 'A%';
displays all the rows having Column name starting with 'A'.

```
mysql> SELECT * FROM studentdata WHERE StudentName LIKE 'D%';
+--------+-------------+-------+---------+-----------------+------------+-------+
| RollNo | StudentName | Class | Section | DateOfAdmission | TotalMarks | Grade |
+--------+-------------+-------+---------+-----------------+------------+-------+
|      4 | Dina Lekharu |    10 | A       | 2014-01-18      |        461 | NULL  |
|      6 | Diya Jain    |    10 | B       | 2014-01-18      |        490 | NULL  |
+--------+-------------+-------+---------+-----------------+------------+-------+
2 rows in set (0.00 sec)
```

Fig.3.27 Like Clause with 'D%'

SELECT * FROM <TableName> WHERE <ColumnName> LIKE '%a';

It displays all the records having names ending in 'a'.

```
mysql> SELECT * FROM studentdata WHERE StudentName LIKE '%a';
+--------+-------------+-------+---------+-----------------+------------+-------+
| RollNo | StudentName | Class | Section | DateOfAdmission | TotalMarks | Grade |
+--------+-------------+-------+---------+-----------------+------------+-------+
|      1 | Anita Saxena |    10 | A       | 2014-01-18      |        450 | NULL  |
|      2 | Zina Sarma   |    10 | B       | 2014-01-18      |        365 | NULL  |
+--------+-------------+-------+---------+-----------------+------------+-------+
2 rows in set (0.00 sec)
```

Fig. 3.28 Like Clause with '%a'

SELECT * FROM <TableName> WHERE <ColumnName> LIKE 'M_____________' ;

This displays all the rows having names starting with 'M' followed by any twelve characters.

```
mysql> SELECT * FROM studentdata WHERE StudentName LIKE 'M_____________';
+--------+---------------+-------+---------+-----------------+------------+-------+
| RollNo | StudentName   | Class | Section | DateOfAdmission | TotalMarks | Grade |
+--------+---------------+-------+---------+-----------------+------------+-------+
|      7 | Mayank Bansal |    10 | C       | 2014-01-18      |        450 | NULL  |
+--------+---------------+-------+---------+-----------------+------------+-------+
1 row in set (0.00 sec)
```

Fig. 3.29 Like Clause with 'M_____________'

3.5.7 Range Searching

The BETWEEN clause is used to search the data within the range (specified lower & upper limit). It includes both the lower and upper limit in the range.
The query,
SELECT * FROM studentdata WHERE TotalMarks BETWEEN 350 AND 450;

Displays all the records having TotalMarks in the range 350 to 450 (inclusive of lower and upper limits).

```
mysql> SELECT * FROM studentdata WHERE  TotalMarks BETWEEN 350 AND 450;
+--------+------------------+-------+---------+-----------------+------------+-------+
| RollNo | StudentName      | Class | Section | DateOfAdmission | TotalMarks | Grade |
+--------+------------------+-------+---------+-----------------+------------+-------+
|      1 | Anita Saxena     |    10 | A       | 2014-03-15      |        450 | NULL  |
|      2 | Zina Sarma       |    10 | B       | 2014-03-15      |        400 | NULL  |
|      3 | Vinay Singh      |    10 | C       | 2014-03-15      |        420 | NULL  |
|      5 | Chandan Mukherjee|    10 | A       | 2014-03-15      |        390 | NULL  |
|      7 | Mayank Bansal    |    10 | C       | 2014-03-15      |        450 | NULL  |
|      8 | Fiona Faruq      |    10 | C       | 2014-03-15      |        380 | NULL  |
+--------+------------------+-------+---------+-----------------+------------+-------+
6 rows in set (0.00 sec)
```

Fig. 3.30 Range using BETWEEN clause

3.6 WORKING WITH OPERATORS

MySOL supports a number of operators that can be used for performing calculations on the table values.
Let us discuss these operators and their functions.

3.6.1 RaArithmetic Operators

Operator	Meaning
+	Addition
-	Subtraction
/	Division
*	Multiplication
%	Modulus or remainder

You can use arithmetic operators while viewing records in a table (in WHERE clause) as well as performing data manipulation operations such as INSERT, UPDATE, and DELETE. The arithmetic operators are given in the above table.

For example, in the query:

SELECT RollNo, StudentName, TotalMarks, TotalMarks/500*100 FROM studentdata;

The calculation is done in the TotalMarks field and the result is displayed. The calculated column does not have any column name. By default, the entire expression is used as the column name while displaying the output.

```
mysql> SELECT RollNo, StudentName, TotalMarks, TotalMarks/500*100 FROM studentdata;
+--------+------------------+------------+--------------------+
| RollNo | StudentName      | TotalMarks | TotalMarks/500*100 |
+--------+------------------+------------+--------------------+
|      1 | Anita Saxena     |        450 |            90.0000 |
|      2 | Zina Sarma       |        365 |            73.0000 |
|      3 | Vinay Singh      |        420 |            84.0000 |
|      4 | Dina Lekharu     |        461 |            92.2000 |
|      5 | Chandan Mukherjee|        390 |            78.0000 |
|      6 | Diya Jain        |        490 |            98.0000 |
|      7 | Mayank Bansal    |        450 |            90.0000 |
|      8 | Fiona Faruq      |        380 |            76.0000 |
+--------+------------------+------------+--------------------+
8 rows in set (0.01 sec)
```

Fig.3.31 Calculation using arithmetic operators

3.6.2 Relational operators

Operator	Meaning
=	Equal to
<> or !=	Not equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to

For example;

SELECT * FROM <TableName> WHERE TotalMarks <=400;

This query will display all records where the TotalMarks is less than or equal to 400.

```
mysql> SELECT * FROM studentdata WHERE TotalMarks <=400;
+--------+------------------+-------+---------+-----------------+------------+-------+
| RollNo | StudentName      | Class | Section | DateOfAdmission | TotalMarks | Grade |
+--------+------------------+-------+---------+-----------------+------------+-------+
|      2 | Zina Sarma       |    10 | B       | 2014-01-18      |        365 | NULL  |
|      5 | Chandan Mukherjee |    10 | A       | 2014-01-18      |        390 | NULL  |
|      8 | Fiona Faruq      |    10 | C       | 2014-01-18      |        380 | NULL  |
+--------+------------------+-------+---------+-----------------+------------+-------+
3 rows in set (0.00 sec)
```

Fig.3.32 Using Relational Operator

3.6.1 Logical operators

The three logical operators are AND, OR, and NOT.

AND operator It requires that all the conditions be true for inclusion in the result. The query:

SELECT * FROM studentdata WHERE TotalMarks >= 200 AND TotalMarks <= 400;

will display all the records that satisfy both the conditions given in the WHERE clause and will be
included in the output.

```
mysql> SELECT * FROM studentdata  WHERE TotalMarks >= 200 AND TotalMarks <= 400;
+--------+------------------+-------+---------+-----------------+------------+-------+
| RollNo | StudentName      | Class | Section | DateOfAdmission | TotalMarks | Grade |
+--------+------------------+-------+---------+-----------------+------------+-------+
|      2 | Zina Sarma       |    10 | B       | 2014-01-18      |        365 | NULL  |
|      5 | Chandan Mukherjee |    10 | A       | 2014-01-18      |        390 | NULL  |
|      8 | Fiona Faruq      |    10 | C       | 2014-01-18      |        380 | NULL  |
+--------+------------------+-------+---------+-----------------+------------+-------+
3 rows in set (0.01 sec)
```

Fig. 3.33 Query using AND operator

OR operator It requires any one of the conditions to be true for inclusion in the result. The query:

SELECT * FROM studentdata WHERE Section = 'A' OR Section = 'B';

will display the records that satisfy any one of the conditions in the WHERE clause.

```
mysql> UPDATE studentdata SET DateOfAdmission= '2014-3-15';
Query OK, 8 rows affected (0.03 sec)
Rows matched: 8  Changed: 8  Warnings: 0

mysql> SELECT * FROM studentdata;
+--------+------------------+-------+---------+-----------------+-----------+-------+
| RollNo | StudentName      | Class | Section | DateOfAdmission | TotalMarks | Grade |
+--------+------------------+-------+---------+-----------------+-----------+-------+
|      1 | Anita Saxena     |    10 | A       | 2014-03-15      |        450 | NULL  |
|      2 | Zina Sarma       |    10 | B       | 2014-03-15      |        365 | NULL  |
|      3 | Vinay Singh      |    10 | C       | 2014-03-15      |        420 | NULL  |
|      4 | Dina Lekharu     |    10 | A       | 2014-03-15      |        461 | NULL  |
|      5 | Chandan Mukherjee|    10 | A       | 2014-03-15      |        390 | NULL  |
|      6 | Diya Jain        |    10 | B       | 2014-03-15      |        490 | NULL  |
|      7 | Mayank Bansal    |    10 | C       | 2014-03-15      |        450 | NULL  |
|      8 | Fiona Faruq      |    10 | C       | 2014-03-15      |        380 | NULL  |
+--------+------------------+-------+---------+-----------------+-----------+-------+
8 rows in set (0.00 sec)
```

Fig. 3.34 Query using OR operator

NOT operator It displays only those records that do not satisfy the specified condition. The query:

SELECT * FROM student WHERE NOT (Section ='A');

will display all the records that do not satisfy the condition.

```
mysql> SELECT * FROM studentdata WHERE NOT (Section = 'A');
+--------+--------------+-------+---------+-----------------+-----------+-------+
| RollNo | StudentName  | Class | Section | DateOfAdmission | TotalMarks | Grade |
+--------+--------------+-------+---------+-----------------+-----------+-------+
|      2 | Zina Sarma   |    10 | B       | 2014-01-18      |        365 | NULL  |
|      3 | Vinay Singh  |    10 | C       | 2014-01-18      |        420 | NULL  |
|      6 | Diya Jain    |    10 | B       | 2014-01-18      |        490 | NULL  |
|      7 | Mayank Bansal|    10 | C       | 2014-01-18      |        450 | NULL  |
|      8 | Fiona Faruq  |    10 | C       | 2014-01-18      |        380 | NULL  |
+--------+--------------+-------+---------+-----------------+-----------+-------+
5 rows in set (0.00 sec)
```

Fig. 3.35 Query using NOT operator

3.7 SORTING DATA IN A TABLE

Sorting is used for arranging data in large databases. Sorting can be done in MySQL using the ORDER BY clause. The data in the table can be sorted either in ascending or in descending order. The syntax is:

SELECT * FROM <TableName> ORDER BY ColumnName1 <Sort order>], ColumnName2 [<Sort order>];

The ORDER BY clause is used only with the SELECT statement. The sort order is DESC for descending order and ASC for ascending order. If no sort order is mentioned, sorting is done in ascending order. The query:

SELECT * FROM studentdata ORDER BY StudentName;

will sort the table in ascending order of the Student Name field.

```
mysql> SELECT * FROM studentdata  ORDER BY StudentName;
+--------+------------------+-------+---------+-----------------+------------+-------+
| RollNo | StudentName      | Class | Section | DateOfAdmission | TotalMarks | Grade |
+--------+------------------+-------+---------+-----------------+------------+-------+
|      1 | Anita Saxena     |    10 | A       | 2014-01-18      |        450 | NULL  |
|      5 | Chandan Mukherjee|    10 | A       | 2014-01-18      |        390 | NULL  |
|      4 | Dina Lekharu     |    10 | A       | 2014-01-18      |        461 | NULL  |
|      6 | Diya Jain        |    10 | B       | 2014-01-18      |        490 | NULL  |
|      8 | Fiona Faruq      |    10 | C       | 2014-01-18      |        380 | NULL  |
|      7 | Mayank Bansal    |    10 | C       | 2014-01-18      |        450 | NULL  |
|      3 | Vinay Singh      |    10 | C       | 2014-01-18      |        420 | NULL  |
|      2 | Zina Sarma       |    10 | B       | 2014-01-18      |        365 | NULL  |
+--------+------------------+-------+---------+-----------------+------------+-------+
8 rows in set (0.00 sec)
```

Fig. 3.36 Ascending order of StudentName

The query:

SELECT * FROM studentdata ORDER BY TotalMarks DESC;

will sort the table in descending order of TotalMarks.

```
mysql> SELECT * FROM studentdata ORDER BY TotalMarks DESC;
+--------+------------------+-------+---------+-----------------+------------+-------+
| RollNo | StudentName      | Class | Section | DateOfAdmission | TotalMarks | Grade |
+--------+------------------+-------+---------+-----------------+------------+-------+
|      6 | Diya Jain        |    10 | B       | 2014-01-18      |        490 | NULL  |
|      4 | Dina Lekharu     |    10 | A       | 2014-01-18      |        461 | NULL  |
|      1 | Anita Saxena     |    10 | A       | 2014-01-18      |        450 | NULL  |
|      7 | Mayank Bansal    |    10 | C       | 2014-01-18      |        450 | NULL  |
|      3 | Vinay Singh      |    10 | C       | 2014-01-18      |        420 | NULL  |
|      5 | Chandan Mukherjee|    10 | A       | 2014-01-18      |        390 | NULL  |
|      8 | Fiona Faruq      |    10 | C       | 2014-01-18      |        380 | NULL  |
|      2 | Zina Sarma       |    10 | B       | 2014-01-18      |        365 | NULL  |
+--------+------------------+-------+---------+-----------------+------------+-------+
8 rows in set (0.01 sec)
```

Fig. 3.37 Descending order of TotalMarks

3.7.1 Deleting Data From Tables

The DELETE command deletes rows that satisfy the condition specified in the WHERE clause and returns the number of records deleted. If you do not have a WHERE clause in the query, all the rows will be deleted. The general format for this query is:

DELETE FROM <TableName> WHERE <Condition>;

Delete All Rows

The query:
DELETE FROM studentdata;
will delete all the rows from the table studentdata.

Delete Specific Row(s)

The query:
DELETE FROM studentdata WHERE Section ='A';
will delete all the rows where Section is 'A'.

3.7.2 Updating Data in Tables

The UPDATE command is used to change or modify data values in a table. This is different from the MODIFY clause in the ALTER TABLE command, which is used to change the data type and constraint (PRIMARY KEY, NOT NULL) of existing columns in a table. The UPDATE statement updates all the specified columns, in every row that exists in the table. The SET clause indicates the column data that should be modified and the new values they should take. The WHERE clause, if given, defines the specified rows that should be updated. If you do not have a WHERE clause, all table rows are updated.

The general format is:
UPDATE <TableName> SET <ColumnName1>=< Expression1›, <ColumnName2>=< Expression2> WHERE <Condition1>;

The query:
UPDATE studentdata SET DateOfAdmission='2014-03-15";
changes the DateOfAdmission of all the students to 15th March 2014.

```
mysql> UPDATE studentdata SET DateOfAdmission= '2014-3-15';
Query OK, 8 rows affected (0.03 sec)
Rows matched: 8  Changed: 8  Warnings: 0

mysql> SELECT * FROM studentdata;
+--------+------------------+-------+---------+-----------------+------------+-------+
| RollNo | StudentName      | Class | Section | DateOfAdmission | TotalMarks | Grade |
+--------+------------------+-------+---------+-----------------+------------+-------+
|      1 | Anita Saxena     |    10 | A       | 2014-03-15      |        450 | NULL  |
|      2 | Zina Sarma       |    10 | B       | 2014-03-15      |        365 | NULL  |
|      3 | Vinay Singh      |    10 | C       | 2014-03-15      |        420 | NULL  |
|      4 | Dina Lekharu     |    10 | A       | 2014-03-15      |        461 | NULL  |
|      5 | Chandan Mukherjee|    10 | A       | 2014-03-15      |        390 | NULL  |
|      6 | Diya Jain        |    10 | B       | 2014-03-15      |        490 | NULL  |
|      7 | Mayank Bansal    |    10 | C       | 2014-03-15      |        450 | NULL  |
|      8 | Fiona Faruq      |    10 | C       | 2014-03-15      |        380 | NULL  |
+--------+------------------+-------+---------+-----------------+------------+-------+
8 rows in set (0.00 sec)
```

Fig. 3.38 Updating DateOfAdmission

The following queries:

UPDATE studentdata SET TotalMarks = 400 WHERE RollNo= 2;

will only update the TotalMarks of rows with RollNo= 2.

```
mysql> UPDATE studentdata SET TotalMarks = 400  WHERE RollNo = 2;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM studentdata;
+--------+------------------+-------+---------+-----------------+------------+-------+
| RollNo | StudentName      | Class | Section | DateOfAdmission | TotalMarks | Grade |
+--------+------------------+-------+---------+-----------------+------------+-------+
|      1 | Anita Saxena     |    10 | A       | 2014-03-15      |        450 | NULL  |
|      2 | Zina Sarma       |    10 | B       | 2014-03-15      |        400 | NULL  |
|      3 | Vinay Singh      |    10 | C       | 2014-03-15      |        420 | NULL  |
|      4 | Dina Lekharu     |    10 | A       | 2014-03-15      |        461 | NULL  |
|      5 | Chandan Mukherjee|    10 | A       | 2014-03-15      |        390 | NULL  |
|      6 | Diya Jain        |    10 | B       | 2014-03-15      |        490 | NULL  |
|      7 | Mayank Bansal    |    10 | C       | 2014-03-15      |        450 | NULL  |
|      8 | Fiona Faruq      |    10 | C       | 2014-03-15      |        380 | NULL  |
+--------+------------------+-------+---------+-----------------+------------+-------+
8 rows in set (0.00 sec)
```

Fig. 3.39 Updating Specific Data

3.7.3 Aggregate or Group Functions

In MySQL, aggregate functions) are applied on a group of values as input and return a single value as the result.

Aggregate or group functions

Functions	Meaning
SUM()	Returns the sum of values of specified columns/expressions.
MAX()	Returns the maximum value of a set of values of specified columns/expressions
MIN()	AVG() Returns the average of values of specified columns/expressions.
AVG()	Returns the average of values of specified columns/expressions.
COUNT()	Returns the number of values in specified columns/expressions.
COUNT(*)	Returns the number of rows in the table.

The following queries will explain the working of aggregate functions.

SELECT SUM(TotalMarks), AVG(TotalMarks), MAX(TotalMarks), MIN(TotalMarks) FROM studentdata;

The result shows the sum, average, maximum, and minimum values of the EmpSalary column.

```
mysql> SELECT SUM(TotalMarks) , AVG(TotalMarks) , MAX(TotalMarks), MIN(TotalMarks) FROM studentdata;
+----------------+----------------+----------------+----------------+
| SUM(TotalMarks) | AVG(TotalMarks) | MAX(TotalMarks) | MIN(TotalMarks) |
+----------------+----------------+----------------+----------------+
|           3441 |       430.1250 |            490 |            380 |
+----------------+----------------+----------------+----------------+
1 row in set (0.01 sec)
```

Fig. 3.40 Using Aggregate or Group Functions

SELECT COUNT(*) FROM studentdata WHERE Section = 'C';

It gives a count of records with the Section as 'C'.

```
mysql> SELECT COUNT(*) FROM studentdata WHERE Section = 'C';
+----------+
| COUNT(*) |
+----------+
|        3 |
+----------+
1 row in set (0.00 sec)
```

Fig. 3.41 Counting rows

SELECT COUNT(RollNo) FROM studentdata;

It returns the number of values in the RollNo column.

```
mysql> SELECT COUNT(RollNo) FROM studentdata;
+---------------+
| COUNT(RollNo) |
+---------------+
|             8 |
+---------------+
1 row in set (0.00 sec)
```

Fig. 3.42 Counting values

KEYWORDS LEARNED IN THIS CHAPTER

CREATE SHOW USE INSERT SELECT ALTER
WHERE MODIFY RENAME SORT OPERATORS
CONSTRAINTS PRIMARY NULL NOT NULL

Exercise

I. MULTIPLE CHOICE QUESTIONS:

1. The command used to modify the content of a table is:
   a) ALTER TABLE
   b) SELECT
   c) UPDATE

2. The command used to display the table structure is:
   a) DISPLAY
   b) STRUCTURE
   c) DESCRIBE

3. A table name should begin with:
   a) Number
   b) Alphabet
   c) Symbol

4. The command used to delete the database physically:
   a) DELETE
   b) ERASE
   c) DROP

5. This wildcard character allows finding a match for any string of any length, including zero length:
   a) *
   b) %
   c) #

6. This operator displays only those records that do not satisfy the specified condition,
   a) AND b) OR c) NOT

II. FILL IN THE BLANKS:

1. MySQL is named after co-founder Michael Widenius's daughter, _____.
2. The number of rows denotes the _____ of the table.
3. The number of _____ denotes the Degree of the table.
4. _____ words are not allowed in a table name.
5. A MySQL statement is terminated by a _____.
6. The underscore wildcard allows finding a match for any _____ character.

III. ANSWER THE FOLLOWING QUESTIONS:

1. Who were the developers of MySQL?
2. Why is MySQL becoming so popular? Give two reasons.
3. What is a constraint? Name any two constraints.
4. Give examples of DML commands?
5. What are the characteristics by which you can determine the data type of MySQL?
6. What is the query to display the table structure?
7. What is the query to display all the records in a table?
8. List the Arithmetic Operators used in MySQL.
9. List the Relational Operators used in MySQL.
10. Differentiate between COUNT(*) and COUNT.
11. What are the rules for naming a table in MySQL?
12. Explain the five categories of SQL commands?

IV. PRACTICAL EXERCISE

1. Create a Table 'customer' in a database named 'Klubmart' with the following fields:

Column Name	Data Type	Size	Constraints
CustomerID	INT		PRIMARY KEY
Customer Name	VARCHAR	20	NOT NULL
CustomerMobile	VARCHAR	11	NOT NULL
CustomerCategory	CHAR	1	It is either X, Y or Z

2. Display the structure of the table.

3. Insert five records in the table.

4. Display all the records.

5. Display the CustomerName & CustomerMobile.

6. Display the CustomerName whose CustomerCategory is 'X'.

7. Display the CustomerID whose CustomerCategory is 'Y' OR 'Z'.

8. Display the distinct CustomerCategory from the table.

V. SOLUTION

1. The MySQL command to create the database is:

```
CREATE DATABASE Klubmart;
```

To use the Database Klubmart, type:

```
USE Klubmart;
```

To create the table 'customer' in the database 'Klubmart':

```
CREATE TABLE customer
(CustomerID INT PRIMARY KEY,
CustomerName VARCHAR (20) NOT NULL,
CustomerMobile VARCHAR (10) NOT NULL,
CustomerCategory CHAR);
```

2. To display the structure of the table type the command:

```
DESCRIBE customer;
```

3. To insert the record type the following command:

```
INSERT INTO customer values (1, 'Atul','9435110011', 'X');
INSERT INTO customer values (2, 'Ashwini','9864078011', 'Y');
INSERT INTO customer values (3, 'Anjana','7086219344', 'Z');
INSERT INTO customer values (4, 'Archana','9435987611', 'Y');
INSERT INTO customer values (5, 'Amresh','9435110011', 'X');
```

4. SELECT * FROM customer;

5. SELECT CustomerName, CustomerMobile FROM customer;

6. SELECT CustomerName FROM customer WHERE CustomerCategory = 'X';

7. SELECT CustomerID FROM customer WHERE CustomerCategory = 'Y' OR CustomerCategory = 'Z';

8. SELECT DISTINCT(CustomerCategory) FROM customer;

2. Create a table Worker in a database named Organisation. Enter the records given in the table below.

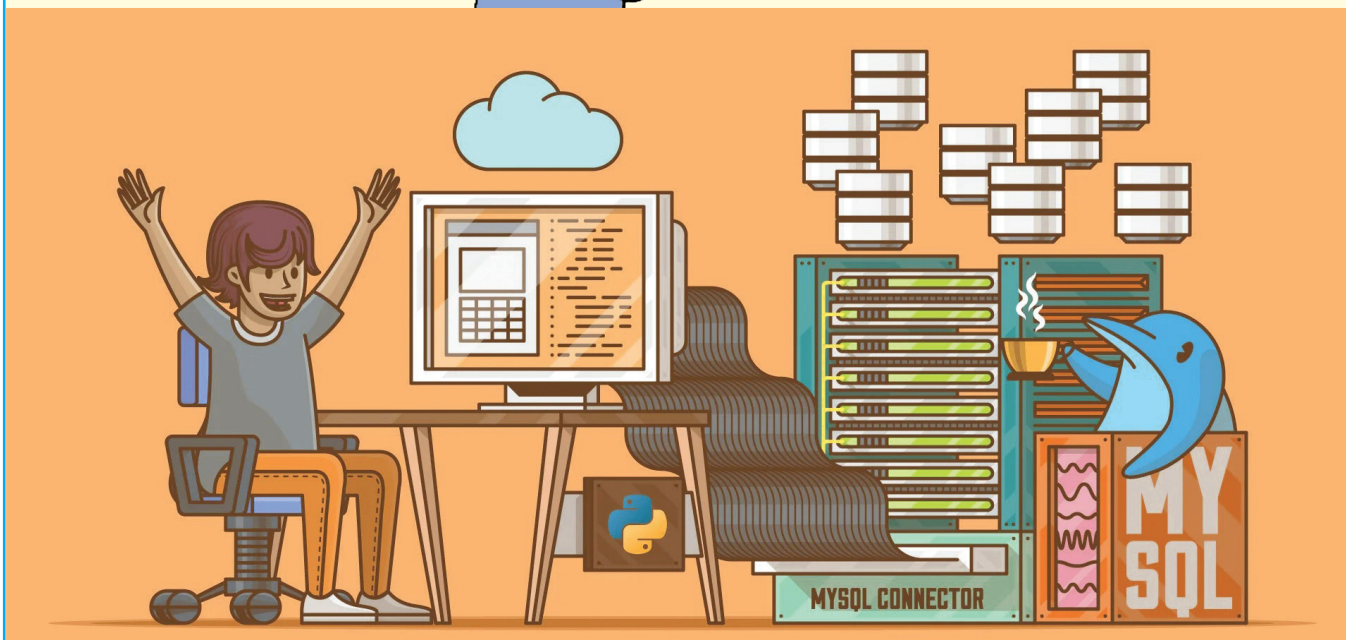
Worker_ID	First_Name	Last_Name	Salary	Joining_Date	Department
001	Rebecca	Doungel	40000	2014-02-20	HR
002	Jahnavi	Borthakur	68000	2014-06-11	Admin
003	Digvijay	Goswami	50000	2014-02-20	HR
004	Krishang	Shandilya	50000	2014-02-20	Admin
005	Jaspal	Bhatti	55000	2014-06-11	Admin
006	Kankana	Devi	20000	2014-06-11	Account
007	Shristi	Goswami	75000	2014-01-20	Account
008	Geeta	Sharma	90000	2014-04-11	Admin

Write the commands to do the following:

1. Display Worker_ID and Joining_Date of Admin Department.
2. Display records in ascending order of Salary.
3. Display records having Salary in the range 50000 to 90000.
4. Display the record of persons who First_Name ends with 'a'.
5. Display the records of HR and Account Department.
6. The worker is given bonus as 1% of the Salary. Display the First_Name, Last_Name and Bonus.
7. Display the sum, maximum, minimum and average Salary.
8. Display the count of records where Department = 'HR'.

Solution

1. SELECT Worker_ID, Joining_Date FROM Worker WHERE Department = 'Admin';
2. SELECT * FROM Worker ORDER BY Salary;
3. SELECT * FROM Worker BETWEEN 50000 AND 90000;
4. SELECT * FROM Worker WHERE First_Name LIKE '%a';
5. SELECT * FROM Worker WHERE Department = 'HR' OR Department = 'Account';
6. SELECT First_Name, Last_Name, 0.01 * Salary AS "BONUS" FROM Worker;
7. SELECT SUM(Salary), MAX(Salary), MIN(Salary), AVG(Salary);
8. SELECT COUNT(*) FROM Worker WHERE Department = 'HR';



CHAPTER 4

Introduction to Loops

In this chapter

We will learn a very useful feature of the programming languages that provides an efficient way to code when we want to perform a task many times. This feature is named as loop. We will also learn different types of loops. We will write several interesting programs using different loops.

4.1 IMPORTANCE OF LOOP IN PROGRAMMING LANGUAGE

Let us consider a simple C program as shown in Example 4.1. The program has 5 printf() statements. As a result, the program displays the sentence "I read in class X under SEBA" 5 times. Imagine for a while that we need to display the sentence 1000 times. Shall we write 1000 printf() statements?

```
#include<stdio.h>

int main()
{
        printf("I read in class X under SEBA");
        printf("I read in class X under SEBA");
        printf("I read in class X under SEBA");
        printf("I read in class X under SEBA");
        printf("I read in class X under SEBA");

        return 0;
}
```

Example 4.1: A C program to print "I read in class X under SEBA" 5 times.

We can obviously write 1000 printf() statements. But this is a horrible way of programming. Question is, is there any better way to do that? The answer is yes, using a concept called **loop.**

Whenever we need to do a work many times, we can write the work only once and put it inside a loop. Every programming language has this support. Loop is used for repeated execution of a set of statements in a program. We can rewrite the previous program as shown in Example 4.2. We see that we have written the printf() statement only once instead of 5.

```
#include<stdio.h>
int main()
{
        int i = 0;

        while (i < 5)
        {
                printf("I read in class X under SEBA \n");
                i++;
         }

         return 0;
}
```

Example 4.2: A C program to print "I read in class X under SEBA" 5 times.

We can do many other interesting and important things using a loop. We will learn them gradually.

Quick Activity : Run both the programs of Example 4.1 and 4.2 on your computer. Then modify the program to display a sentence 9 times.

4.2 TYPES OF LOOPS IN C

C programming language provides three types of loop constructs: while, do-while and for. We will learn them one by one.

4.2.1 While loop

Let us start with the while loop. Example 4.2 uses a while loop construct. The signature of a while loop is shown using Figure 4.1.

```
while (condition)
{
    //statements
}
//outside statements
```

Figure 4.1: Signature of a while loop

The statements inside the while loop are executed as long as the condition evaluates as true. In case of Example 4.2, the condition was "value of the variable i is to be less than 5" and the statements were the printf() and the increment operation on i.

At first, the condition of the while is checked. In case it evaluates as true, all the statements inside the loop are executed. After executing all the statements of the loop every time, the condition is checked. If the condition happens to be true, the statements are executed again. When the condition evaluates as false, the control comes out of the loop and executes the outside statements.

Let us see another C program (Example 4.3) that uses a while loop. The program asks for entering an integer 5 times. Every time a number is entered, the program displays the number back. The loop stops as soon as the value of the variable ***index*** reaches 5.

```
#include<stdio.h>

int main()
{
        int var;
        int index = 0;

        while( index < 5 )
        {
                printf("\n Enter a number: ");
                scanf("%d", &var);

                printf("\n You entered %d", var);

                index++;
        }

        return 0;
}
```

Example 4.3: A C program for echoing an integer input 5 times.

In the last two examples, the loop ran for a fixed number of times (5 times). We may use a variable and make the situation dynamic where the loop will run the number of times the user wants. Example 4.4 demonstrates the same.

```
#include<stdio.h>

int main()
{
        int var;
        int index = 0, n;

        printf("Enter number of times you want to run the loop: ");
        scanf("%d", &n);

        while( index < n )
        {
                printf("\n Enter a number: ");
                scanf("%d", &var);

                printf("\n You entered %d", var);

                index++;
        }

        return 0;
}
```

Example 4.4: A C program for echoing an integer input.

In the program, we are taking an input in a variable “n” (line number 7). The value stored in this variable will tell how many times we want the while loop to run. We have given a condition index<n in the while loop. This means the loop will run as long as the value stored in the variable “index” is lesser than that of “n”. The initial value of index is 0 and at each step of the loop, we increment the value of index (line number 13) by 1.

So there comes a point when index and n become equal. Then the loop condition becomes false and the loop ends there. The control will come out of the loop and it will execute the statement outside the loop. Here, the statement is return 0 (line number 15).

4.2.2 Do-while loop

In the while loop construct, first the condition is checked and then the statements are executed as long as the condition is true. Now, let us say, we want to execute the statements **at least once** irrespective of the condition and then the further execution should depend on the condition. In that case, we may use another loop in C called **do-while.**

Example 4.5 demonstrates the program of Example 4.2 using a do-while loop.

```
#include<stdio.h>

int main()
{
        int i = 0;

        do
        {
                printf("I read in class X under SEBA \n");
                i++;
        }
        while (i < 5);

        return 0;
}
```

Example 4.5: A C program to print "I read in class X under SEBA" 5 times using a do-while loop.

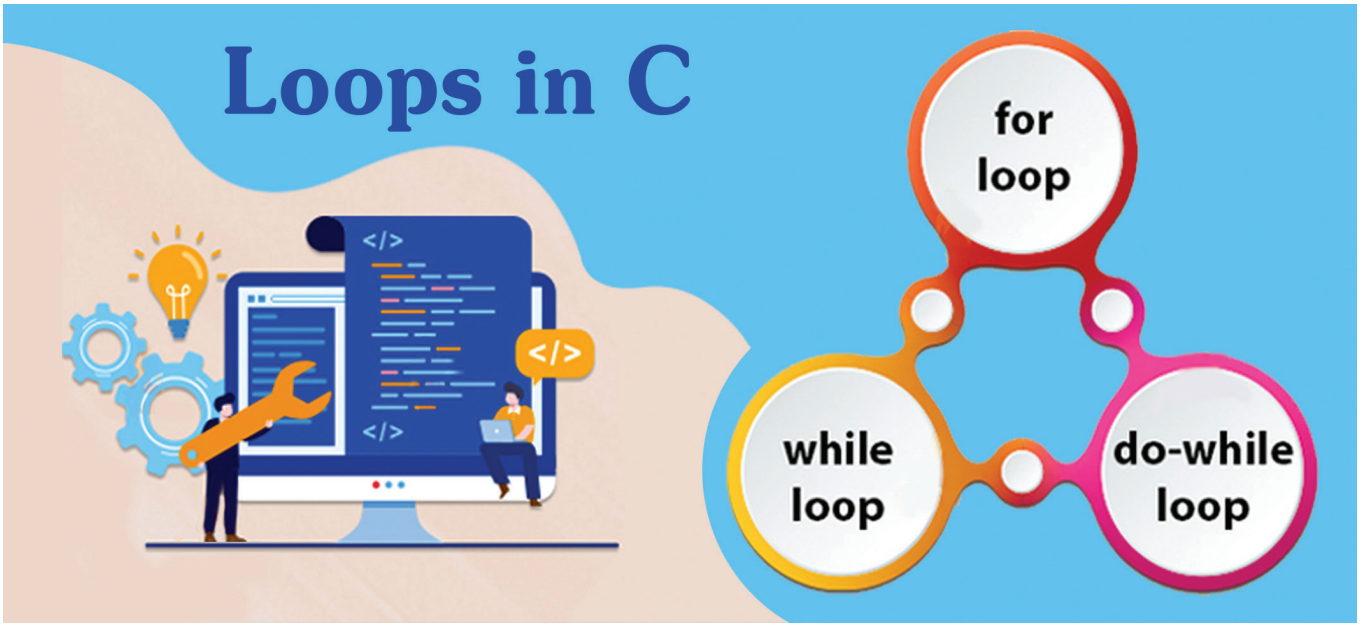
When we execute the program, we observe that the program output of Example 4.2 and Example 4.5 are the same. Then the question is why and how can we use a do-while loop in a more realistic situation. To understand this, let us revisit Example 4.3. Here the program executes a set of statements any number of times a user wants. But the user must decide before the loop starts its execution.

Now let us see the program of Example 4.6. Here, the program executes the set of statements once (echoching the entered number). Then the program gives an option to the user to stop. In case the user chooses so, the statements enclosed under the loop will not execute further and the control comes out of the loop.

```c
#include<stdio.h>

int main()
{
        int var, choice =1;

        do
        {
              printf("\n Enter a number: ");
              scanf("%d", &var);
              printf("\n You entered %d", var);

              printf("\n Do you want to stop? Enter 0, otherwise press any integer: ");
              scanf("%d", &choice);
        }
        while (choice != 0);

        return 0;
}
```

Example 4.6: A C program for echoing an integer input using do-while loop.

Controlling of the loop execution in Example 4.6 is achieved using a variable "choice". After executing the set of statements once, the program asks the user to enter a value to the variable "choice". The loop will run as long as the value of the variable is not zero. If the user wishes to stop further execution of the loop, he/she enters 0 whenever asked for.



Let us take another program that uses a do-while loop (Example 4.7). We have modified the program of Example 4.6 to find the summation of all the numbers a user enters. In order to do so, we took another integer variable "sum" and we initialized it to zero (line number 5 in the program).

```
#include<stdio.h>

int main()
{
        int var, choice =1;
        int sum = 0;

        do
        {
                printf("\n Enter a number: ");
                scanf("%d", &var);
                printf("\n You entered %d", var);

                sum = sum + var;

                printf("\n Do you want to stop? Enter 0, otherwise press any integer: ");
                scanf("%d", &choice);
        }
        while (choice != 0);

        printf("\n Summation of the numbers entered till now is: %d", sum);

        return 0;
}
```

Example 4.7: A C program for finding summation of integers using do-while loop.

Next, the user enters a number from the keyboard and stores it in the variable "var" (line number 9 in the program). We then add this number with the previously calculated summation stored in the variable "sum" and the summation is updated with the newly calculated value (see line number 11 in the program - written in bold font).

This process (reading a number from the keyboard and adding to the summation) continues as long as the user wants to do so. When the user enters zero (line number 13 in the program),

the loop terminates and the program prints the latest value of the variable "sum". The variable contains the summation of all the numbers the user enters.

We have learnt about Flowchart in class IX. Do you remember that? Have a look in Figure 4.3. This shows a flowchart for the problem we have just seen - to find the summation of a series of integers entered using the keyboard.

This flowchart depicts the behaviour of the do-while loop. At first, one number is read from the keyboard and it is added to the previous summation (set as 0 in the beginning). Then we decide whether we want to add more numbers or not. Accordingly, the control takes its path.

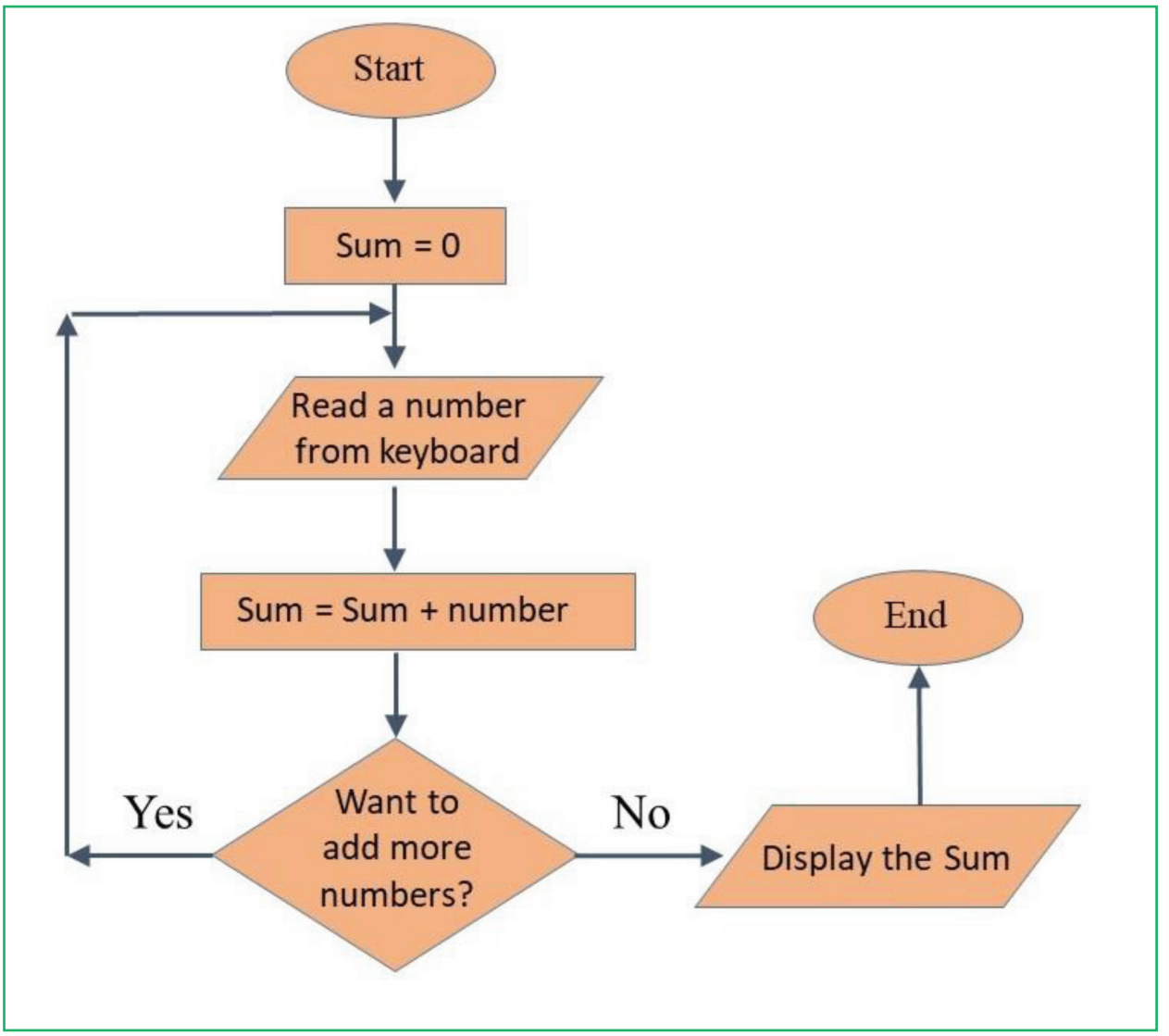


Figure 4.3: Flowchart for finding summation of integers using loop.

4.2.3 For loop

In addition to the above-mentioned loops (while and do-while), we use another loop in C programming language - called for loop. A **for** loop has three portions :

1. Initialization expression
2. Condition checking or testing expression
3. Update expression

The signature of the for loop is shown in Figure 4.4. We put a semicolon (;) to denote the end of an expression. The initialization expression contains all the statements we want to execute before entering the loop. This generally contains initialization statements. The second expression contains all the controlling statements of the loop such as condition checking. The update expression contains the statements that need to be executed in each iteration of the loop.

```
for (initialization expression; test expression; update expression)
{
        // body of the loop statements we want to execute
}
//outside statement
```

Figure 4.4: Signature of a for loop

We will now write the program of Example 4.2 using a for loop. If we recall Example 4.2, we initialized the variable i to zero before the start of the loop in line number 4. We need to write this statement in the initialization expression of the for loop.

Next, we put a condition in the while loop whether the value of the variable i is less than 5 or not. We now need to write this statement in the condition checking expression of the for loop. In each iteration of the while loop, we have updated the value of the variable i by 1. We will do the same thing in the update expression. Refer to the line number 5 of Example 4.8. The enclosed statement of the loop is a printf() statement and this is same in both the programs.

```
#include<stdio.h>

 int main()
 {
        int i;

         for (i = 0; i < 5; i++)
         {
                printf("I read in class X under SEBA \n");
         }

        return 0;
 }
```

Example 4.8: A C program to print "I read in class X under SEBA" 5 times using a for loop.

Let us now write a C program using a for loop to evaluate the following expression.

sum = 1+2+3+4+ . . . +N

Here, N is an input from the user.

The program is shown using Example 4.9. We get the value for N using a scanf() function in line number 6. In order to find the summation, we are using a for loop that runs for i=1 to N. At each step of the loop, we are adding the value of the variable i with the previously calculated sum. This step is similar to the program of Example 4.7.

As the value of the variable i varies from 1 to N, we get the summation of its values as 1+2+3+4+ . . . +N at the end of the loop. The loop runs as long as the value of i is less or equal to N. The loop ends when i becomes N+1.

```
#include<stdio.h>

int main()
{
        int i, N, sum=0;

        printf("Enter the value of N: ");
        scanf("%d",&N);

        for (i = 1; i <= N; i++)
        {
            sum = sum + i;
        }

        printf("\n The summation is %d", sum);

        return 0;
}
```

Example 4.9: A C program to print the summation of a series 1+2+3+4+ . . . +N.

Now we can extend this program to find the summation of N arbitrary numbers entered using the keyboard. The purpose of the for loop in the above program was to run the loop N times. In this program as well, we need a loop that runs N times. Thus we will not change the for loop statement.

We need to change the statements that are written inside the loop. As we need to get a number from the user at each iteration of the loop, we need to use a scanf() function. We use a variable "var" to read a number from the keyboard (line number 9 of Example 4.10). Then we add this number with the previously calculated summation to get the new summation (line number 12 of Example 4.10).

```
#include<stdio.h>

int main()
{
        int i, N, var, sum=0;

        printf("Enter the value of N: ");
        scanf("%d", &N);

        for (i = 1; i <= N; i++)
        {
            printf("\n Enter a number: ");
            scanf("%d", &var);
            printf("\n You have entered: %d", var);

             sum = sum + var;
        }

        printf("\n The summation is %d", sum);


        return 0;
}
```

Example 4.10: A C program to print the summation of N numbers entered using the keyboard.

In all the programs written using a for loop, we have used only one statement in each expression. We may put more than one statement in an expression. This is beyond the scope of the book and hence we will not write any programs using that feature.

4.3 SOME MORE EXAMPLES USING LOOP

In this section, we will do some interesting things using loop. We will use the for loop to write the programs here. You may try using a while loop as well.

A. Let us write a C program to display the following pattern on our monitor.

```
X
X X
X X X
X X X X
X X X X X
```

```
#include<stdio.h>

int main()
{
        int i;

        for (i=1; i <= 1; i++)
        {
                printf(" X ");
        }
        printf("\n");

        for (i=1; i <= 2; i++)
        {
                printf(" X ");
        }
        printf("\n");

        for (i=1; i <= 3; i++)
        {
                printf(" X ");
        }
        printf("\n");

        for (i=1; i <= 4; i++)
        {
                printf(" X ");
        }
        printf("\n");

        for (i=1; i <= 5; i++)
        {
                printf(" X ");
        }
        printf("\n");

        return 0;
}
```

Example 4.11: A C program to display a pattern on monitor.

The purpose of the program is simply to display "X" in a systematic way. In the first line, the program needs to display only one X. In the second line, it displays X two times with a space in between. Similarly, in the third line, the program displays X three times and so on.

You can see from the program (Example 4.11) that we have used a for loop to display "X". In the first line as we need to display X only time, we used the loop so that it iterates only one time and it displays X in its iteration. Then we printed a line using printf("\n") to go to the next line.

In the second line, we need to display X two times. Thus the loop needs to iterate two times. We used the same signature in the loop as before.

As we are required to display 5 lines, we have used 5 for loops in the program. Job of all the for loops was to display "X" but a different number of times. Thus only the condition checking expressions of the for loops were different.

B. Let us write a C program to display the following pattern on our monitor.

```
X X X X X
X X X X
X X X
X X
X
```

This pattern is similar to the above pattern. The difference is that in the first line, we need to display "X" 5 times instead of one. Then 4 times in the next line and so on. The program is presented using Example 4.12. the first for loop in writen so that it iterates 5 times. The second one in writen so that it iterates 4 times and so on.

```
#include<stdio.h>

int main()
{
        int i;

        for (i=1; i <= 5; i++)
        {
                printf(" X ");
        }
        printf("\n");
```

```
        for (i=1; i <= 4; i++)
        {
                printf(" X ");
        }
        printf("\n");

        for (i=1; i <= 3; i++)
        {
                printf(" X ");
        }
        printf("\n");

        for (i=1; i <= 2; i++)
        {
                printf(" X ");
        }
        printf("\n");

        for (i=1; i <= 1; i++)
        {
                printf(" X ");
        }
        printf("\n");

        return 0;
}
```

Example 4.12: A C program to display a pattern on monitor.

C. Let us write a C program to display the following pattern on our monitor.

X X X X X
X X X X
X X X
X X
X
X X
X X X
X X X X
X X X X X

This pattern is also similar to the above two patterns. We need to display the character "X" a different number of times in different lines.

- Line 1: 5 times
- Line 2: 4 times
- Line 3: 3 times
- Line 4: 2 times
- Line 5: 1 times
- Line 6: 2 times
- Line 7: 3 times
- Line 8: 4 times
- Line 9: 5 times

We will be using 9 for loops. Each for loop displays the character "X". First loop will display 5 times, the second loop will display 4 times, and so on as per the list mentioned above. Refer to Example 4.13 for the complete program.

```
#include<stdio.h>

int main()
{
        int i;

        for (i=1; i <= 5; i++)
        {
                printf(" X ");
        }
        printf("\n");

        for (i=1; i <= 4; i++)
        {
                printf(" X ");
        }
        printf("\n");

        for (i=1; i <= 3; i++)
        {
                printf(" X ");
        }
        printf("\n");
        for (i=1; i <= 2; i++)
        {
```

```
            printf(" X ");
        }
        printf("\n");

        for (i=1; i <= 1; i++)
        {
            printf(" X ");
        }
        printf("\n");

        for (i=1; i <= 2; i++)
        {
            printf(" X ");
        }
        printf("\n");

        for (i=1; i <= 3; i++)
        {
            printf(" X ");
        }
        printf("\n");

        for (i=1; i <= 4; i++)
        {
            printf(" X ");
        }
        printf("\n");

        for (i=1; i <= 5; i++)
        {
            printf(" X ");
        }
        printf("\n");

        return 0;
}
```

Example 4.13: A C program to display a pattern on monitor.

We observe that the length of the program is increasing as we are writing more loops. We also need to observe that all the for loops were essentially doing the same work - displaying the character "X". In the next chapter, we will make use of this observation and learn something interesting where we will write a loop inside another loop. This will significantly reduce the program length.

D. A C program to extract the individual digits from a given integer.

Here, we are given an integer and we need to find all the digits from it

For example, suppose we consider an integer 8724. The digits of the integer will be 8, 7, 2, and 4.

Let us try to develop the logic of the problem. Then we can write the logic using C programming language's syntax to build the program.

In order to extract the digits, we can use the remainder or modulus operation (%). If we perform a modulus operation on the given number with 10, we can extract the last digit, that is 4 because we know **8724 % 10 = 4.**

If we want to extract the next digit from the last, we need to have a number 872. Now, 872 can be calculated from 8724 by performing a division operation by 10. That is, **8724 / 10 = 872.** We perform these two operations repeatedly - the modulus operation to extract a digit and the division operation to reduce the number by one tenth. Refer to the program represented using Example 4.14. The modulus operation is performed in line number 10 and the division operation is performed in line number 12.

```
#include<stdio.h>

int main()
{
        int number, temp, digit;

        printf("Enter the integer: ");
        scanf("%d", &number);

        temp = number;

        while(temp > 0)
        {
                digit = temp % 10;
                printf("\n Extracted digit is %d", digit);
                temp = temp / 10;
        }

        printf("\n");

        return 0;
}
```

Example 4.14: A C program to extract the individual digits from an integer.

In the program, we did not want to change the actual number that is taken as input from the keyboard and hence, we took help of a temporary variable "temp". The variable is initialized with the value that was read from the keyboard (line number 7 in Example 4.14). Then the modulus and division operations were performed on the temporary variable (line number 10 and 12 in Example 4.14).

In the second iteration of the loop, the value of temp is **872.** The extracted digit becomes **872%10 = 2.** Then temp is divided by 10 to get the new temp as **872 / 10 = 87.**

As the current temp value is greater than 0, the loop will enter its third iteration. In this iteration, the digit is extracted as **87%10 = 7** and the new temp becomes **87 / 10 = 8.**

Temp is still greater than 0 and hence the loop enters into its fourth iteration. Now, the digit is extracted as **8%10 = 8** and the new temp is **8 / 10 = 0.**

As the value of the temp becomes zero, the condition of the while loop (line number 8) is not true and thus the control will come out of the loop and it prints a new line (line number 14) and the program terminates.

You may compare this program with that of Example B.4 from class IX book. This will help you to understand the importance of loop. In Example B.4, we restricted the length of the number to be of 5 digits and we had to use seperate variables to store the digits. But the current program uses a while loop and uses a single variable "digit". There is no restriction on the length of the number and the program works for any integer.

E. A C program to find the summation of digits of a given integer.

For example, if the integer is 8724, the output will be 21.
21 is formed by adding the digits: 8+7+2+4.

In the previous example, we have already extracted the individual digits of an integer. Now, we need to find the summation of these digits. As soon as a digit is extracted from the number, we will add this digit to a previously calculated summation. We will repeatedly perform this operation. The logic is similar to the one we used while finding the summation of a series of numbers.

```
#include<stdio.h>

   int main()
{
        int number, temp, digit, sum=0;
```

```
        printf("Enter the integer: ");
        scanf("%d", &number);
        temp = number;

        while ( temp > 0 )
        {
                digit = temp % 10;
                printf("\n Extracted digit is %d", digit);

                sum = sum + digit;

                temp = temp / 10;
        }

        printf("\n The summation is: %d", sum);

        printf("\n");

        return 0;
}
```

Example 4.15: A C program to find the summation of the digits of an integer.

Example 4.15 shows the program. In this program, we brought the following modifications to the program of Example 4.14.

1. Declared a variable "sum" and initialized it to zero (line number 4)
2. Added the extracted digit to the previous summation and updated the new summation (line number 12)
3. Finally, we displayed the summation using a printf() statement (line number 15).

F. A C program to check whether a number is prime or not.

We know that a number is prime when it is not divisible by any number other than 1 or itself. We will use this very logic in the program.

Refer to the program in Example 4.16. We first take the number as input from the keyboard. Then we run a loop from **i=0 to number/2** to check the divisibility of the number. It is sufficient to run the loop till number/2. We check the divisibility of the number using a modulus (%) operation and double equal to (= =) operation.

```
#include<stdio.h>

int main()
{
        int number, i, flag = 0;

        printf("Please input  a number: ");
        scanf("%d", &number);

        for( i=2; i <= number/2; i++ )
        {
                if ( number % i = = 0 )
                {
                        flag = 1;
                        break;
                }
        }

        if ( flag = = 0 && number != 1 )
                printf("%d is a prime number.\n",number);

         else
                printf("%d is not a prime number",number);

        return 0;
}
```

Example 4.16: A C program to check whether a number is prime or not.

As soon as the number becomes divisible by another number (line number 9), we set a flag (line number 11) and come out of the loop using a "break" statement (line number 12). As we come out of the loop, we check whether the flag is set or not.

If the flag is not set and the number is not 1, then we announce the number to be a prime number. Otherwise the number is not a prime number. We use printf() statements for the announcement.

In the next chapter, we will write one loop inside another loop to tackle some more interesting problems.

Questions:

1. Why do we use a loop in a C program?

2. Do we need to use only one type of loop in a C program? Justify your answer by writing a C program.

3. What will happen if we write a while loop with 1 in place of the condition ? Try it in a simple C program. Hint:

```
while (1)
{
printf("We must raise our voice against corruption \n");
}
```

4. Name different portions of a for loop. Can we put more than one statement within a portion?

5. Answer with TRUE or FALSE.

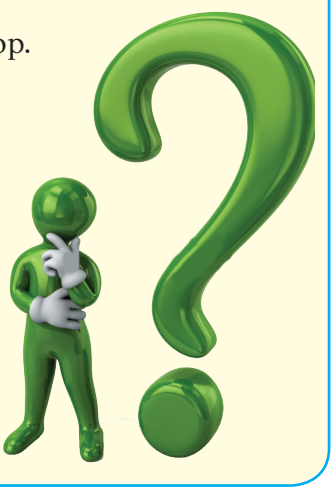
(i) If the condition of the while loop is false, the control comes to the second statement inside the loop.

(ii) We can use at most three loops in a single C program.

(iii) The statements inside the do-while loop executes at least once even if the condition is false.

(iv) Only the first statement inside the do-while loop executes when the condition is false.

(v) In a do-while loop, the condition is written at the end of the loop.



6. Programming exercises:

A. Write a C program to find the summation of the following series

(a). $1^2 + 2^2 + 3^2 + 4^2 + \ldots + N^2$

(b). $1^3 + 2^3 + 3^3 + 4^3 + \ldots + N^3$

(c). 1*2 + 2*3 + 3*4 + . . . + N*(N+1)

B. Write a C program to continuously take a number as input and announce whether the number is odd or even. Hint: use do-while loop.

C. Write a C program to display the following pattern.

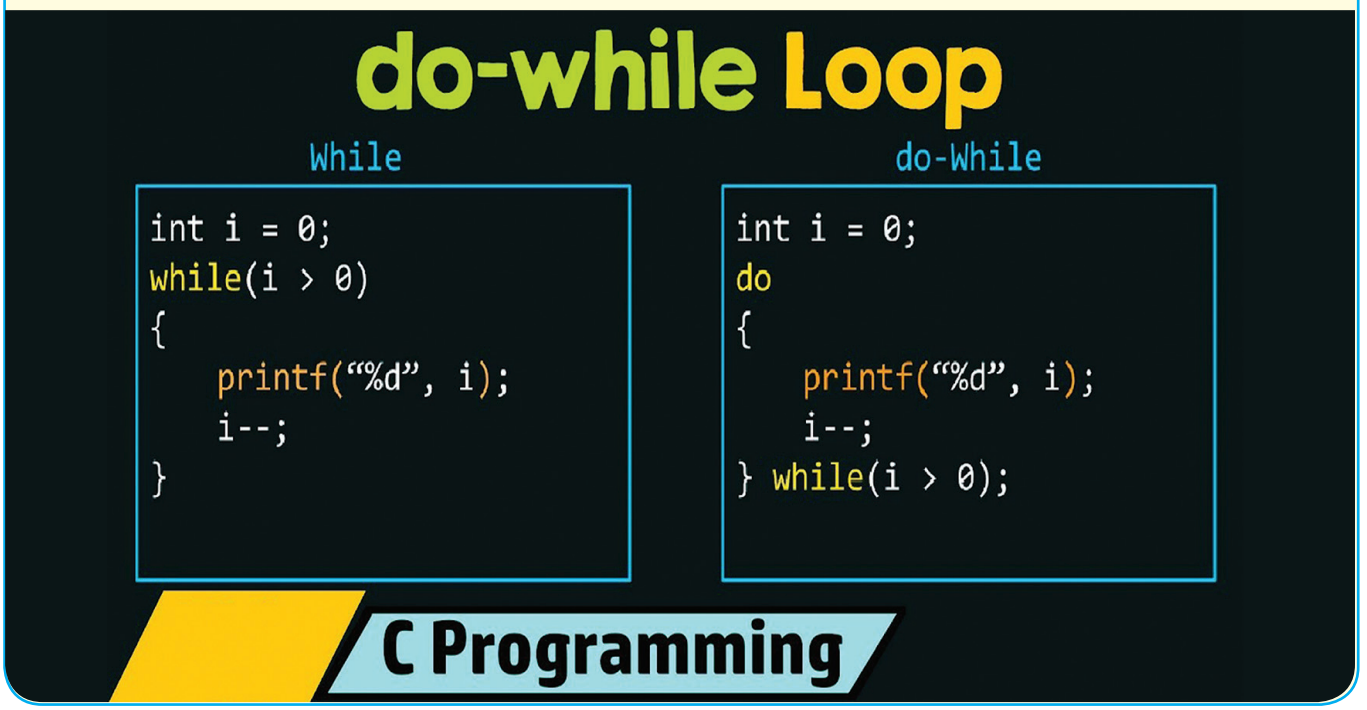
1
1 1
1 1 1
1 1 1 1
1 1 1 1 1

D. Write a C program to display the following pattern.

5
5 4
5 4 3
5 4 3 2
5 4 3 2 1

E. Write a C program to display the following pattern.

5 4 3 2 1
5 4 3 2
5 4 3
5 4
5



CHAPTER 5

Nested loops in C

In this chapter

We have learned 3 different types of loops. We used them in our programs. In this chapter, we will learn putting one loop inside another, called nested looping.

5.1 INTRODUCTION TO NESTED LOOP

Whenever we write a loop inside another loop, we call it a **nested loop**. The first loop is called the **outer loop** and the loop that appears inside another loop is called the **inner loop**. There can be more than one inner loop in a program. The nested loop construct simplifies programming in many cases. This also reduces the program length.

The signature of a nested loop is presented below. Here loop 1 is called the outer loop and loop 2 is called the inner loop.

```
loop 1
{
        loop 2
        {
                statements
        }
}
```

We present a simple nested loop construct below where both the outer and inner loops are taken as while loops. The outer loop runs for i = 0 to 9 and for every i value, the inner loop runs for j = 0 to 19. The inner loop simply displays the values of i and j at each step. In the next section, we will solve some interesting problems using nested loop constructs.

```
int i = 0;

while ( i < 10 )
{
        int j = 0;
        while ( j < 20)
        {
                printf( " %d %d \n ", i, j );
                j++;
        }

        i++;
}
```

5.2 PROBLEM SOLVING USING NESTED LOOP

Recall the programs of Example 4.12 and Example 4.13 of the previous chapter. We used one loop for displaying one line. As we were to display 5 and 9 lines, we used loop 5 and 9 number of times in the programs. But the same task can be done with a lesser number of loops.

Let us take Example 4.12. We need to display the following pattern.

```
X X X X X
X X X X
X X X
X X
X
```

Let us have a close look at the pattern and write the tasks to be done in each line as follows.

Line Number	Task to be done
1	To display X 5 times
2	To display X 4 times
3	To display X 3 times
4	To display X 2 times
5	To display X 1 time

We have seen that in every line, the task is the same - to display X. The difference is in the number of times X is to be displayed. If we want to write a general formula, we may say that in the kth line, we need to display X (6-k) times.

In line number 1, k = 1, thus, 6-k is 6 - 1 = 5.
In line number 2, k = 2, thus, 6-k is 6 - 2 = 4.
In line number 3, k = 3, thus, 6-k is 6 - 3 = 3.
In line number 4, k = 4, thus, 6-k is 6 - 4 = 2.
In line number 5, k = 5, thus, 6-k is 6 - 5 = 1.

So, we are to write a loop in the following way.

```
loop: iterate over k from 1 to 5
{
        Display X (6-k) times.
        Go to next line
}
```

Now, if we use a for loop, the above segment expands to the following.

```
for (k=1; k<=5; k++)
{
        Display X (6-k) times.
        Go to next line
}
```

Next, we need to write the code for the display part. We can again use another for loop there. We need to use a loop so that it runs (6-k) times and in each iteration, it needs to display the character X. Thus we get the following code segment.

```
for (k = 1; k <= 5; k++)
{
        for ( i = 1; i <= (6-k); i++)
        {
                printf("X ");
        }

        printf("\n");
}
```

The complete program is shown using Example 5.1. The for loop in line number 5 is the outer loop and the for loop in line number 7 is the inner loop. We see a drastic reduction in the program length as compared to Example 4.12.

```
#include <stdio.h>

int main()
{
        int k, i;

        for (k = 1; k <= 5; k++)
        {
                for ( i = 1; i <= (6-k); i++)
                {
                        printf("X ");
        }
                printf("\n");
        }

        return 0;
}
```

Example 5.1: A C program to display a pattern.

Now, let us make the program flexible. The last program displayed 5 lines and the number of characters in each line decreased by one - 5 characters in line 1, 4 in line 2 and so on. If we want the number of lines to be dynamic, we may give an option to the user to enter from the keyboard. Then the program should display the patterns accordingly.

If the user enters 9, the program should display 9 characters (X) in the first line. Then, 8 characters in line 2, 7 in line 3 and so on. The pattern is shown below.

```
X X X X X X X X X
X X X X X X X X
X X X X X X X
X X X X X X
X X X X X
X X X X
X X X
X X
X
```

We observe that if we want N lines to be displayed, the number of characters in the first line is N. In the next line, it is N-1, then N-2 and so on.

In the last program, we started with 5 characters. In this case, we need to start with N characters. The remaining logic is the same in both cases. When we needed to start with 5, we used (6-k) in the program. That means, in this case, we need to use (N+1-k).

The complete program is shown using Example 5.2. We initially give an option to the user for entering the number of lines to be displayed. Then we run the for loops (as in Example 5.1) using N as the boundary variable. Here, we write **N** in place of **5** in the outer loop and **(N+1-k)** in place of **(6-k)** in the inner loop.

```
#include <stdio.h>

int main()
{
int k, i, N;

        printf("Please enter number of lines to be displayed: ");
        scanf("%d", &N);

        for (k = 1; k <= N+1; k++)
        {
                for ( i = 1; i <= (N+1-k); i++)
                {
                        printf("X ");
                }

                printf("\n");
        }

        return 0;
}
```

Quick Activity : Write the above program using whole loops. Execute both the programs on your computer.

Example 5.2: A C program to display a dynamic pattern.

Now let us develop a C program to display the pattern considered in Example 4.13 using nested loops. The pattern is shown below.

```
X X X X X
X X X X
X X X
X X
X
X X
X X X
X X X X
X X X X X
```

We observe that the number of characters displayed per line in the above pattern initially decreases (till line 5) and then it increases. We will write code separately - one for the decreasing portion and the other is for the increasing portion.

```
X X X X X
X X X X
X X X              Portion 1: Decreasing pattern
X X
X      ------------
X X
X X X              Portion 2: Increasing pattern
X X X X
X X X X X
```

We have already written a program for the decreasing portion in Example 5.2. To design the complete pattern, we will add the code for the increasing portion.

The increasing portion is shown below.

```
X X
X X X
X X X X
X X X X X
```

If we analyse the above pattern, we see the following.

- Line 1: 2 times X
- Line 2: 3 times X
- Line 3: 4 times X
- Line 4: 5 times X

When we are in line number k, we need to display the character (k+1) times. We need to continue this for k = 1 to 4.

Thus to produce the required pattern, we need to write code as below.

```
for k = 1 to 5
        Display X (6-k) times.
        Go to the next line.

Display a line.

for k = 1 to 4
        Display X (k+1) times.
        Go to the next line.
```

We have already learned how to write code segments for the above steps. We will simply put those to develop the complete program. Example 5.3 shows the program.

In the program, line numbers 5 to 12 produce the first portion (decreasing portion) and line numbers 13 to 20 produce the second portion (increasing portion).

```
#include <stdio.h>

int main()
{
        int k, i;
        // decreasing portion
        for (k = 1; k <= 5; k++)
        {
                for ( i = 1; i <= (6-k); i++)
                {
                        printf("X ");
                }
                printf("\n");
        }
        // increasing portion
        for (k = 1; k <= 4; k++)
        {
                for ( i = 1; i <= k+1; i++)
```

```
        {
            printf("X ");
        }
        printf("\n");
    }

    return 0;
}
```

Example 5.3: A C program to display a pattern.

Quick Activity : Extend the above program to make it dynamic where users will enter the number of lines for the pattern from the keyboard.

Till now, we have written programs where we used one loop to display the patterns for a line. Now, let us see some patterns where we need to use **more than one loop for displaying a single line**. Let us write a C program to display the following pattern.

```
    X
   X X
  X X X
 X X X X
X X X X X
```

The first thing that comes to our mind is that the number of times the character X is displayed in each line is the line number itself. At kth line number, we need to display the character k times. But this is not sufficient. Even if we need to display X only one time in the first line, X is not displayed in the first place. In the first line, we need to display X in 5th place. In the second line, X is to be displayed in 4th place, next, in the 3rd place and so on.

This means we need to leave some space before displaying the character X. The following pattern shows the same.

```
_ _ _ _ X
_ _ _ X X
_ _ X X X
_ X X X X
X X X X X
```

The following table lists the task to be done in each line.

Line number	First task	Second task
1	To display 4 spaces	To display X, 1 time
2	To display 3 space	To display X, 2 time
3	To display 2 spaces	To display X, 3 time
4	To display 1 space	To display X, 4 time
5	No Space to display	To display X, 5 time

The pseudo-code for this can be written as follows.

```
for k = 1 to 5
        Display SPACE (5-k) times
        Display X, k times
        Go to the next line
```

Example 5.4 shows the complete C program for displaying the pattern. We used the variable "i" for displaying the space and the variable "j" for displaying the character X. We can see that there is one outer loop in the program and there are two inner loops.

```
#include <stdio.h>

int main()
{
        int i, j, k;

        for (k = 1; k<=5; k++)
        {
                for(i = 1; i<=(5-k); i++)
                {
                        printf(" ");
                }

                for(j = 1; j<= k; j++)
                {
                        printf("X");
                }

                printf("\n");
        }

        return 0;
}
```

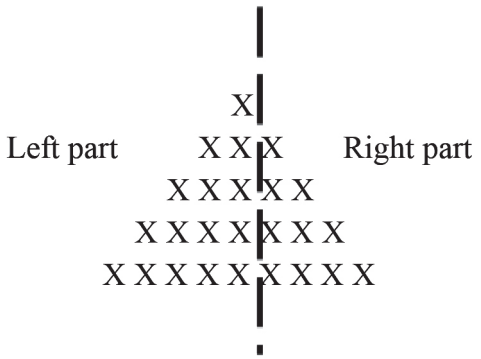
Example 5.4: A C program to display a pattern.

Quick Activity : Extend the above program to make it dynamic where users will enter the number of lines for the pattern from the keyboard. The number of lines to be displayed may be stored in a variable N.

Let us now extend the program in Example 5.4 for displaying the following pattern.

```
    X
   X X X
  X X X X X
 X X X X X X X
X X X X X X X X X
```

If we analyze the above pattern, we see that the pattern can be divided into two parts as follows.



We have already written a program for the left part. Let us now develop code for the right part. Then we will add this code segment to the existing program to produce the required pattern.

In the right part as well, we need to display character X in each line. In the first line, we do not display any character. In the second line, we display X one time. Next, we display 2 times and so on.

Thus the complete task for displaying the required pattern can be summarized as follows.

Line number	Task 1	Task 2	Task 3
1	To display 4 spaces	To display X 1 time	To display X 0 time
2	To display 3 spaces	To display X 2 time	To display X 1 time
3	To display 2 spaces	To display X 3 time	To display X 2 time
4	To display 1 space	To display X 4 time	To display X 3 time
5	To display 0 space	To display X 5 time	To display X 4 time

Task 1 and Task 2 are for the left part and Task 3 is for the right part. The pseudo-code for all the tasks can be written as follows.

```
for k = 1 to 5
        Display SPACE (5-k) times.
        Display X, k times.
        Display X, (k-1) times.
        Go to the next line.
```

Now the complete program is shown in Example 5.5. The first inner for loop is responsible for performing Task 1 (line numbers 7 to 10). The second inner for loop is responsible for performing Task 2 (line numbers 11 to 14). The third inner for loop is responsible for performing Task 3 (line numbers 15 to 18).

```
#include <stdio.h>

int main()
{
        int i, j, k, p;

        for (k = 1; k <=5; k++)
        {
                for(i = 1; i<=(5-k); i++)
                {
                        printf(" ");
                }

                for(j = 1; j <= k; j++)
                {
                        printf("X");
                }

                for(p = 1; p <= (k-1); p++)
                {
                        printf("X");
                }

                printf("\n");
        }

        return 0;
}
```

Example 5.5: A C program to display a pattern.

We know that there may be more than one way to program a problem. Let us learn another way to program for displaying the same pattern. Here, we will divide the task per line in two parts.

i) to display required number of spaces
ii) to display required number of X

Line number	Task 1	Task 2
1	To display 4 spaces	To display X, 1 time
2	To display 3 spaces	To display X, 3 time
3	To display 2 spaces	To display X, 5 time
4	To display 1 spaces	To display X, 7 time
5	To display 0 spaces	To display X, 9 time

In case of Task 1, at the kth line, we need to display (k-1) spaces. In case of Task 2, at the kth line, we need to display X (2k-1) times. For the first line, it is 2*1-1 = 1. For the second line, k = 2 and 2k-1 = 2*2 - 1 = 3 and so on.

Thus the pseudo-code for both the tasks can be written as follows.

```
for k = 1 to 5
        Display SPACE (5-k) times.
        Display X, 2k-1 times.
        Go to the next line.
```

The complete program is shown in Example 5.6. We see that we have used only two inner loops instead of three.

```
#include <stdio.h>

int main()
{
        int i, j, k, p;

        for (k = 1; k <=5; k++)
        {
                for(i = 1; i<=(5-k); i++)
                {
                        printf(" ");
                }

                for(j = 1; j <= (2*k - 1); j++)
                {
```

```
                printf("X");
            }

            printf("\n");
        }

        return 0;
}
```

Example 5.6: A C program to display a pattern.

Quick Activity : Modify the program of Example 5.5 to display the patterns in 9 lines.

Let us consider a pattern that displays integers instead of a fixed character. For instance, consider the pattern displayed below.

When we compare this pattern with the previous one, we see that the space placement is the same. After displaying the required number of spaces, we need to display some integers in each line. We also see that every line starts with 1 and goes upto a number.

```
        1
      1 2 3
    1 2 3 4 5
  1 2 3 4 5 6 7
1 2 3 4 5 6 7 8 9
```

For the first line, it goes up to 1. For the second line, it goes up to 3. In case of third line, it goes up to 5 and so on. If we want to generalize, for the kth line, the limit will be 2k-1.

Thus in order to display the required pattern, we need to perform two tasks per line.

i) to display required number of spaces,
ii) to display integers from 1 to 2k-1.

Thus the pseudo-code for the same may be written as follows.

```
for k = 1 to 5

        Display SPACE (5-k) times.

        for j = 1 to 2k-1
                Display j

        Go to the next line.
```

The complete program is shown in Example 5.6. We see that the program is almost the same as the previous one. The only change is in line number 13 that displays the value of the variable "j" instead of X.

```
#include <stdio.h>

int main()
{
        int i, j, k;

        for (k = 1; k <=5; k++)
        {
                for(i = 1; i<=(5-k); i++)
                {
                        printf(" ");
                }

                for(j = 1; j <= (2*k - 1); j++)
                {
                        printf("%d ", j);
                }

                printf("\n");
        }

        return 0;
}
```

Example 5.7: A C program to display a pattern.

Exercise :

1. Write C programs to display the following patterns using nested loop construct.

a.
```
1 2 3
1 2 3
1 2 3
1 2 3
1 2 3
```

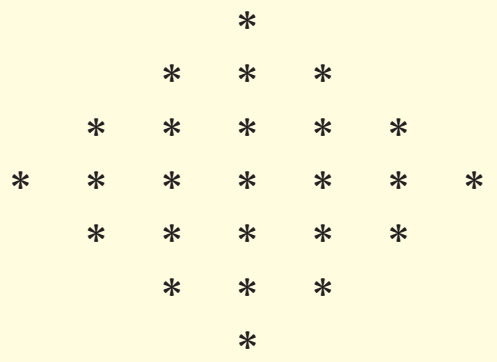
b.
```
1 2 1
1 2 1
1 2 1
1 2 1
1 2 1
```

c.
```
4 3 2 1
4 3 2 1
4 3 2 1
4 3 2 1
4 3 2 1
```

d.
```
    2
  2 3 4
2 3 4 5 6
```

e.
```
    1
  1 2 1
1 2 3 2 1
```

f.



g.

```
x                       x
x   x                   x   x
x   x   x               x   x   x
x   x   x   x           x   x   x   x
x   x   x   x   x       x   x   x   x   x
```

2. Modify the solution of question no. 1 to accept the number of lines as the input. The program should make the display pattern accordingly (Hint: write separate programs).

3. Extend the programs of Example 5.6 and Example 5.7 to make it dynamic by accepting the number of lines as an input from the keyboard.

4. what is a nested loop? Why do one use nested loops in our programs?

5. Do we need to use same type of loops as outer and inner loops? Justify your answer with some code segments.

6. Can we put a third loop inside the inner loop of a nested loop constract? Write a C program to justify your answer.

CHAPTER 6

Arrays in C

In this chapter

In the last two chapters, we learned about loops. In this chapter, we will use loops to access different variables that are stored in contiguous memory locations in our computer. We will also write programs to solve many interesting problems.

6.1 MOTIVATION

Let us recall the program for finding the summation of a series of numbers. We can refer to the C program of Example 4.7 as shown below.

```
#include<stdio.h>
int main()
{
        int var, choice =1;
        int sum = 0;
        do
        {
                printf("\n Enter a number: ");
                scanf("%d", &var);
                printf("\n You entered %d", var);
                sum = sum + var;
                printf("\n Do you want to stop? Enter 0, otherwise press any integer: ");
                scanf("%d", &choice);
        }
        while (choice != 0);
        printf("\n Summation of the numbers entered till now is: %d", sum);
        return 0;
}
```

In this program, we used a single variable "var" to store the number we entered from the keyboard. Thus everytime we enter a new number from the keyboard, the old one is lost. The variable is declared in line number 4 and the value is stored in it in line number 9.

Now, let us think for a while that we require the numbers later on. Suppose, we need to see the largest among these numbers. We may also need those for any other purpose.

The most natural way to handle this situation is to store these numbers in separate variables and then use them the way we want. We may declare the numbers as follows.

```
int num1, num2, num3, num4, num5, num6, num7, num8, num9;
```

Imagine that we need to use 100 such numbers. Shall we declare 100 integer variables?

Answer is No! We have a much better way to handle this situation. Let us study the concept of "array" in this chapter that tackles this issue.

6.2 INTRODUCTION

Array is a collection of similar data items where elements are stored in contiguous memory locations in our computer. Each data item of an array is typically called an **element** and each element has a specific memory location and this can be accessed by its relative position in the array. This relative position can also be referred as the **index** of the element. This can be shown using Figure 6.1.

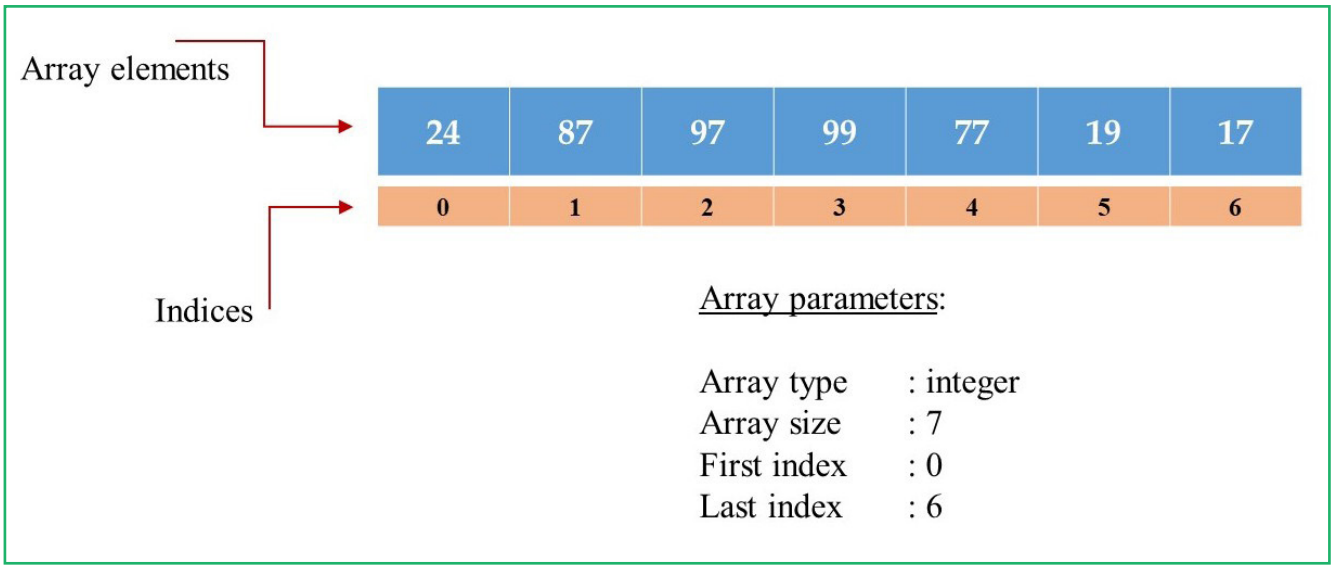


Figure 6.1: Illustration of an array.

The figure shows an integer type of array, meaning that the array is a collection of all integer type of elements. We may also say that the array can store only integer type of data. The array shown in the figure stores integers 24, 87, 97, 99, 77, 19 and 17.

(i) The size of an array in a number that says how many elements we can store in it at a time (maximum)
(ii) This is also called the capacity of the array.
(iii) However, it may not store any element at all.

The elements in an array are stored one after another and they can be accessed by specifying their relative positions or indices. The index of the first element in an array is 0 by default. The index of the second element is 1 and so on. The index of the last element is the size of the array minus 1.

In Figure 6.1, the size of the array is 7. This at a time means we can not store more than 7 elements in it The first element in the array is 24 and its index is 0. The last element in the array is 17 and its index is 7-1 = 6.

6.3 ARRAY DECLARATION IN C

Every programming language supports arrays. The declaration and other syntax may vary a bit from one language to another. In C programming language, the array of Figure 6.1 can be declared as follows.

```
int num[7];
```

Here, **num** is the name of the array. This is also called the **base address** of the array. The declaration states that "num" is an integer type of array that can store seven elements at max. Base address is the address of the first element of an array. we will learn more about this in Chapter-8.

If we want to declare by assigning initial values to the elements, we may use the following declaration.

```
int num[7] = {24, 87, 97, 99, 77, 19, 17};
```

If we want to declare an array of size 17 that can store character type data, we can use the following statement.

```
char sate[17];
```

Here, the name of the array is "state" and every element of the array can store any one character. As a whole, the whole array can store seventeen characters.

Whenever we want to declare a character type array and assign some initial values, we may declare in the following way.

```
char state[5] = {'A', 'S', 'S', 'A', 'M'};
```

Array in C programming can be declared in other ways as well. Let us not discuss this now.

Quick activity : Declare a character type array and assign your first name in it.

6.4 ACCESSING ARRAY ELEMENTS IN C

Till now we have seen how to declare an array in the C programming language. We will now see how we can access each element of an array - how to take input, how to display etc.

We have already seen that each element in an array has a specific location or index. We can use this in order to access a particular element. Figure 6.2 shows this clearly.

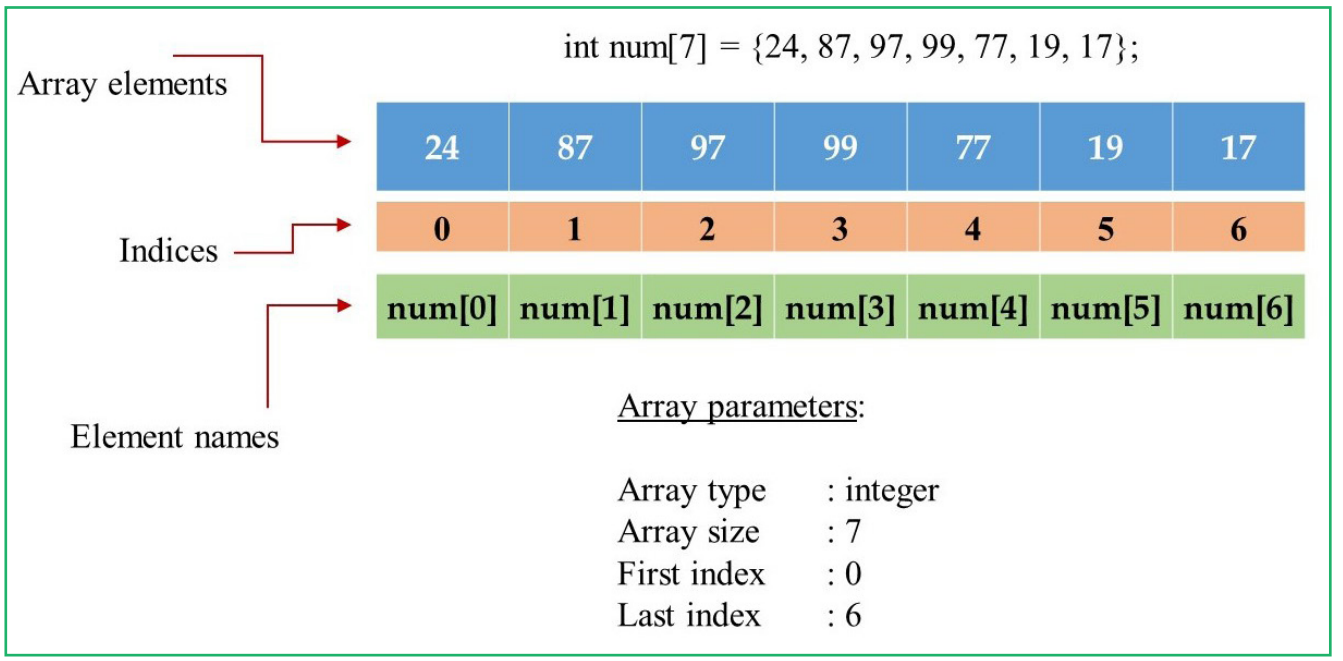


Figure 6.2: Illustration of an array num[7].

First element in the array can be referred as num[0]. The second one can be referred as num[1] and so on. In general, any individual element can be referred as num[id], where num is the name of the array and id is the index of that element in the array.

If we want to access the kth element in an array, we will use num[k-1] where num is the name of the array. kth element is present at index k-1.

Each element can be treated as a single variable and we can use it the way we treat a variable - to take input from the keyboard, to display it or to use it for any operation.

In the example considered in Figure 6.2, we have taken an integer type of array. Thus the following code segment will display the 3rd element of the array.

```
printf("%d", num[2]);
```

If we want to display all the elements of the array one by one, we can use a loop in the following way.

```
for (int index = 0; index < 7; index ++)
    printf("%d", num[index]);
```

The value of the variable index will go from 0 to 6 as the loop iterates. As a result, the printf() statement displays num[0], num[1], . . . num[6].

A complete C program is shown in Example 6.1. This program declares two arrays: num and state. The first one is an integer array and the second one is a char array. Both the arrays are declared with some initial values in their elements.

```
#include<stdio.h>
int main()
{
    int num[7] = {2,3,4,5,6,7,8};
    printf("\n Initial values of the first array elements are: ");
    for(int index=0; index<7; index++)
        printf("%d ", num[index]);
    char state[5] = {'A', 'S', 'S', 'A', 'M'};
    printf("\n Initial values of the second array elements are: ");
    for(int index=0; index<5; index++)
        printf("%c ", state[index]);
    return 0;
}
```

Example 6.1 A C program to declare and display array elements.

The first for loop was used to display the elements of the first array (line number 6 and 7). We used another for loop to display the elements of the second array (line number 9 and 10). As the elements of the first array are of type integer, we used %d format specifier to display. But the elements of the second array are of type character. So we used %c format specifier (line number 10).

In the above code segments, we have learned another syntax of the C language. That is, we can declare a variable in the initialization portion of the for loop. Then we can use that variable within that loop. We did this in case of the variable **index** in the code segments.

If we want to take input to the array from the keyboard, we can apply a similar strategy. For taking input to one particular element in the array (say, the 3rd element), we can use the following code segment.

```
scanf("%d", num[2]);
```

In case we wish to take input in the whole array, we scan the elements one by one using a loop. The following code segment can do this job.

```
for (int index = 0; index < 7; index ++)
    scanf("%d", &num[index]);
```

```
#include<stdio.h>

int main()
{
    int num[7] = {2,3,4,5,6,7,8};

    printf("\n Initial values of the array elements are: ");
    for(int i=0; i<7; i++)
        printf("%d ", num[i]);

    printf("\n Taking fresh input to the array elements: ");

    for(int i=0; i<7; i++)
    {
        printf( "\n Enter element %d: ", (i+1) );
        scanf("%d", &num[i]);
    }

    printf("\n New array elements are: ");
    for(int i=0; i<7; i++)
        printf("%d ", num[i]);

    return 0;

}
```

Example 6.2 A C program to take input to an array from the keyboard.

Example 6.2 shows a complete C program that displays the initial values of the array elements (line number 6 and 7), then takes input to all the array elements (line numbers 8 to 12) and then displays the new values (line number 14 and 15).

6.5 SOLVING PROBLEMS USING ARRAY

In this section, we will solve some of the interesting and commonly used problems that use array.

6.5.1 Finding summation of a series of numbers.

Let us now revisit the problem of finding the summation of a series of numbers as stated in Section 6.1. We will use an array to store the numbers and then we will find the summation of those numbers.

We have already learned in the previous section about scanning numbers from the keyboard into an array. We use the same strategy for this purpose. Let us see this step by step with corresponding code segments.

Step 1: Declare an integer type array with sufficient capacity.

```
int number[20];
```

This array can store upto 20 integers.

Step 2: Taking input to the array elements.

This step can be performed with two tasks.

Task 2.1: Asking users to enter the total number of elements in the series. This has to be less or equal to the capacity of the array. We store this in a variable “totalEle”.

```
printf("\n Enter total number of elements in the series: ");
scanf("%d", &totalEle);
```

Task 2.2: Asking users to enter the numbers and storing them in the array. We use a for loop to access the elements. Here we use the variable “totalEle” for indexing. The index of the array elements goes from 0 to totalEle -1.

```
for(int i=0; i<totalEle; i++)
{
        printf( "\n Enter element %d: ", (i+1) );
        scanf("%d", &number[i]);
}
```

Step 3: Calculating the summation of the array elements.

```
for(int i=0; i<totalEle; i++)
{
        sum = sum + number[i];
}
```

We calculate the summation by visiting every element one by one. At each iteration of the loop, we add the element to the previously calculated summation.

Step 4: Declaration of the result.

```
printf("\n Summation is: %d", sum);
```

We do this by displaying the value of the variable "sum". Example 6.3 shows the complete C program.

```
#include<stdio.h>

int main()
{
    int number[20];
    int totalEle, sum=0;

    printf("\n Enter total number of elements in the series: ");
    scanf("%d", &totalEle);
    printf("\n Taking input to the array elements: ");
    for(int i=0; i<totalEle; i++)
    {
```

```
        printf( "\n Enter element %d: ", (i+1) );
        scanf("%d", &number[i]);
    }
    for(int i=0; i<totalEle; i++)
    {
        sum = sum + number[i];
    }
    printf("\n Summation is: %d", sum);
    return 0;
}
```

Example 6.3 A C program to find the summation of a series of numbers using an array.

Quick Activity : Modify the above C program to calculate the average of the numbers.

6.5.2 Finding the largest among a series of numbers.

Similar to the previous problem, we will use an array to store the elements and then we will apply the logic to find the largest among them.

Thus steps 1 and 2 will be the same as the previous problem. The code segments are presented below.

```
int number[20];

printf("\n Enter total number of elements in the series: ");
scanf("%d", &totalEle);

for(int i=0; i<totalEle; i++)
{
    printf( "\n Enter element %d: ", (i+1) );
    scanf("%d", &number[i]);
}
```

Step 3: Finding the largest element

In order to find the largest number, we will apply a logic that we have already learned in Class 9. Let us present the flowchart below.

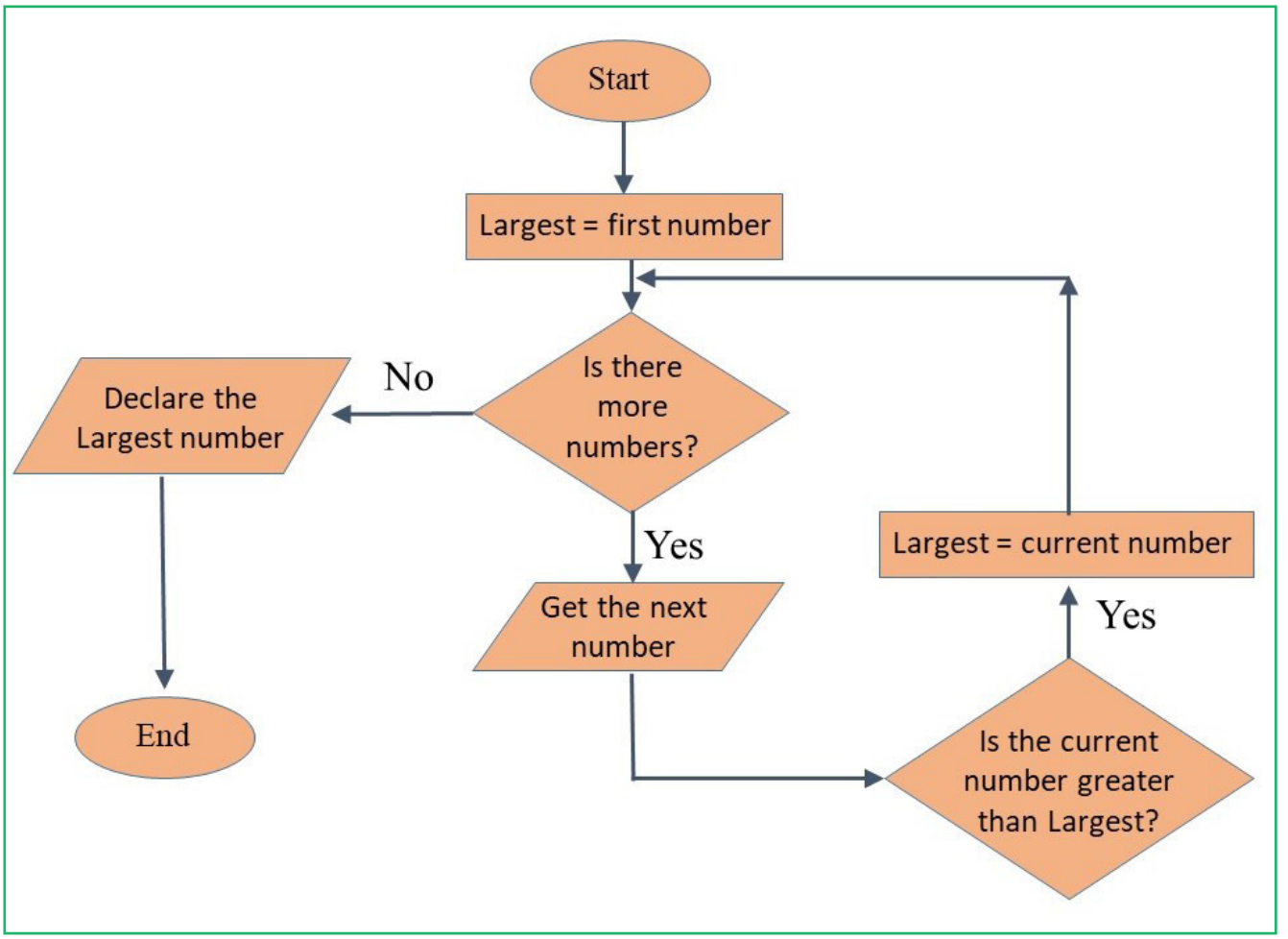


Figure 6.3: Flowchart for finding the largest number from a series of numbers.

The code segment for the above step is presented below.

```
int largest = number[0];

for(int i=1; i<totalEle; i++)
{
        if (number[i] > largest)
        largest = number[i];
}
```

Step 4: Declaring the result

This step is similar to the Step 4 of the previous problem.

The complete C program for finding the largest number among a series of numbers is shown in Example 6.4.

```
#include<stdio.h>

int main()
{
        int number[20];
        int totalEle;

        printf("\n Enter total number of elements in the series: ");
        scanf("%d", &totalEle);
        printf("\n Taking input to the array elements: ");
        for(int i=0; i<totalEle; i++)
        {
                printf( "\n Enter element %d: ", (i+1) );
                scanf("%d", &number[i]);
        }
        int largest = number[0];
        for(int i=1; i<totalEle; i++)
        {
                        if  ( number[i] > largest )
                                largest = number[i];

        }
        printf("\n Largest element is: %d", largest);
        return 0;
}
```

Example 6.4 A C program to find the largest number among a series of numbers using an array.

6.5.3 Finding all the even numbers from a series of numbers.

We first need to store the elements into an array and thus Steps 1 and 2 remain the same. We need to put the logic to find the even numbers in Step 3.

We will run through the array elements and for every element, we will check whether the element is divisible by 2 or not. If it is divisible by 2, it is an even number. The pseudocode for this step is presented below.

```
for(int i=0; i<totalEle; i++)
{
        if ( number[i] %2 == 0 )
                printf("\n %d is an even number", number[i]);
}
```

In the above code segment, we check the divisibility with the help of a remainder operator (%). When we divide a number by 2 and the remainder is zero, it is an even number. Here, double equal to (==) operator is used for the comparison.

The complete C program is shown using Example 6.5.

```
#include<stdio.h>

int main()
{
        int number[20];
        int totalEle;

        printf("\n Enter total number of elements in the series: ");
        scanf("%d", &totalEle);
        printf("\n Taking input to the array elements: ");
        for(int i=0; i<totalEle; i++)
        {
                printf( "\n Enter element %d: ", (i+1) );
                scanf("%d", &number[i]);
        }

        for(int i=0; i<totalEle; i++)
        {
                        if ( number[i] %2 == 0 )
                                printf("\n %d is an even number", number[i]);

        }
        return 0;
}
```

Example 6.5 A C program to find all the even numbers in a series of numbers.

Quick activity : Modify the above program to display the positions (indices) along with the numbers.

6.5.4 Counting all the even numbers from a series of numbers.

In the previous program, we ran through the array elements and displayed the ones that were even. Now, we need to count all such numbers.

Logic for this problem is this: We will initialize a counter to zero. Whenever we see an even number in the array, we increment the counter value. Finally, the value of the counter will give the number of even numbers in the array. We can demonstrate this logic using Figure 6.4.

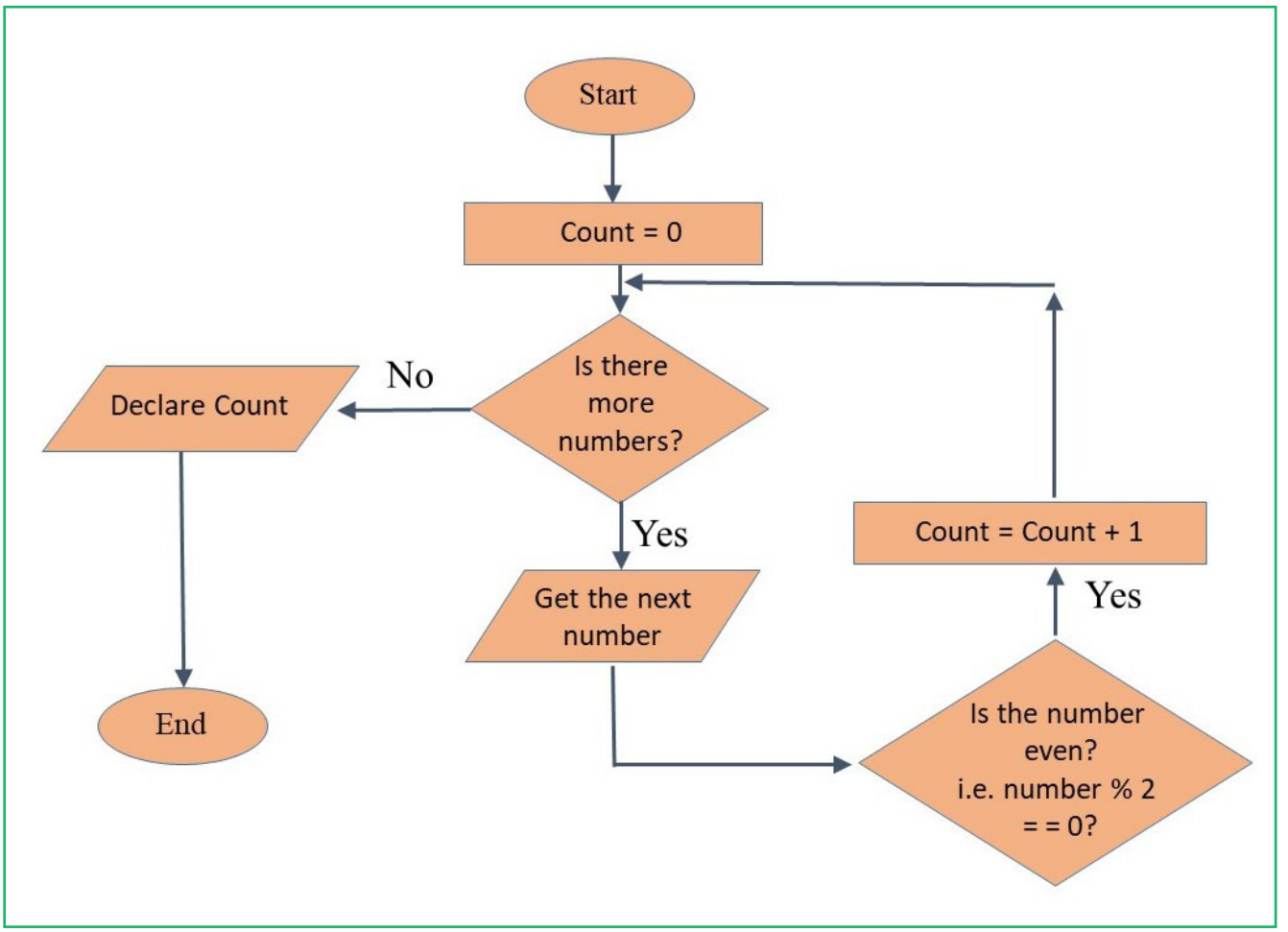


Figure 6.4: Flowchart for calculating total number of even numbers from a series of numbers.

We may also represent our logic using the below figure that uses the array notations. In Figure 6.5, totEle represents the total number of numbers stored in the array and number is the name of the array.

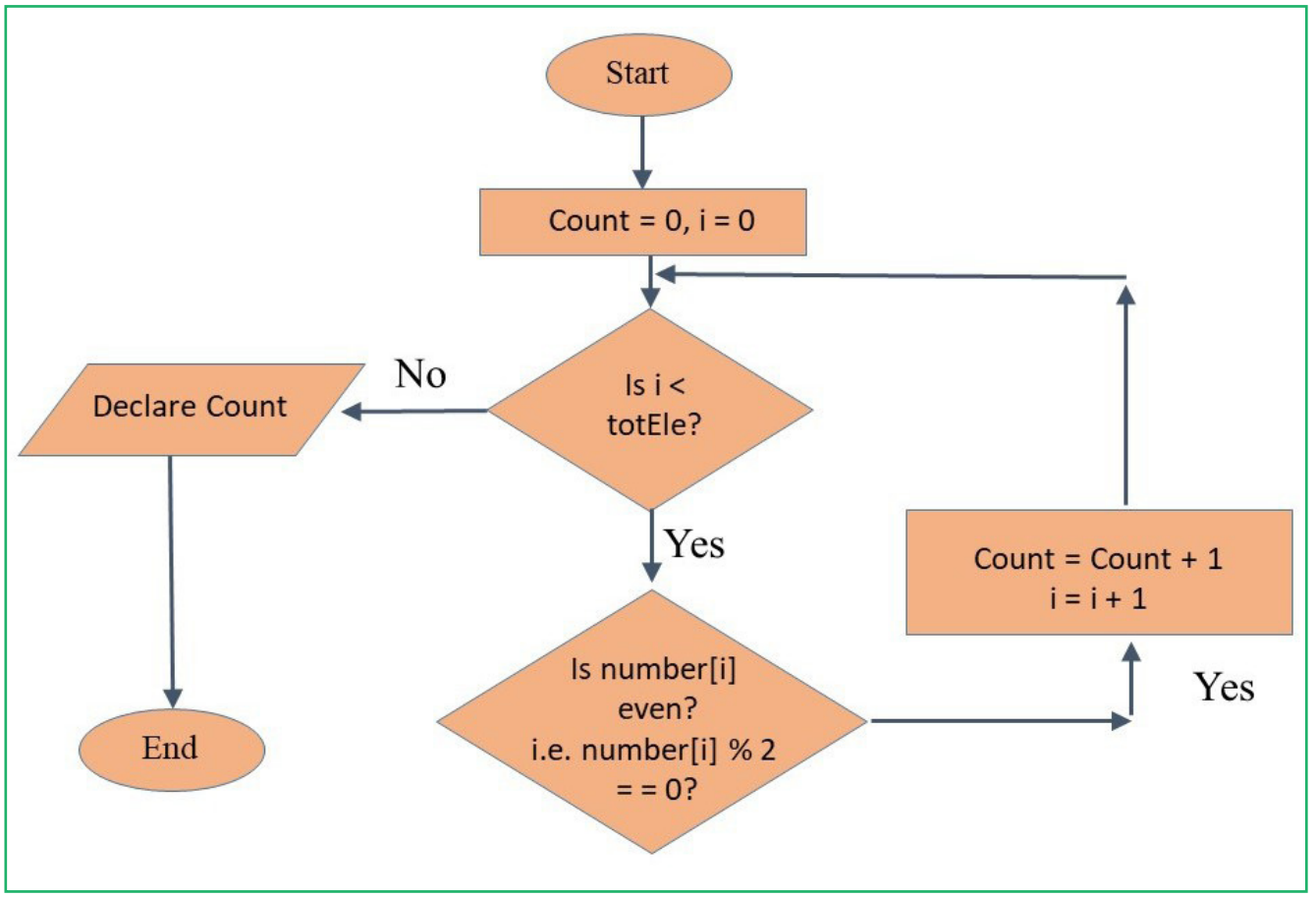


Figure 6.5: Another flowchart for finding the total number of even numbers from a series of numbers.

Let us write the code segment for this portion below.

```
count = 0;

for(int i=0; i<totalEle; i++)
{
        if ( number[i] %2 == 0 )
        {
                printf("\n %d is an even number", number[i]);
                count = count + 1;
        }
}

printf("\n Total number of even numbers: %d", count);
```

The complete C program for this problem is presented using Example 6.6.

```
#include<stdio.h>

int main()
{
        int number[20];
        int totalEle, count;

        printf("\n Enter total number of elements in the series: ");
        scanf("%d", &totalEle);
        printf("\n Taking input to the array elements: ");
        for(int i=0; i<totalEle; i++)
        {
                printf( "\n Enter element %d: ", (i+1) );
                scanf("%d", &number[i]);
        }
        count =0;
        for(int i=0; i<totalEle; i++)
        {
                if ( number[i] %2 == 0 )
                {
                        printf("\n %d is an even number", number[i]);
                        count = count + 1;
                }


        }
        printf("\n Total number of even numbers: %d", count);
        return 0;
}
```

Example 6.6 A C program to calculate total number of even numbers in a series of numbers.

6.5.5 Searching an element in an array

Suppose, we are given a series of numbers and we store these numbers in an array. We are now given another number. We need to write a C program to check whether this number is present in the series of numbers or not. If the number is present, we need to report its index in the array.

We can apply the same strategy as before to store the numbers in the array. Now let us develope a logic for searching. The following flowchart shows the searching logic.

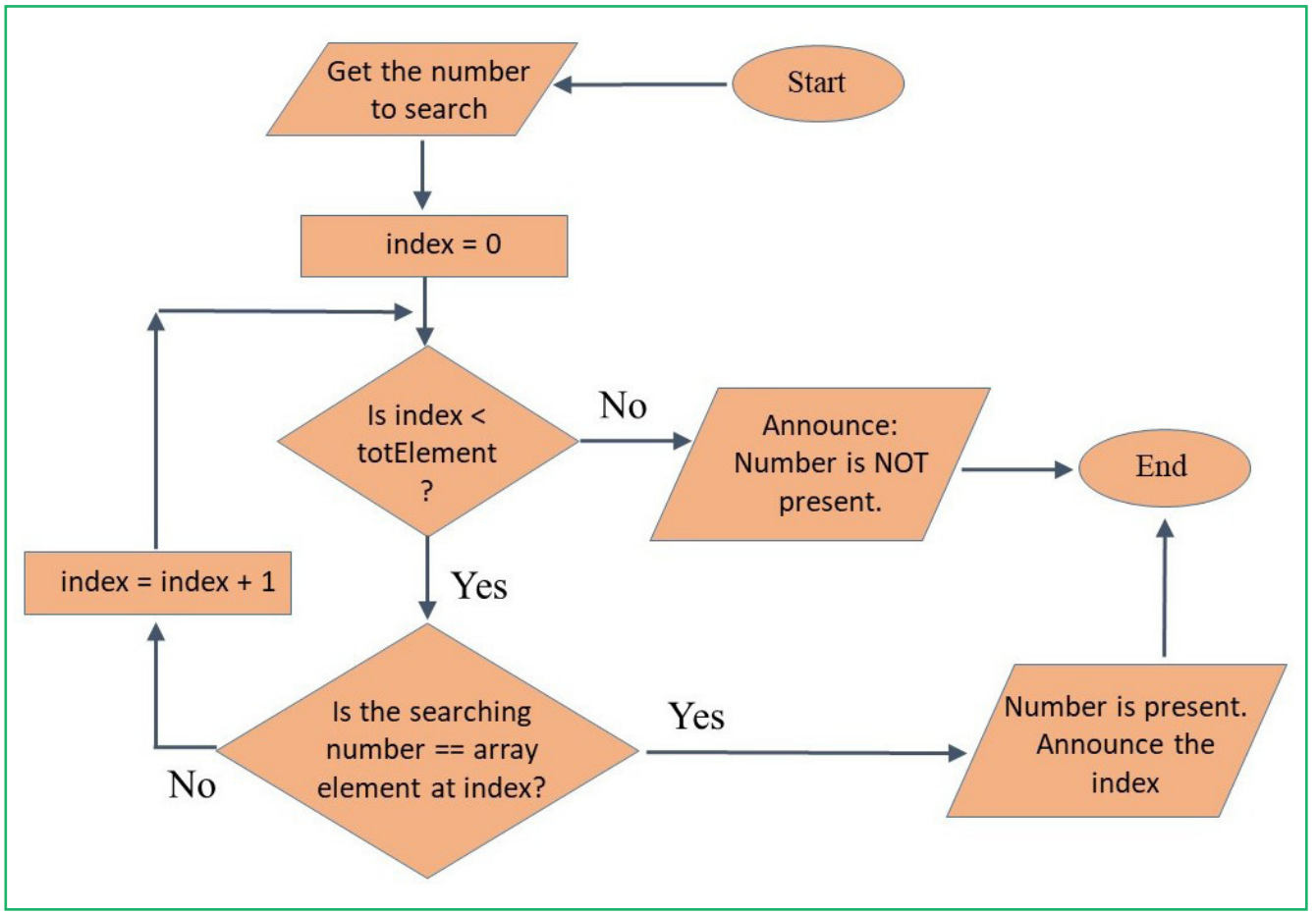


Figure 6.6: A flowchart for searching a number in an array.

The corresponding code segment is given below.

```
printf("\n Enter the number you want to search: ");
scanf("%d", &item);

int index = 0;

while ( index < totalEle )
{
        if ( number[index] = = item )
        {
                printf("\n The given number is found in the array at index %d", index);
                return 0;
        }
        index = index + 1;
}

        printf("\n Given number is NOT present");
```

We have used a while loop in the code segment. You may use any other loop as well. The first two lines in the code segment are responsible for getting the number for searching from the keyboard. This number is stored in a variable “item”.

We initialize the index to zero and the loop runs as long as it is less than totalEle that indicates the total number of elements present in the array.

As soon as the item and the array element matches (line number 6 in the code segment), we announce that the item is present and we terminate the program by returning an integer **(return 0).**

Otherwise, we go to the next element in the array by incrementing the index. When all the elements in the array are visited, we declare that the item is not present. The complete program is shown in Example 6.7.

```
#include<stdio.h>

int main()
{
        int number[20];
        int totalEle, item;

        printf("\n Enter total number of elements in the series: ");
        scanf("%d", &totalEle);
        printf("\n Taking input to the array elements: ");
        for(int i=0; i<totalEle; i++)
        {
                printf( "\n Enter element %d: ", (i+1) );
                scanf("%d", &number[i]);
}
printf("\n Enter the number you want to search: ");
scanf("%d", &item);

int index = 0;
while ( index < totalEle )
{
                if (number[index] = = item )
                {
                        printf("\n The given number is found in the array at index %d", index);
                        return 0;
                }
                index = index + 1;
        }
        printf("\n Given number is NOT present");
        return 0;

}
```

Example 6.7 A C program to search a number from a series of numbers.

Quick activity : Run the above program with inputs such that the searching element appears more than once in the series of numbers. For example, take array elements as {4, 5, 6, 9, 8, 7, 4, 5} and search for 5.

Have you observed anything important in the above activity?

When we search for the number 5 in the series of numbers, we see that the number 5 is present at indices 1 and 7. But the program terminates displaying that it is present at index 1.

How can we handle this situation? Whenever a number is present more than once, we want to display all the indices.

If we observe the above program carefully, we see that this was happening because we terminated the program by returning 0 whenever we got a match (line number 9 in the code segment).

Naturally, we can not end the program here, even if we get a match! We need to report that it is a match and we need to go ahead to look for the next match if there is any.

Thus the code segment becomes as follows.

```
printf("\n Enter the number you want to search: ");
scanf("%d", &item);

int index = 0;

while ( index < totalEle )
{
        if ( number[index] = = item )
        {
                printf("\n The given number is found in the array at index %d", index);

        }
        index = index + 1;
}

printf("\n Given number is NOT present");
```

Quick activity : Replace this code segment and rerun the program with the previous input. That is, take array elements as {4, 5, 6, 9, 8, 7, 4, 5} and search for 5.

What have you observed now?

The program rightly displays all the appearances of a given number. So we solved the above issue.

But the program also displays the line “Given number is NOT present”. This is wrong - we do not want this line to be displayed.

We can solve this with the help of another variable - flag.

The logic is simple! While visiting the array elements, if we find a match we take a note of it by setting a flag variable. After we finish visiting all the elements in the array, if the flag is still in reset condition, we can safely say that we have not found any match and we can declare that the item is not present.

The code segment is presented below.

```
printf("\n Enter the number you want to search: ");
scanf("%d", &item);

int index = 0, flag = 0;

while ( index < totalEle )
{
        if ( number[index] = = item )
        {
                printf("\n The given number is found in the array at index %d", index);
                flag = 1;

        }
        index = index + 1;
}

if ( flag = = 0)
        printf("\n Given number is NOT present");
```

The complete C program is presented using Example 6.8. This program reports all the appearances of a given number in the series of numbers.

```
#include<stdio.h>

int main()
{
        int number[20];
        int totalEle, item;

        printf("\n Enter total number of elements in the series: ");
        scanf("%d", &totalEle);
        printf("\n Taking input to the array elements: ");
        for(int i=0; i<totalEle; i++)
        {
                printf( "\n Enter element %d: ", (i+1) );
                scanf("%d", &number[i]);
        }
        printf("\n Enter the number you want to search: ");
        scanf("%d", &item);

        int index = 0, flag = 0;
        while ( index < totalEle )
        {
                if (number[index] = = item )
                {
                        printf("\n The given number is found in the array at index %d", index);
                        flag = 1;
                }
                index = index + 1;
        }
        if ( flag = = 0 )
                printf("\n Given number is NOT present");
        return 0;
}
```

Example 6.8 A C program to search a number from a series of numbers.

Quick Activity : Write and execute the above program using a for loop in place of the while loop.

6.5.6.Displaying the array elements in reverse order

In all the programs above, we have traversed the array elements from lower index to higher index (from 0 to SIZE-1). In this problem, we are asked to display the elements in reverse order.

G	U	W	H	A	T	I

If above is the array, the program should display I T A H A W U G.

To do so, we can start with the highest index and we can move toward the lower index by decrementing the index value. We continue to display one element at a time using a printf() statement. We can do this in each iteration of a loop.

The following code segment can do this.

```
int index = SIZE - 1;

printf("Elements in the reverse order are: \n");

while ( index >= 0)
{
        printf("%c ", char_array[index]);

        index = index - 1;
}
```

Here, char_array is the name of a character type array which has SIZE number of elements in it. The indices of the array elements start at 0 and end at SIZE-1. As we are dealing with a character type array, we used %c in the printf() statement to display the elements.

A complete C program demonstrating the same is presented in Example 6.9. In the program, the variable index is initialized with 7 because we have eight elements in the array and 7 is the index of the last element.

```
#include<stdio.h>

int main()
{
        char char_array [ ] = {'G', 'U', 'W', 'A', 'H', 'A', 'T', 'I'};
```

```
        printf("Elements in the reverse order are: \n");

        int index = 7;

        while(index >= 0)
        {
            printf("%c ", char_array[index]);
            index = index - 1;
        }
        return 0;
}
```

Example 6.9: A C program to display array elements in reverse order.

Quick Activity : Write a similar C program for an integer type of array where the array elements are taken from the user as input and display them in reverse order.

6.5.7. Displaying the odd position elements of an array

Sometimes, we do not need to display all the elements. We may need to display only the elements that are stored either in even or odd positions.

Suppose, we need to display only the odd positioned elements. For the given array as shown below, the program should display **G W H T**.

G	U	W	H	A	T	I

Figure 6.7 demonstrates this nicely.

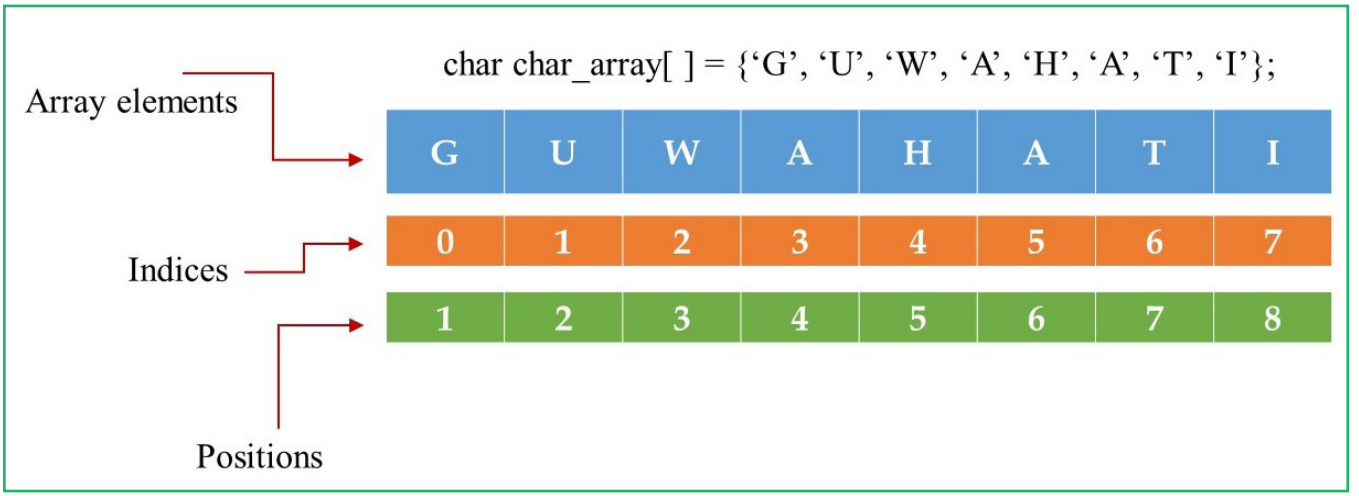


Figure 6.7: Array elements with indices and positions.

In that case we need to display the elements whose indices are 0, 2, 4, and so on. Thus, we can start at index 0 and at each iteration of the loop, we can increment the index by 2.

Following is a code segment for this. Here, 8 is the number of elements in the array.

```
int index = 0;

while(index < 8)
{
        printf("%c ", char_array[index]);

        index = index + 2;
}
```

We present the complete program in Example 6.10.

```
#include<stdio.h>

int main()
{
        char char_array [ ] = {'G', 'U', 'W', 'A', 'H', 'A', 'T', 'I'};

        printf("Elements in the odd position are: \n");

        int index = 0;

        while(index < 8)
        {
                printf("%c ", char_array[index]);
                index = index + 2;
        }
        return 0;
}
```

Example 6.10: A C program to display the elements of the odd positions.

Quick Activity : Write a similar C program for an integer type of array where the array elements are taken from the user as input and display only the even positioned elements.

6.5.7. Replacing an array element - index is given

Suppose, we have an array and we have stored some elements into it. Now, we want to replace an existing element with a new one.

We know that the elements in an array are stored in specific locations - identified by indices. If we want to place a new element replacing an existing one, we need to have the index or the location where we will place it. We may take this index as an input from the user.

We may also get the index by performing a search operation on the array. We see this in the next problem.

The following code segment does our job. The last statement updates the array element of indexed by *pos* with a number *item*. Here, we are taking both the number and the index as input from the keyboard.

```
printf("\n Enter the new number: ");
scanf("%d", &item);

printf("\n Enter the index of the new number: ");
scanf("%d", &pos);

number[pos] = item;
```

Example 6.11 shows the complete C program.

```
#include<stdio.h>

int main()
{
        int number[20];
        int totalEle, item, pos;

        printf("\n Enter total number of elements in the series: ");
        scanf("%d", &totalEle);

        printf("\n Taking input to the array elements: ");
        for(int i=0; i<totalEle; i++)
        {
```

```
            printf( "\n Enter element %d: ", (i+1) );
            scanf("%d", &number[i]);
        }

        printf("\n Now elements of the array are: ");

        for(int i=0; i<totalEle; i++)
        {
            printf("%d ", number[i]);
        }

        printf("\n Replacing elements... ");

        printf("\n Enter the new number: ");
        scanf("%d", &item);

        printf("\n Enter the index of the new number: ");
        scanf("%d", &pos);

        number[pos] = item;

        printf("\n Now elements of the array are: ");

        for(int i=0; i<totalEle; i++)
        {
            printf("%d ", number[i]);
        }
        return 0;
}
```

Example 6.11: A C program to replace an array element given by an index.

6.5.8. Replacing an array element - element is given

In the previous problem, for replacing an existing element with a new one, we were given the index where we were to place the new element. In this problem, we are not given the index. We are given the element that is to be replaced.

Hence, our first job is to find the index of the element that is to be replaced. For this, we go through the array element and we make a comparison.

When we get a match, we place the new element in that position using the index. The following code segment shows this. Line number 7 makes the comparison here.

```
printf("\n Enter the old number you want to replace: ");
scanf("%d", &old_num);

printf("\n Enter the new number: ");
scanf("%d", &item);

for(int index = 0; index <totalEle; index++)
{
        if(number[index] = = old_num)
        {
                printf("\n Existing number is found at index %d", index);

                number[index] = item;

                printf("\n Number is replaced");
        }
}
```

5	9	5	2	5	5	3	1

This will replace all the occurrences of the given element. For example, if the array looks like above and if we want to replace number 5 with 0, the following will be the status of the array.

0	3	0	0	2	0	9	0

```
#include<stdio.h>

#include<stdio.h>

int main()
{
        int number[20];
        int totalEle, item, old_num;

        printf("\n Enter total number of elements in the series: ");
```

```
        scanf("%d", &totalEle);

        printf("\n Taking input to the array elements: ");
        for(int i=0; i<totalEle; i++)
        {
                printf( "\n Enter element %d: ", (i+1) );
                scanf("%d", &number[i]);
        }
        printf("\n Now elements of the array are: ");
        for(int i=0; i<totalEle; i++)
        {
                printf("%d ", number[i]);
        }

        printf("\n Replacing elements... ");
        printf("\n Enter the old number you want to replace: ");
        scanf("%d", &old_num);

        printf("\n Enter the new number: ");
        scanf("%d", &item);

        for(int index = 0; index <totalEle; index++)
        {
                if(number[index] == old_num)
                {
                        printf("\n Existing number is found at index %d", index);
                        number[index] = item;
                        printf("\n Number is replaced");
                }
        }
        printf("\n Now elements of the array are: ");
        for(int i=0; i<totalEle; i++)
        {
                printf("%d ", number[i]);
        }
        return 0;
}
```

Example 6.12: A C program to replace all the occurrences of a number in an array.

6.5.9. Counting number of occurrences of an element

In this problem, we are given a series of elements that are stored in an array. Then we are given a single element and we need to find the number of times the given element occurs in the series. Let us understand this with an example.

7	8	3	3	4	8

In the above array, if we are given number 3, then the program should output 2 because the number 3 occurs 2 times in the array.

We can see that the problem is similar to "searching". So we will be applying a similar logic.

We will keep a counter and we will visit all the elements. While visiting an element, we will see whether the element matches with the given element (by performing a comparison operation). If it matches, we will increment the counter.

We will announce the counter value as we finish visiting all the elements in the array.

Following code segment does the job.

```
printf("\n Enter the element you want to count: ");
scanf("%d", &item);

int count = 0;

for(int index = 0; index <totalEle; index++)
{
        if(number[index] = = item)
        {
                count = count + 1;
        }
}

printf("\n %d is present %d number of times in the array", item, count);
```

Example 6.13 shows the complete C program. Here, we are taking an integer type array with capacity 20. We take the numbers as inputs from the users.

```
#include<stdio.h>

#include<stdio.h>

int main()
{
	int number[20];
	int totalEle, item;

	printf("\n Enter total number of elements in the series: ");
	scanf("%d", &totalEle);

	printf("\n Taking input to the array elements: ");
	for(int i=0; i<totalEle; i++)
	{
		printf( "\n Enter element %d: ", (i+1) );
		scanf("%d", &number[i]);
	}
	printf("\n Now elements of the array are: ");
	for(int i=0; i<totalEle; i++)
	{
		printf("%d ", number[i]);
	}

	printf("\n Enter the element you want to count: ");
	scanf("%d", &item);

	int count = 0;

	for(int index = 0; index <totalEle; index++)
	{
		if(number[index] == item)
		{
			count = count + 1;
		}
	}
	printf("\n %d is present %d number of times in the array", item, count);
	return 0;
}
```

Example 6.13: A C program to count number of occurrences of given in an array.

6.6 ARRAY AS STRING

In most of the programs above, we have taken the array of type int. In a few cases, we have taken char type arrays. A string is defined as an array of characters. The last character in a string is a special character '\0'. This is also called the NULL character in a string. The NULL character is to indicate the end of the valid characters of the array.

There is a dedicated header file in C called string.h that supports many library functions for strings. Let us see a simple C program in Example 6.14 that takes a string as input and displays that string. We used gets() function to take the input from the keyboard and puts() function to display it.

```
#include<stdio.h>
#include<string.h>

int main()
{
        char name[20];

        printf("\n Enter your name: ");
        gets(name);

        printf("\n Your name is: ");
        puts(name);
        return 0;
}
```

Example 6.14: A C program to handle a char array as string.

The string.h header file supports many library functions for performing operations on strings. We will see one such function strlen() in Example 6.15. The function returns the length of the string, that is the number of characters in the array. We use this in the for loop to go through the array elements and to display them one by one.

```
#include<stdio.h>
#include<string.h>

int main()
```

```
{
        char name[20];

        printf("\n Enter your name: ");
        gets(name);
        printf("\n Your name is: ");
        puts(name);

        printf("\n Displaying the string character wise: ");
        for(int index = 0; index < strlen(name); index++)
        {
                printf(" %c ",name[index]);
        }
        return 0;
}
```

Example 6.15: A C program to display the string characters.

6.7 DEMERITS OF ARRAYS

We have seen usages of arrays in solving several problems. In this section, let us see some of the demerits of arrays.

Though array has a huge usefulness, it has some limitations as well. Let us list them below.

1. Fixed size
2. Homogenous data
3. Contiguous storage

Fixed size: We have already seen that before using an array, we must declare it with a size. This is the capacity of the array. If some requirement arises (at run time)n to store more elements in that array, we cannot do so. For instance, in an integer array with size 10, we cannot store 11 integers.

Homogeneous data: We know that an array is a collection of similar types of data. We cannot store a float variable in an integer array.

Contiguous storage: The elements in an array are stored in contiguous memory locations - one after another. We cannot simply delete an element or add an element in an arbitrary location.

Figure 6.8 explains this with an example. Suppose, an array stores 'SEBAASSAM' and we want to put a space in between SEBA and ASSAM. Then, all the characters from indices 8, 7, 6, 5 and 4 have to be shifted toward the right side. This is a costly operation.

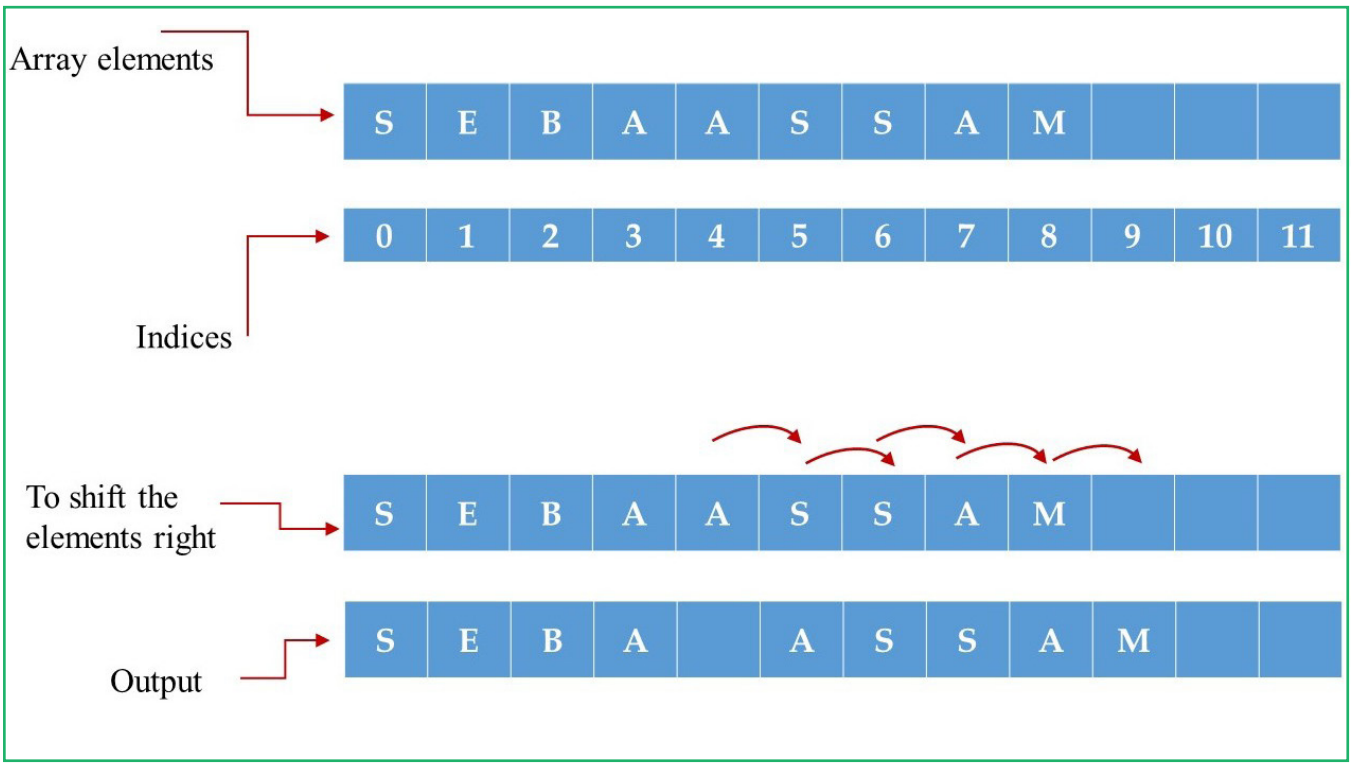
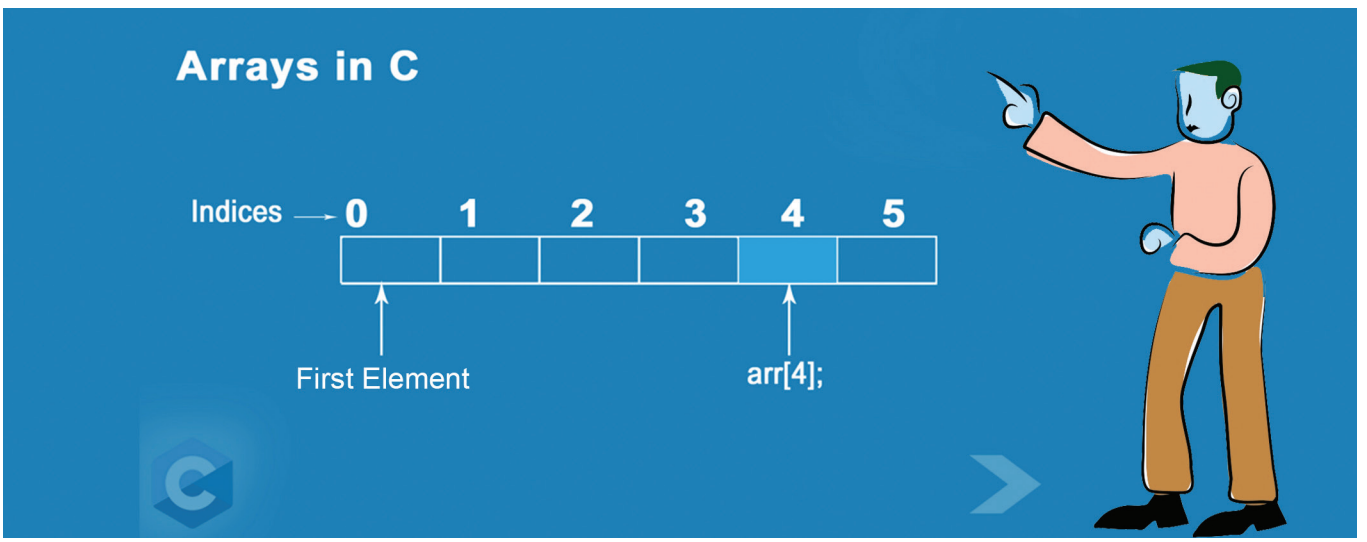


Figure 6.8: Inserting an element in an array.

In this chapter, we have learned about arrays and its usages along with some limitations. We have solved a good number of problems and written programs in C. Another set of problems are given in the Exercise. We also highly encourage the students to think of some variations of the problems and then try to develop solution strategies for the same. Finally, they should try to write programs.



Exercise

1. Define array. Why do we use arrays in computer programs?

2. Can we store both integer and float types of data in a single array? Demonstrate this by writing a simple C program.

3. Write the indices of the first and last element of the following array declaration.

```
char city [7]={'S', 'I', 'L', 'C', 'H', 'A', 'R', };
```

4. Write a C program and declare an integer type array with capacity 7. Take input to the array from the keyboard. Display the 8th element of the array. Hint: display num[7] if num is the name of the array. Analyze the output.

5. Write a C program and declare an integer type array with 7 elements in it. Display the address of the individual elements in the array.

6. Write a C program and declare two integer type arrays, each with capacity 7. Take input only to the first array. Write a loop to copy the elements of the first array to the second one. Display the elements of the second array.

7. Write a strategy to find the summation of all the even numbers stored in an array. Write a C program for the same. If the array elements are {1, 2, 4, 3, 5, 6, 7, 7, 8}, the output of the program will be 20.

8. Write a strategy to find the summation of all the even positioned numbers stored in an array. Write a C program for the same. If the array elements are {1, 2, 4, 3, 5, 6, 7, 7, 8}, the output of the program will be 18.

9. Write the logic to replace only the first occurrence of an element in an array. For example, if the array elements are {1, 2, 3, 4, 5, 1, 2, 3} and we want to replace element 3 by 0, the array content becomes {1, 2, 0, 4, 5, 1, 2, 3}. Write a complete C program incorporating the logic.

10. Write the logic to replace only the last occurrence of an element in an array. For example, if the array elements are {1, 2, 3, 4, 5, 1, 2, 3} and we want to replace element 3 by 0, the array content becomes {1, 2, 3, 4, 5, 1, 2, 0}. Write a complete C program incorporating the logic.

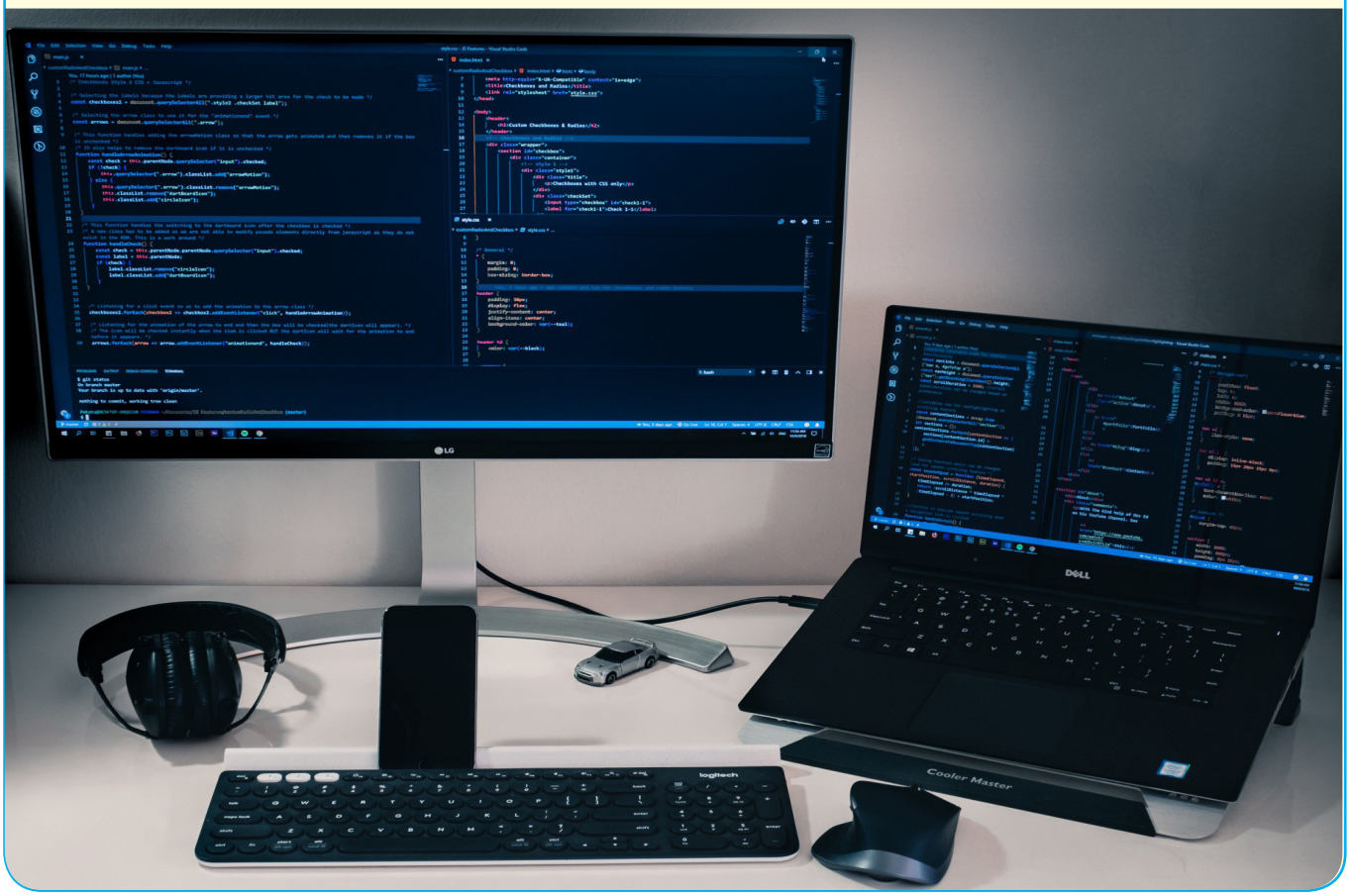
11. Write a C program to replace all the even positioned elements in an integer array by 0. For example, if the array elements are {1, 2, 3, 9, 5, 5, 7, 1, 9}, it becomes {1, 0, 3, 0, 5, 0, 7, 0, 9}.

12. Write a strategy to replace all the odd numbers in an integer array by 0. For example, if the array elements are {1, 2, 3, 9, 5, 5, 7, 1, 9}, it becomes {0, 2, 0, 0, 0, 0, 0, 0, 0}. Write a C program for the same.

13. Write a C program to store your name and your mother's name in two different strings. Display them one after another.

14. Write a C program to declare 3 integer type arrays to store the marks of 10 students scored in 3 different subjects. Take another integer to store the average marks of the students. The average mark of a student is the average of the marks scored by the student in 3 subjects. This should be calculated and inserted by the program. Then you should display the average marks of the students. The program should also display "PASS" or "FAIL" along with the average as per the rule: PASS if average >= 45, FAIL otherwise.

15. Write any two limitations of arrays. Can you store your name and roll number in the same aray?



CHAPTER 7

Functions in C

In this chapter

We will learn an interesting and useful feature of the programming languages, called function. Function facilitater a better way of coding. It increaser the readablity of the program an well. In addition, we will learn some new syntax of C programing language here.

7.1 INTRODUCTION

We have seen that when we need to do many tasks in a program, the length of the program increases. Sometimes, it becomes difficult to read and understand that. There is an interesting construct in computer programming that handles this issue nicely - called **function**!

Moreover, if we need to perform a task many times, we typically need to write the code that many times. But if we create a function and put the statements there, we just need to write only once. In that case, everytime we need to perform the task, we can simply make a call to the function and we are done.

A function in computer programming is a group of statements that together perform a task. We can divide our code of a program into separate functions. How we divide the code among different functions is up to us, but logically the division is such that each function performs a specific task.

There are some common terminologies with respect to functions in C. Let us first understand these.

Function name : Every function has to be given a valid name. Any valid identifier except the keywords can be a function name in C.

Return type : Every function in C usually returns a value. The data type of the returning value becomes the return type of the function. For example, if we return an integer from a function, the return type of the function becomes int. In case a function does not return anything, its return type becomes **void**. A function can return only one item.

Function parameter : We can pass or send some data to a function when it is executed. Parameters in a function refer to these data. Parameters are like placeholders; they take some data along with them as input.

```
float calculate_interest ( int principal_amont, int year, float rate_of_interest);
```

The above code segment indicates a function whose name is "calculate_interest". The return type of the function is "float" and the function has three parameters - "principal_amount", "year" and "rate_of_interest".

Parameters of a function are also termed as arguments. A function in C may not accept any parameter. We call such functions as 0-argument functions.

7.2 COMPONENTS OF FUNCTION

There are three parts of a function in C.

1. Function declaration: A function must be declared first before its use. This is to tell the compiler about the function name, function parameters, and return type. We give an example of a function declaration below.

```
float calculate_interest ( int principal_amont, int year, float rate_of_interest);
```

2. Function call: This statement is responsible for the execution of a function. A function starts its execution when we make a call to it. A function can be called from anywhere in the program. The parameter list in function calling and function declaration must match. We must pass the same number of parameters as it is declared in the function declaration.

The following code statement makes a call to the function "calculate_interest" and passes three parameters. The result of the function is stored in a variable "interest".

```
float interest = calculate_interest ( principal_amont, year, rate_of_interest );
```

3. Function definition: This portion of a function contains the actual statements that are executed when the function is called. The last statement in a function definition is the return statement that we wish to return to the caller of the function.

The following code segment is the definition of a function calculate_interest(). The function calculates simple interest using the values of the parameters and returns it.

```
float calculate_interest (  int principal_amont, int year, float rate_of_intterest )
{
        float interest =  principal_amont * int year * rate_of_intterest / 100;

        return interest;
}
```

Figure 7.1 shows a simple C program that uses a function sum() to find the summation of two integers. The function accepts two integer parameters and returns another integer. In the program, main() is another function and it is calling a function sum() in line number 6. Thus main() can be called as a **caller function** and sum() can be called as a **callee function.**

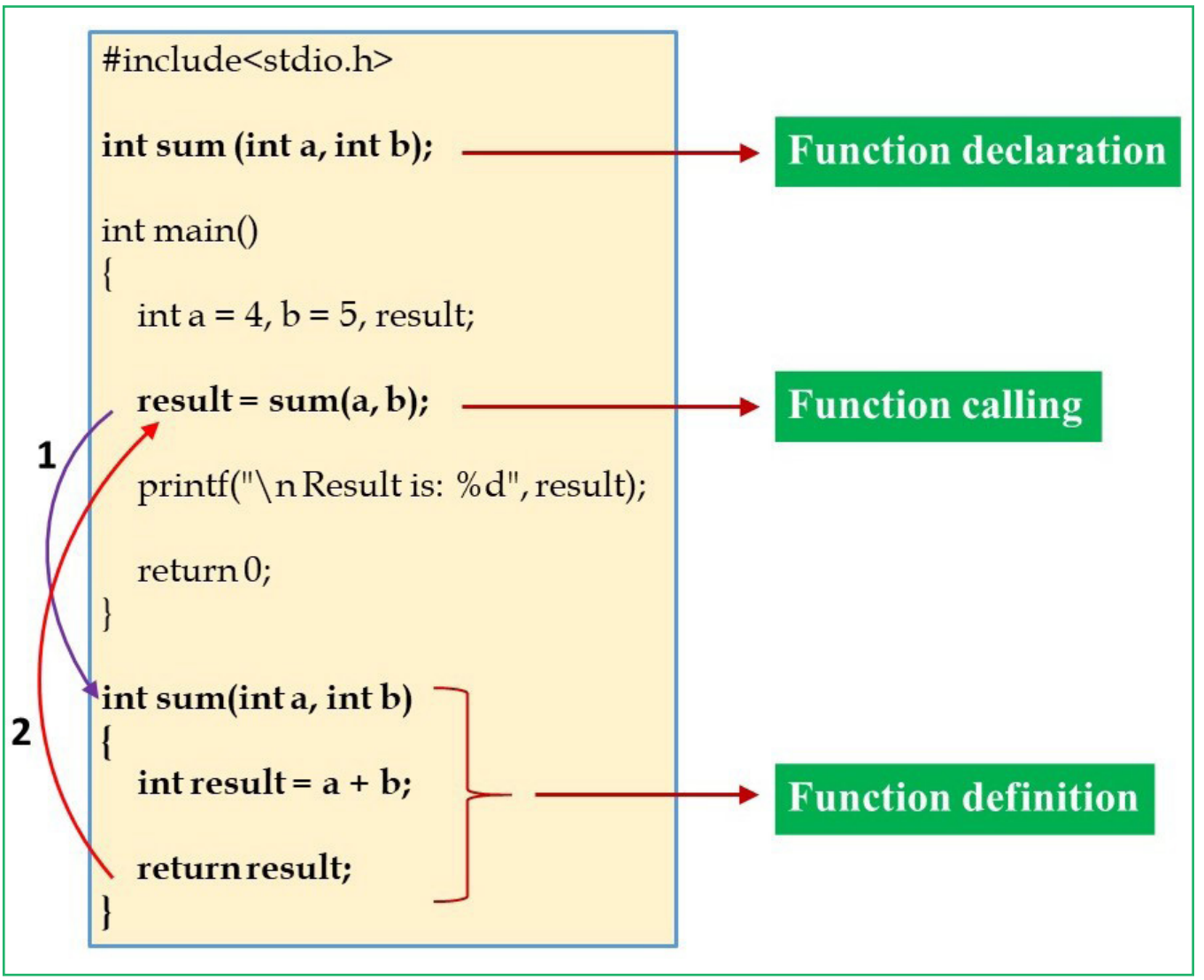


Figure 7.1: A C program showing a function.

Function declaration and definition can also be written together. We will see examples of this later in this chapter.

There exists only one declaration and definition of a function. But a function may be called any number of times. Everytime we call a function, the control goes from the caller to the definition of the callee. After executing all the statements in the callee function definition, the control goes back to the caller. These are shown using arcs 1 and 2 in Figure 7.1.

7.3 TYPES OF FUNCTION

There are two types of functions in C programming:

1. Library functions: These are the functions that are defined in the C header files. We use these functions while writing our programs. We have already used printf() and scanf() in our programs. There are more such as gets(), puts(), ceil(), floor() etc.

2. User-defined functions: These are the functions that are created by the programmer. The functions we discussed till now (claculate_interest(), sum()) are user-defined functions. These are called user-defined functions because the programmer provides/writes definitions of these functions. But the definitions of library functions are already written in the system.

7.4 SOLVING PROBLEMS USING FUNCTION

In this section, we will write a number of C programs using functions. The problems we consider here to solve are not new. We have already solved similar problems in the previous chapters. Thus the solution strategy or the logic remains the same. What is new here is the syntax of function - how we can solve those problems using functions.

Problem 1: Finding summation of two numbers

Let us start with a very simple program. Let us write a function to find the summation of two integers. Example 7.1 shows the code for this.

```
#include<stdio.h>

int sum (int a, int b);

int main()
{
        int a, b, result;
```

```
        printf("\n Enter the first number: ");
        scanf("%d", &a);

        printf("\n Enter the second number: ");
        scanf("%d", &b);

        result = sum(a, b);

        printf("\n Result is: %d", result);

        return 0;
}

int sum(int a, int b)
{
        int result;

        result  = a + b;

        return result;
}
```

Example 7.1 A C program using a function.

The main function first takes input in two integers a and b from the keyboard. Then it calls a function sum() for doing the calculation (line number 10). The control immediately goes to the definition of the function sum().

In the definition of the function, we declare another integer "result" and the result is assigned the value of a+b. Finally, we return the value of result from the function that carries the summation of a and b (last line of function defition).

As we return from the function, the control goes back to the place from where it made a call (line number 10). Then we display the value using a library function printf().

To check the program flow, we can write some additional printf() statements. Refer to the Example 7.2 that is the same program as Example 7.1 with some additional printf() statements.

```
#include<stdio.h>

int sum (int a, int b);

int main()
{
        int a, b, result;

        printf("\n Enter the first number: ");
        scanf("%d", &a);

        printf("\n Enter the second number: ");
        scanf("%d", &b);

        printf("\n Program control: just before making function call.");

        result = sum(a, b);

        printf("\n Program control: just after return from function call.");

        printf("\n Result is: %d", result);

        return 0;
}

int sum(int a, int b)
{
        printf("\n Program control: inside function definition.");
        int result;

        result  = a + b;

        return result;
}
```

Example 7.2 A C program using a function with extra printf().

As we execute the above program, we see the program control printf() are displayed the following output. We understand from the output that as we call a function, the control goes to the definition of it and then it comes back to the caller.

Program control: just before making function call.

Program control: inside function definition.

Program control: just after return from function call.

Quick activity : Modify the above program to do multiplication of two integers instead of addition.

Problem 2: Finding largest among three numbers.

We have not solved this problem in this book yet. So let us first discuss the solution strategy. Then we will use a function for the same.

We are given three numbers: a, b, and c. Let us first try to answer - when we can say "a" is the largest number. The number "a" is the largest if it is greater than b and it is greater than c. So we make the first comparison between a and b.

If a is greater than b, then we can compare a with c. If a is greater in both the cases, a is the largest. If a is not greater than c, then c happens to be the largest.

Otherwise (if a is not greater than b), the largest can be either b or c. So we can do a comparison between b and c. If b is greater than c, then b is the largest. Otherwise c happens to be the largest.

We present the logic using a flowchart as shown in Figure 7.2.

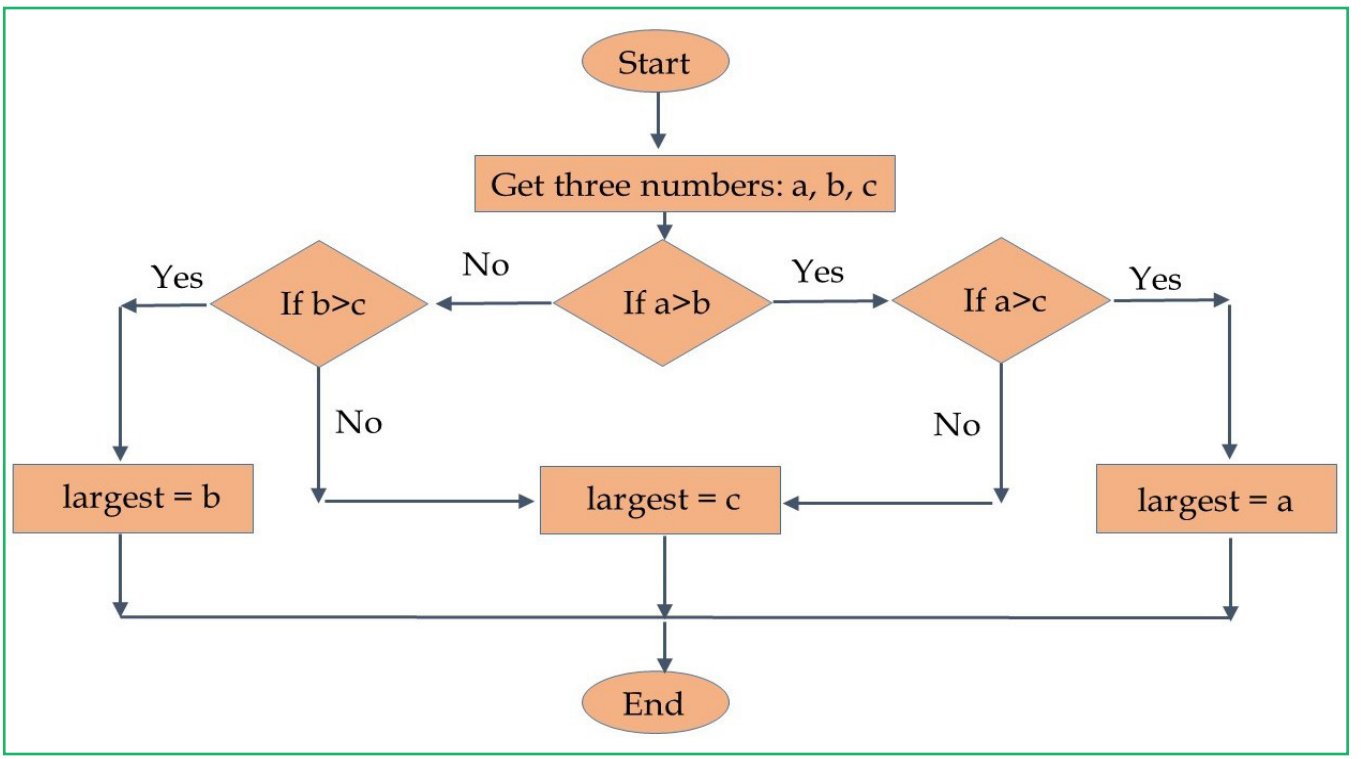


Figure 7.2: A flowchart for finding the largest among three numbers.

We now write a code segment for the above logic as shown below.

```
if ( a > b )
{
        if ( a  > c )
                largest = a;
        else
                largest = c;
}

else
{
        if ( b > c )
                largest = b;
        else
                largest = c;
}
```

The next job is to write a function for this. We will write the above code segment inside a function definition. The function accepts three numbers a, b and c as the parameters and returns the largest one. Following is the code segment.

```
int find_largest ( int a, int b, int c )
{
        int largest = 0;

        if ( a > b )
        {
                if ( a > c )
                        largest = a;
                else
                        largest = c;
        }

        else
        {
                if ( b > c )
                        largest = b;
                else
                        largest = c;
        }

        return largest;
}
```

The complete C program is shown using Example 7.3. The program scans the numbers from the keyboard and then calls a function find_largest() passing these numbers as parameters.

```
#include<stdio.h>

int find_largest (int a, int b, int c);

int main()
{
        int a, b, c, largest = 0;

        printf("\n Enter the first number: ");
        scanf("%d", &a);

        printf("\n Enter the second number: ");
        scanf("%d", &b);

        printf("\n Enter the third number: ");
        scanf("%d", &c);

        largest = find_largest(a, b, c);

        printf("\n Largest number is: %d", largest);

        return 0;
}

int find_largest ( int a, int b, int c )
{
        int largest = 0;

        if ( a > b )
        {
                if ( a > c )
                        largest = a;
                else
                largest = c;
        }
```

```
        else
        {
                if ( b > c )
                        largest = b;
                else
                        largest = c;
        }

        return largest;
}
```

Example 7.3: A C program for finding the largest among three numbers using a function.

Problem 3: Finding largest among a series of numbers

We have already solved this problem in Chapter 6. We will now write a function for the same.

While tackling the problem in Chapter 6, we stored the numbers in an array. Then we applied the logic for finding the largest number.

Here, we will write this logic inside a function and then we will call the function from main(). Example 7.4 shows the complete program. In the program, we have passed only the number of elements (variable totalEle) as the parameter to the function find_lagest().

```
#include<stdio.h>

int find_largest (int);

int number[20];

int main()
{
        int totalEle, largest = 0;

        printf("\n Enter total number of elements in the series: ");
        scanf("%d", &totalEle);

        printf("\n Taking input to the array elements: ");

        for(int i=0; i<totalEle; i++)
        {
```

```
            printf( "\n Enter element %d: ", (i+1) );
            scanf("%d", &number[i]);
        }

        largest = find_largest(totalEle);

        printf("\n Largest number is: %d", largest);

        return 0;
}

int find_largest( int totalEle)
{

        int largest = number[0];

        for(int i=1; i<totalEle; i++)
        {
            if  ( number[i] > largest )
            largest = number[i];

        }

        return largest;
}
```

Example 7.4: A C program for finding the largest among a series of numbers.

We have also learned some new syntax of the language here. First, have a look at line number 2. In the function declaration, we have not written the parameter name; we have only written the data type.

Next, we have declared the array number[20] outside the main(). This makes the variable number a **global variable**. When we make a variable global, any function in the program can directly access that variable. As a result, it becomes possible for the function find_largest() to use the array number (line numbers 3, 5 and 6 in the function definition).

Problem 4: To find the digits of a number

This problem is solved in Chapter 4 (see Example 4.14). The logic was to repeatedly use the reminder operation to extract a digit from the right side and then to reduce the number by one tenth. We continued to do so using a loop as long as the number did not become zero.

We put this as the definition portion of a function. The function accepts the number as a parameter and it does not return any value. Example 7.5 shows the complete code.

```
#include<stdio.h>

void find_digit(int number)
{
        int temp = number, digit = 0;

        while(temp > 0)
        {
                digit = temp % 10;
                printf("\n Extracted digit is %d", digit);
                temp = temp / 10;
        }

}

int main()
{
        int number;

        printf("Enter the integer: ");
        scanf("%d", &number);

        find_digit(number);

        return 0;
}
```

Example 7.5: A C program for finding the digits of a number using function.

Can we find out the function declaration, definition and call for the function we used in the above program?

We can see the function definition and function calling in the above code. In fact, the function definition and declaration are written together. There is no separate declaration of the function. *This is another syntax in C programming we learned here.*

This syntax works when the definition of the function appears before it is called. In the program, the function is called in line number 17 while it is defined from line number 2 itself.

Problem 5: Storing the digits of a number in an array

In the previous problem, we have extracted the digits of a number and displayed them. Now, we need to store them in an array.

If we observe the above program, we see that in each iteration of the while loop, we extracted one digit. Now, this digit is to be stored as an array element. Following code segment can do this job.

```
digit = temp % 10;

digits[index] = digit;
```

Here, digit is the name of the array where we want to store the extracted digits of the number.

In the next iteration, the digit we extract has to be stored as the next array element. This means, the array index is to be incremented in each iteration of the loop. Following is the complete code segment.

```
while ( temp > 0 )
{
        digit = temp % 10;

        digits[index] = digit;

        Index = index + 1;

        temp = temp / 10;
}
```

The complete C program is shown using Example 7.6. The extracted digits of a number is stored in a global array digits [10]. The function find_digit() returns the last index of the array elements. This tells the number of elements stored in the array. We used this information to display the array elements when we came back from the function call.

```
#include<stdio.h>

int digits[10];

int find_digit ( int number )
```

```
{
        int temp = number, digit = 0, index = 0;

        while ( temp > 0 )
        {
                digit = temp % 10;
                printf ( "\n Extracted digit is %d", digit );

                digits[index] = digit;
                index = index + 1;

                temp = temp / 10;
        }

        return (index-1);

}

int main()
{
        int number, lastIndex;

        printf ( "Enter the integer: " );
        scanf ( "%d", &number );

        lastIndex = find_digit ( number );

        printf ( "\n Array elements are: \n" );

        for ( int i=0; i < = lastIndex; i++ )
                printf ("%d ", digits[i] );

        return 0;
}
```

Example 7.6: A C program for storing the digits of a number in an array using function.

We see that the elements are displayed in reversed order. That is, if the input number is 47837, it displays 7 3 8 7 4. To get it displayed in correct order, we can apply the logic for **displaying array elements in reverse order.** This was already discussed in Section 6.5.6 of the previous chapter.

Let us write another function display_digits() to display the digits of the numbers that got stored in the array. This function starts displaying the elements from the last index to 0 with a decrement in the index value at each iteration.

We present the complete program in Example 7.7. You can see that the main() function of the program looks simple and clear. It performs three tasks - (i) reads a number from the keyboard, (ii) calls a function to get the digits, and (iii) calls another function to display the digits.

```
#include<stdio.h>

int digits[10];

int find_digit ( int number )
{
        int temp = number, digit = 0, index = 0;

        while ( temp > 0 )
        {
                digit = temp % 10;
                printf ( "\n Extracted digit is %d", digit );

                digits[index] = digit;
                index = index + 1;

                temp = temp / 10;
        }

        return (index-1);

}

void display_digits(int lastIndex)
{

        printf("\n Array elements are: \n");

        for ( int i = lastIndex; i >= 0; i-- )
                printf ( "%d ", digits[i] );

}
```

```
int main()
{
        int number, lastIndex;

        printf ( "Enter the integer: " );
        scanf ( "%d", &number );

        lastIndex = find_digit ( number );

        printf ( "\n Array elements are: \n" );

        display_digits(lastIndex);

        return 0;
}
```

Example 7.7: A C program for displaying the digits of a number in correct order.

Problem 6: Counting number of adult persons

In this problem, we are given the age of a set of persons and we need to write a function to find the persons who are adults.

We know that a person is an adult if the age of the person is more than 18. We will use this very condition with a "if" construct.

In the first step, we will store the age of the persons in an array. In the next step, we will apply the logic to find the number of adult persons. We will perform these steps using two different functions.

The program is shown using Example 7.8. In this program, we have used an array age[] to store the age of the persons. The variable personCount stores the total number of persons, that is the total number of array elements. Both the array age[] and personCount are declared as global so that both the functions an input () and find Adult () can access them.

```
#include<stdio.h>

int age[20];
int personCount = 0;

void input()
{
        printf ( "\n We are taking inputs in an array \n" );
```

```
        printf ( "\n How many persons are there? " );
        scanf ( "%d", &personCount);

        for ( int i = 0; i < personCount; i++ )
        {
                printf ( "\n Enter age of person %d: ", (i+1) );
                scanf ( "%d", &age[i] );
        }
        printf ( "\n Input taken \n" );
}

void findAdult()
{
        int adultCount = 0;

        for ( int i = 0; i < personCount; i++ )
        {
                if ( age[i] >= 18 )
                {
                        printf( "\n Person %d is an adult ", (i+1) );
                        adultCount++;
                }

                else
                {
                                printf( "\n Person %d is NOT an adult ", (i+1) );
                }

        }

        printf("\n Total number of adult persons: %d", adultCount);
}

int main()
{
        input();

        findAdult();

        printf ("\n\n" );

        return 0;
}
```

Example 7.8: A C program for counting the number of adult persons.

Have a look at the main() function. It simply calls two functions: input() and findAdult(). The input() function takes the ages of the persons as input from the keyboard. The findAdult() function runs through the array elements and checks whether the age is greater than or equal to 18.

If the age of a person is greater than or equal to 18, we announce it and increment a counter *adultCount* that was initially assigned to zero. See the "if-else" clause inside findAdult().

Figure 7.3 shows an execution instance of the program where the user entered ages of 7 persons. Out of them, 4 persons were adults and the program rightly finds that.

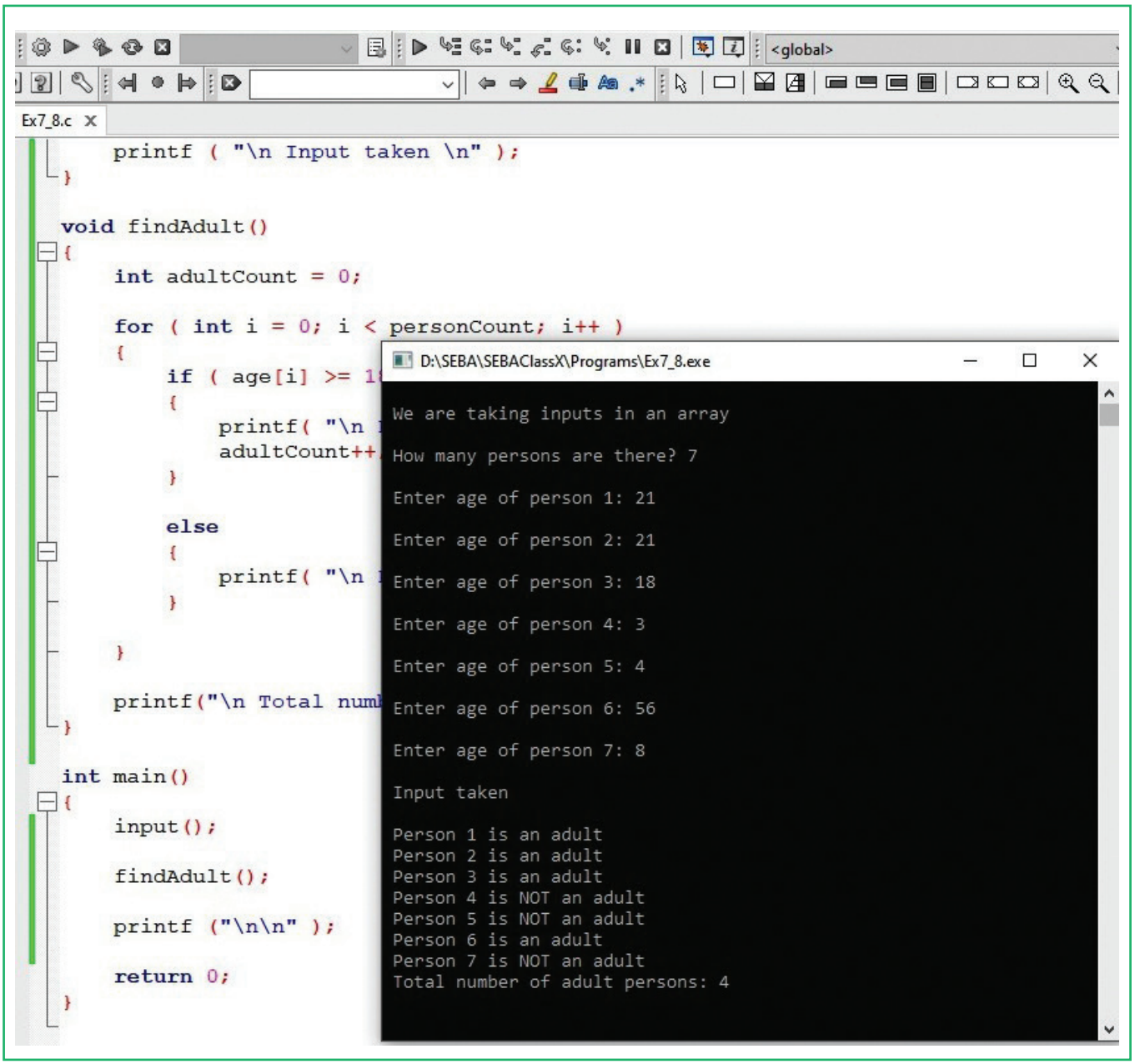


Figure 7.3: An execution instance of Problem 6.

Problem 7: Finding factorial of a number

Factorial of a number has many uses in mathematics and it is denoted by an exclamation mark (!). Let us first understand the problem.

The factorial of a non-negative integer n, denoted by n!, is the product of all positive integers less than or equal to n. We write,

n! = n . (n-1) . (n-2) . . . 3. 2. 1

For example, **5! = 5 . 4. 3. 2. 1 = 120**

Factorial of 0 is considered as 1.

If we need to define a function findFactorial() for calculating the factorial of a number, it will accept a non-negative number as the input. Then, we may use a loop to do the repeated multiplication.

The loop should start with the number n and it needs to run till 1. In every iteration of the loop, we need to decrement the number by 1. We present the code segment below.

```
int factorial = 1;

while ( number > 0 )
{
        factorial = factorial * number;

        number = number - 1;
}
```

In the case of the C program, we can take a number as an input from the user in main() and then we can call the factorial finding function passing the number as a parameter. Example 7.9 shows the complete code.

```
#include<stdio.h>


int findFacotiral ( int number )
{
```

```
        int factorial = 1;
        while ( number > 0 )
        {
                factorial = factorial * number;

                number = number - 1;
        }

        return factorial;
}

int main()
{
        int number, factorial = 0;

        printf ("\n Enter the number: " );
        scanf ( "%d", &number );

        factorial = findFacotiral ( number );

        printf ( "\n Factorial of the number is: %d", factorial );

        return 0;
}
```

Example 7.9: A C program for finding the factorial of a number.

7.5 RECURSIVE FUNCTION

In the last section of this chapter, let us study something very interesting!

We have already learned that a function calls another function. Accordingly, the program control goes from the caller function to the callee function. If a function calls itself, we name it as **recursion**. The function is called a **recursive function**.

The syntax of such a function is shown below. We see that we are calling the function f() from the definition of f();

```
void f()
{
        . . . . . .

        f();

        . . . . . .
}
```

At first the function f() may be called from outside, maybe from main() as shown below.

```
void f()
{
        . . . . . .

        f();
        . . . . . .
}

int main()
{
        . . . . . .

        f();

        . . . . . .
}
```

Now the factorial finding problem (Problem 9) can be programmed nicely. Have a look at Example 7.10. After taking the input in main(), we are calling the function findFacotiral() in line number 13.

The control comes to the definition of the function findFacotiral() with the input number as a parameter. Then the function calls itself with parameter number -1. In the next iteration, the function will call itself with parameter number -2 and so on. When the value of the parameter becomes 0, the function returns 0. Then the control will go in the reverse direction.

```
#include<stdio.h>


int findFacotiral ( int number )
{
        if ( number = = 0 )
                return 1;

        else
                return number * findFacotiral ( number - 1 );

}

int main()
{
        int number, factorial = 0;

        printf ("\n Enter the number: " );
        scanf ( "%d", &number );

        factorial = findFacotiral ( number );

        printf ( "\n Factorial of the number is: %d", factorial );

        return 0;
}
```

Example 7.10: A C program for finding the factorial of a number using recursion.

Figure 7.4 demonstrates an execution of the function for number = 5. Initially, findFactorial was called with 5. The function returns 5 * findFactorial(4). Then, it returns 4 * findFactorial(3) and so on. Finally, findFactorial(0) returns 1.

After traversing in the reverse direction, the function finally returns 5*4*3*2*1 to the initial caller, that is main(). Figure 7.4 shows this.

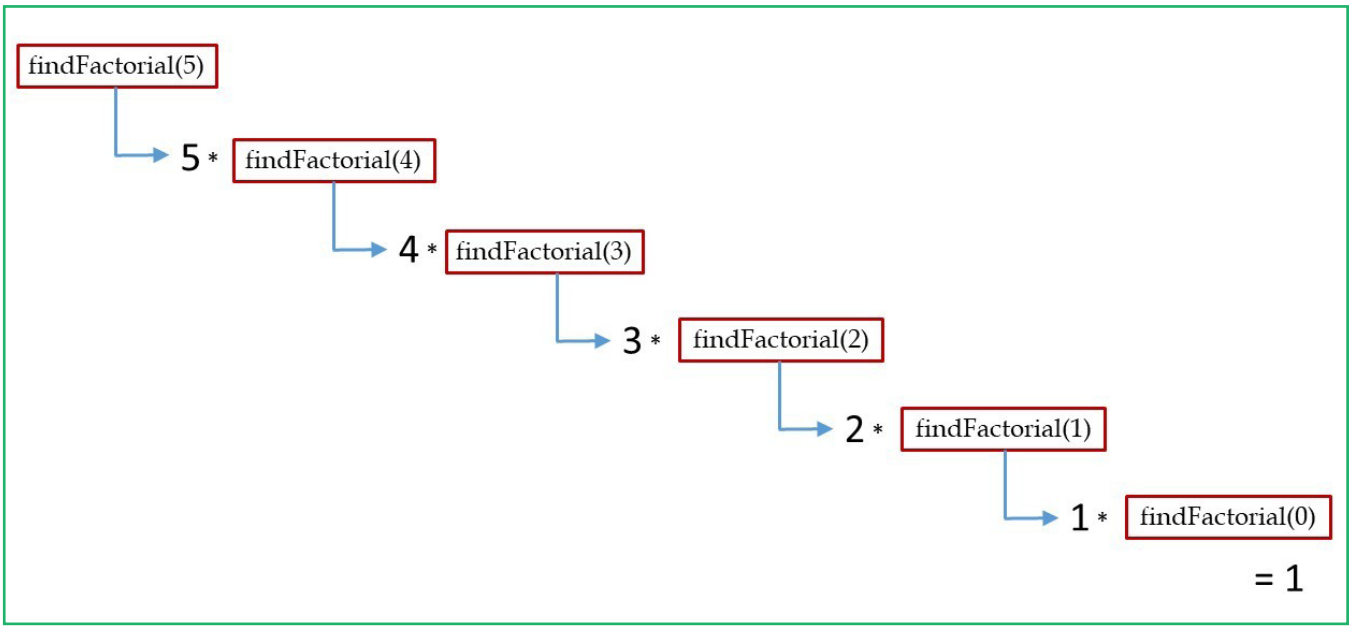


Figure 7.4: Execution instance of a call to the recursive function findFactorial(5)

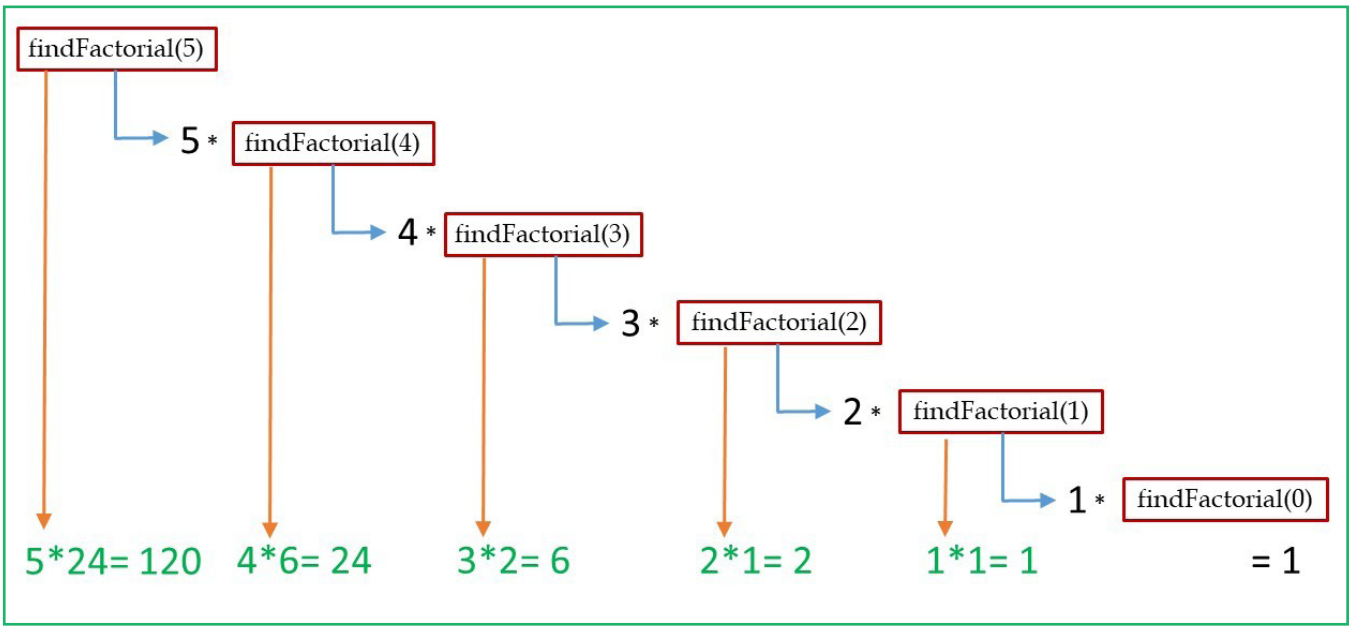


Figure 7.5: Return of the recursive function call findFactorial(5)

Exercise

1. What is a global variable in C? Why do we need such a variable?

2. Write the syntax of a function declaration. The name of the function is calculateAge(). The function accepts the current year and birth year of a person. The function returns the age of the person.

3. Write the code segment for the function definition of the above function calculateAge().

4. What are different types of functions in C ? Differentiate among them.

5. Differentiate between caller and callee functions. write a small C progarm and identify the caller and callee function in it.

6. When do we call a function user-defined? Is printf() a user-defined function? Justify.

7. Can we have two functions with the same name but with different numbers of parameters in a single C program? Write a simple C program to justify your answer.

8. What are different components of a function? Show with a complete C program.

9. Define recursive function. Can we use a recursive function to solve all kinds of problems?

10. Consider the below code and list all the syntax errors.

```
#include<stdio.h>

int fun ( int x )
{
        if ( x %2 == 0 )
        return 1;

        else
        return 0;

}

int main()
```

```
{
        int number;

        printf ("\n Enter the number: " );
        scanf ( "%d", &number );


int x = fun ( );

        return 0;
}
```

11. Consider the code segment below and find out the output if the user enters 5 from the keyboard when asked for.

```
#include<stdio.h>


int fun ( int x )
{
        if ( x %2 == 0 )
        return 1;

        else
        return 0;

}

int main()
{
        int number;

        printf ("\n Enter the number: " );
        scanf ( "%d", &number );

        int x = fun ( number);

        printf("%d", x);

        return 0;
}
```

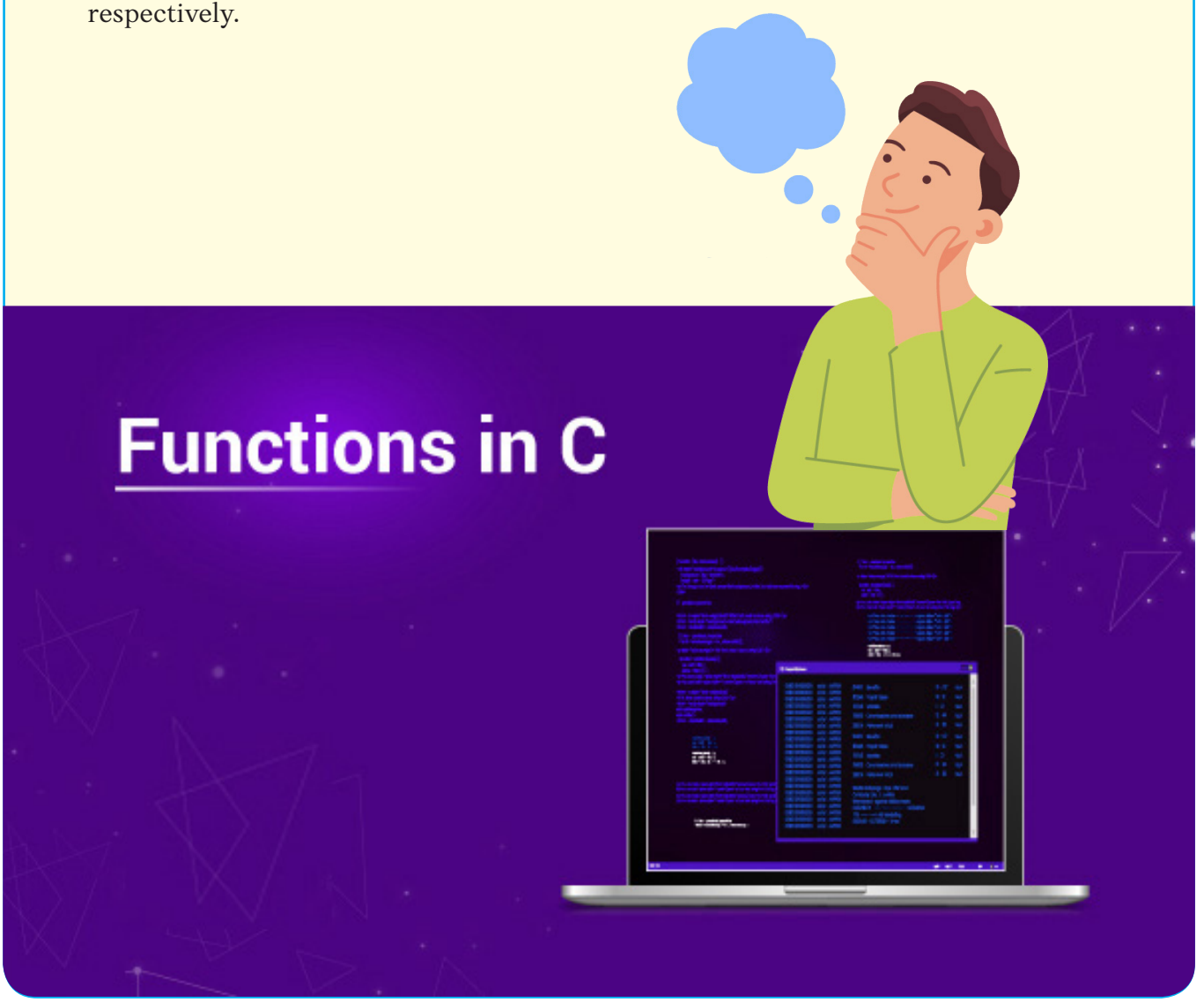
12. Write a C program and define a function square() that accepts a number as the parameter and returns the square of that number as output.
13. Write a C program and define a function search () that searches an element in an array and returns the index of the element.

14. Write a C program and define a recursive function to find the summation of first N natural numbers.

15. Write a C program and define a function add() that accepts three integers. These integers indicate indices of an integer array. The function returns the summation of the elements stored in those indices.

7	8	8	0	0	9

For example, if we call the function add(0, 2, 5), the function will return 24. The output is formed by 7 + 8 + 9 because elements at indices 0, 2 and 5 are 7, 8 and 9 respectively.



CHAPTER 8

Pointers in C

In this chapter

We will learn a small feature of C programming language that provides another way to access a variable. Using this feature, we will also solve one of the dement of array.

8.1 INTRODUCTION

Recall our discussion from Class IX that when we declare a variable in a C program, some space is allocated for that variable in computer memory. The space has a specific location and it is identified by its address.

We may use %p specifier and ampersand operator (&) to display the address of a variable declared in C. Example 8.1 shows a simple program to demonstrate the same. The program will output a hexadecimal number that indicates the memory location where the variable x is stored.

```
#include<stdio.h>

int main()
{
        int x = 79;

        printf( "\n Address of x in memory is %p", &x );

        return 0;
}
```

Example 8.1 A simple C program to display the address of a variable.

Quick activity : Run the above C program and analyze the output. What is the length of the hexadecimal number you get as an output ?

If we want to see the size of the address of x, we may use the sizeof() operator of C. Example 8.2 shows this. The sizeof() operator returns the size in number of bytes.

```
#include<stdio.h>

int main()
{
        int x = 79;

        printf( "\n Address of x in memory is %p", &x );

        printf( "\n Size of the address of x is %u", sizeof(&x) );

        return 0;
}
```

Example 8.2 A simple C program to display the address of a variable and its size.

Quick activity : Run the above C program and see the output. The computer you are using right now uses that many bytes for specifying the address of a variable.

Now, if we want to store this address of variable x in another variable, we can do so with the help of a **pointer**.

A pointer is a special variable that stores the address of another variable. Pointers have many usages in C programming. In this chapter, we will learn some of them.

8.2 USING POINTER WITH A VARIABLE

In the above program of Example 8.2, if we want to have a pointer that stores the address of the variable x, we can do so with the code segment below.

```
int *ptr;

ptr = &x;
```

In the above code segment, variable ptr is declared as a pointer. As the data type is written as int, the pointer ptr **can store the address of any integer type of variable**. The asterisk * used to declare a pointer is the same asterisk used for multiplication. However, in this statement the asterisk is being used to designate a variable as a pointer.

When we assign the address of x in ptr (second line of the code segment), we say that the pointer ptr points to the variable x. Graphically, we may represent in the following way.

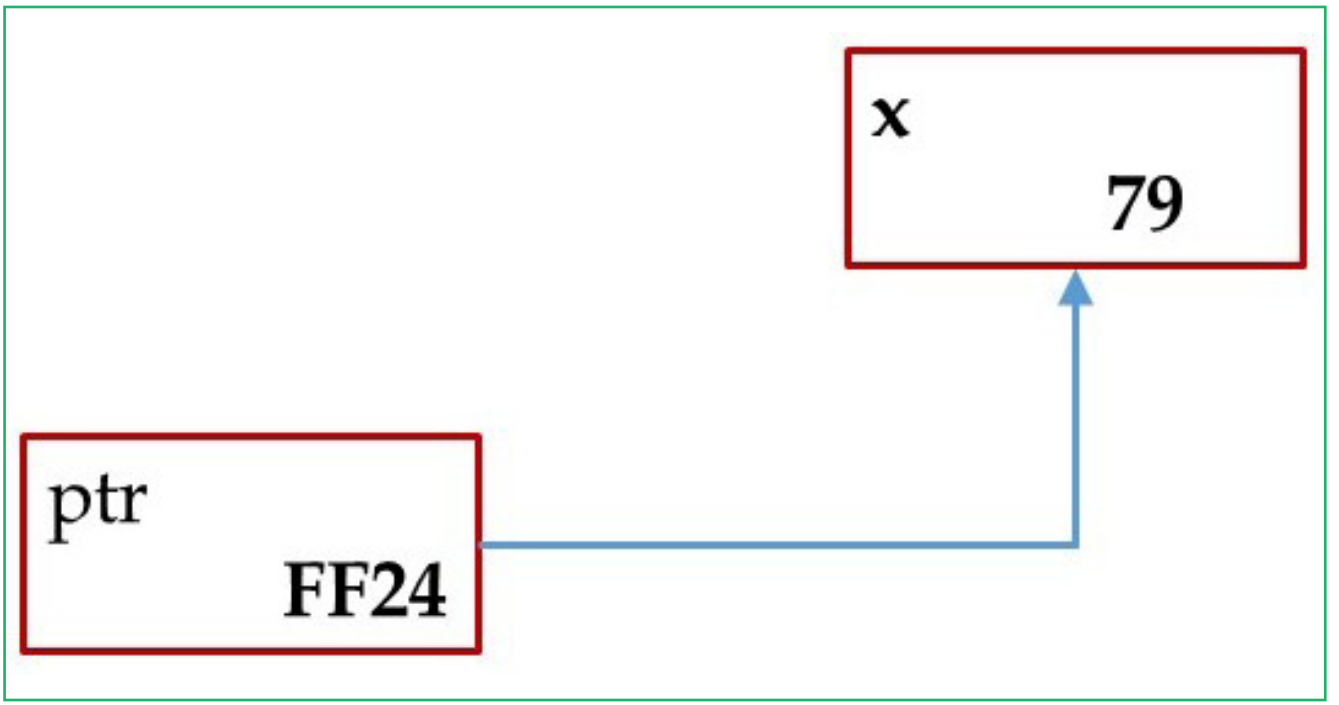
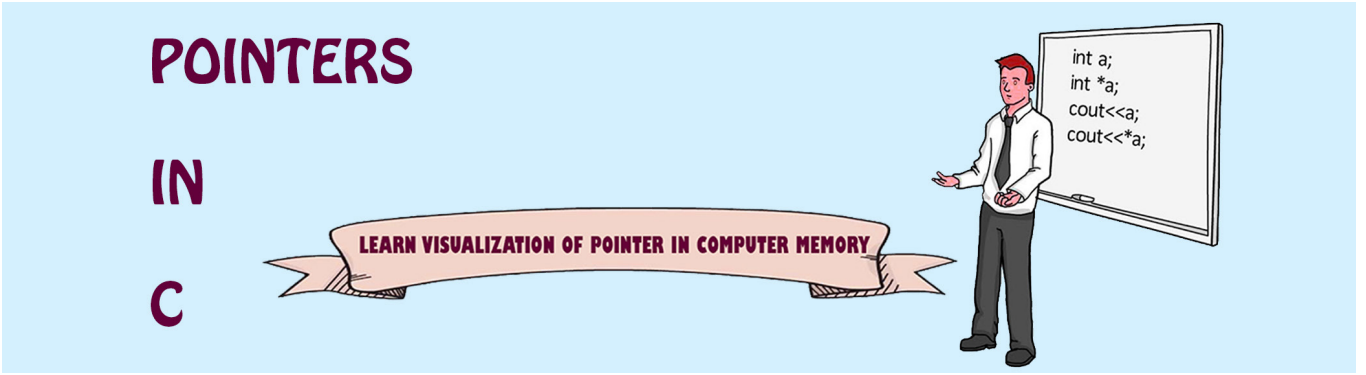


Figure 8.1 Illustration of a pointer in C

In the figure above, the rectangles indicate variables. We used two variables x and ptr. Let us assume that the address of the variable x is FF24. The value of x is taken as 79.

The ptr is also a variable and hence it is also stored in some memory location. We may visualize this using Figure 8.2. We know that we can view the computer memory as a one dimensional array. Both the variables x and ptr are stored in memory. As FF24 is the address of the variable x, the content of the pointer pointing to x is also FF24.



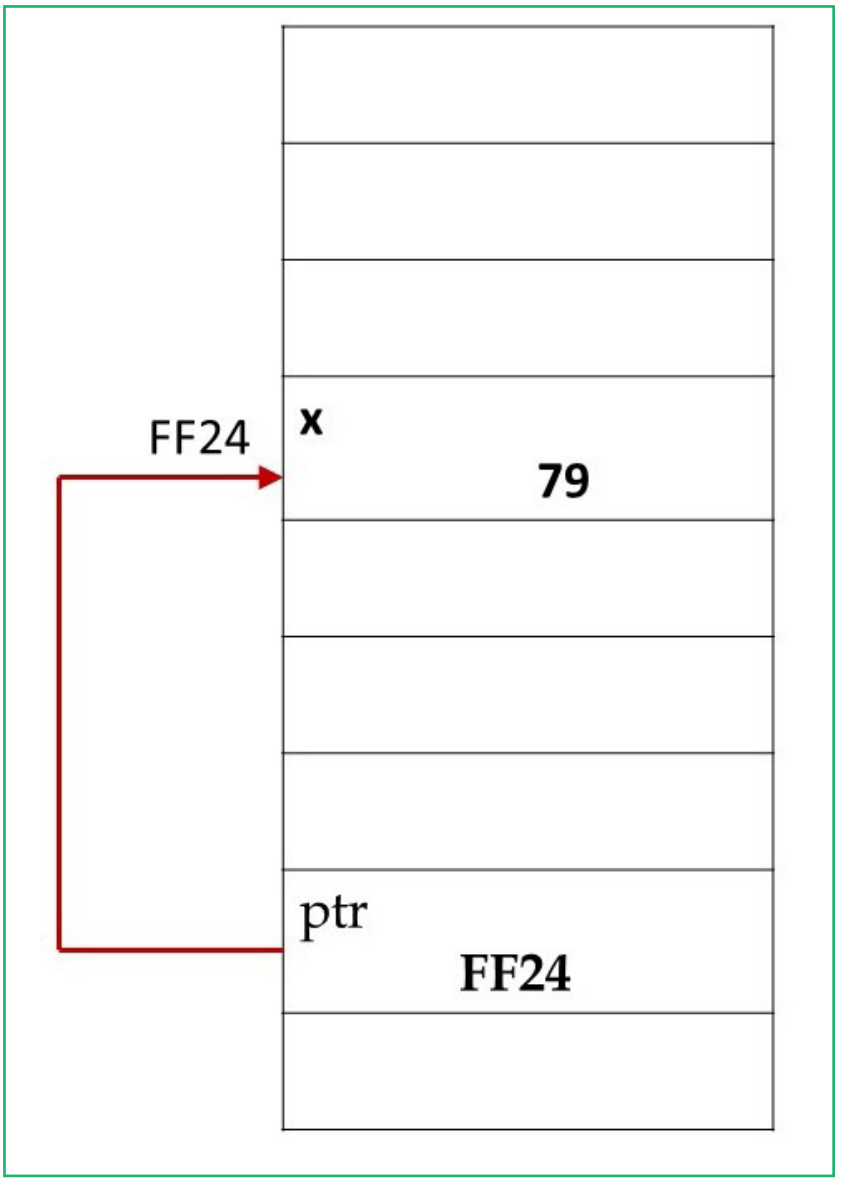


Figure 8.2: Visualization of pointer in computer memory

Let us see a simple C program in Example 8.3 to understand the concept of pointer further. In the program, ptr is declared as an integer type of pointer and it points to another integer variable x.

Now, we can access x with the help of the pointer ptr. See the last printf() statement. We are displaying the value of x with *ptr. We have also displayed the address of x using ptr (second printf() statement).

```
#include<stdio.h>

int main()
{
        int x = 79;

        int *ptr;

        ptr = &x;

        printf( "\n Address of x in memory is %p", &x );

        printf( "\n Address of x in memory using pointer is %p", ptr );

        printf( "\n Value of x is %d", x );

        printf( "\n Value of x using pointer is %d", *ptr );

        return 0;
}
```

Example 8.3 A simple C program to demonstrate a pointer variable.

The pointer gives another way to reach the variable x. The variable x and *ptr can be used alternatively. Any operation done on x is the same doing on *ptr. For instance, if we want to add 7 with x, we may do so by x = x + 7 or, *ptr = *ptr +7, in either way.

The programs in Example 8.4 and 8.5 will give the same output. Both will increment the value of the variable by 7.

```
#include<stdio.h>

int main()
{
        int x = 79;

        int *ptr;

        ptr = &x;

        x = x + 7;

        printf("\n Value of x is %d", x);

        printf("\n Value of x using pointer is %d", *ptr);


        return 0;
}
```

Example 8.4 A C program to demonstrate use of a pointer with a variable.

```
#include<stdio.h>

int main()
{
        int x = 79;

        int *ptr;

        ptr = &x;

        *ptr = *ptr + 7;

        printf("\n Value of x is %d", x);

        printf("\n Value of x using pointer is %d", *ptr);


        return 0;
}
```

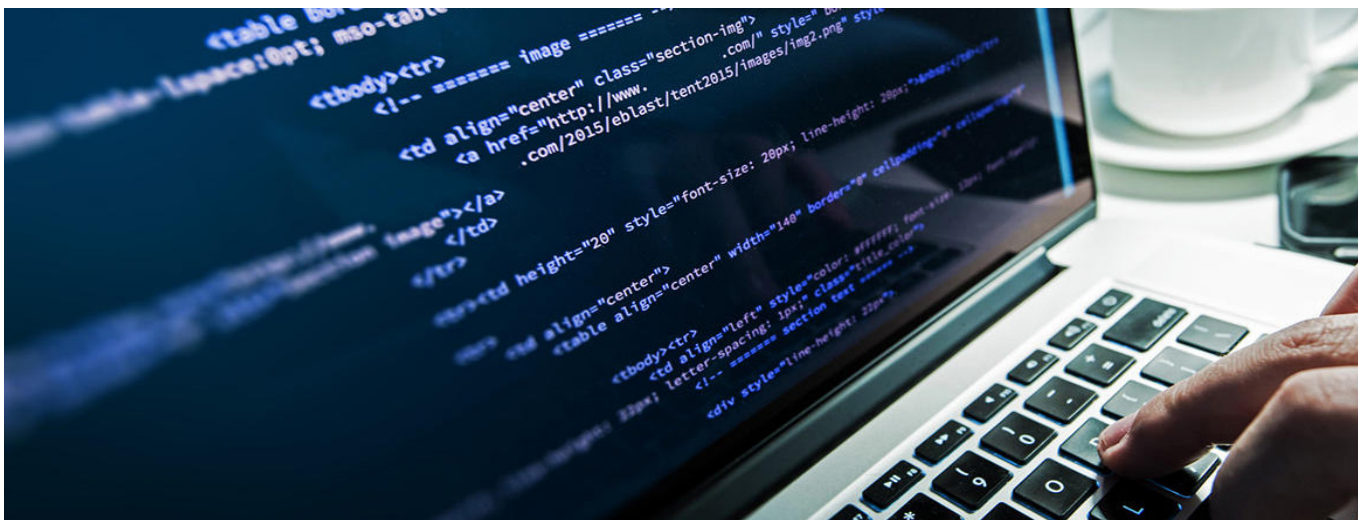
Example 8.5 A C program to demonstrate simple operations using a pointer.

We can use a single pointer to point to different variables at different instances of time. In the above C programs, *ptr was pointing to the variable x. We can make it point to any other variable. But the variable has to be an integer variable because *ptr was declared as type int.

Example 8.6 shows a C program where the pointer was initially pointing to x and then it points to the variable y. Both x and y are int type variables. The first printf() displays 79 and the second printf() displays 27.

```
#include<stdio.h>

int main()
{
        int x = 79, y = 27;

        int *ptr;

        ptr = &x;

        printf("\n Value of x using pointer is %d", *ptr);

        ptr = &y;

        printf("\n Value of y using pointer is %d", *ptr);


        return 0;
}
```

Example 8.6 A C program where a pointer points to different variables at different times.



8.3 USING POINTER WITH DIFFERENT TYPES OF DATA

In all the examples above, a pointer was used to point to an integer. It can also be used to point to other types of variables. In Example 8.7, we present a C program to access a character variable using a pointer. The syntax remains the same.

```
#include<stdio.h>

int main()
{
      char x = 'A';

      char *ptr;

      ptr = &x;

      printf("\n Value of x using pointer is %c", *ptr);


      return 0;
}
```

Example 8.7 A C program using a pointer for a char type variable.

Now, let us see an interesting behaviour of pointers. Have a look at the C program presented in Example 8.8. We have used two pointers *chPtr and *intPtr. The first pointer points to a char type variable and the second pointer points to an int type variable.

```
#include<stdio.h>

int main()
{
      char x = 'A';

      char *chPtr;

      chPtr = &x;
```

```
        printf("\n Value of x using pointer is %c", *chPtr);

        printf("\n Value of the pointer is %p", chPtr);

        chPtr++;

        printf("\n Value of the pointer after increment is %p", chPtr);


        /* Using pointer for int */

        int y = 27;

        int *intPtr;

        intPtr = &y;

        printf("\n\n Value of y using pointer is %d", *intPtr);

        printf("\n Value of the pointer is %p", intPtr);

        intPtr++;

        printf("\n Value of the pointer after increment is %p", intPtr);

        printf("\n\n");


        return 0;
}
```

Example 8.8 A C program where pointers are used for different types of variables.

Quick activity : Run the above C program and observe the output of the program carefully.

Observation:

In the first part of the program, we have used a pointer *chPtr that points to a char type variable x. Next we have incremented the content of the pointer using chPtr++. We see that chPtr value is increased by **one byte**.

In the second part of the program, we did the same thing. But here, we used the pointer for an int type variable y. When we increment the content of the pointer intPtr, it increments by **four bytes**.

This is because chPtr was pointing to a char and chPtr++ operation makes it point to the next char. The size of a char variable is 1 byte. Thus it increments by one byte.

In case of intPtr, the size of an int variable is 4 byte. Thus intPtr++ operation increments the value by four bytes.

8.4 ARRAY AND POINTER

Pointers can be used with arrays to do many useful tasks. In this book, we will discuss two simple but interesting aspect.

8.4.1 Accessing array elements using pointers

Let us first see how we can access the elements of an array using pointers. We have already seen that when we declare an array, a certain amount of memory is allocated by the compiler. For example, an int array of size 5 will allocate 20 bytes (assuming int takes 4 bytes, 5x4=20).

Have a look at the simple C program in Example 8.9. We have taken an int array of size 5 and we have initialized some values to the elements. We have then displayed the addresses of these array elements.

```
#include<stdio.h>

int main()
{
        int x[5] = {7, 8, 1, 0, 1};

        for(int i=0; i<5; i++)
        {
                printf( "\n Address of element %d is %p", i, &x[i] );
```

```
        }

        printf( "\n\n Base Address of the array is %p", x );

        return 0;
    }
```

Example 8.9 A C program displaying the addresses of the array elements.

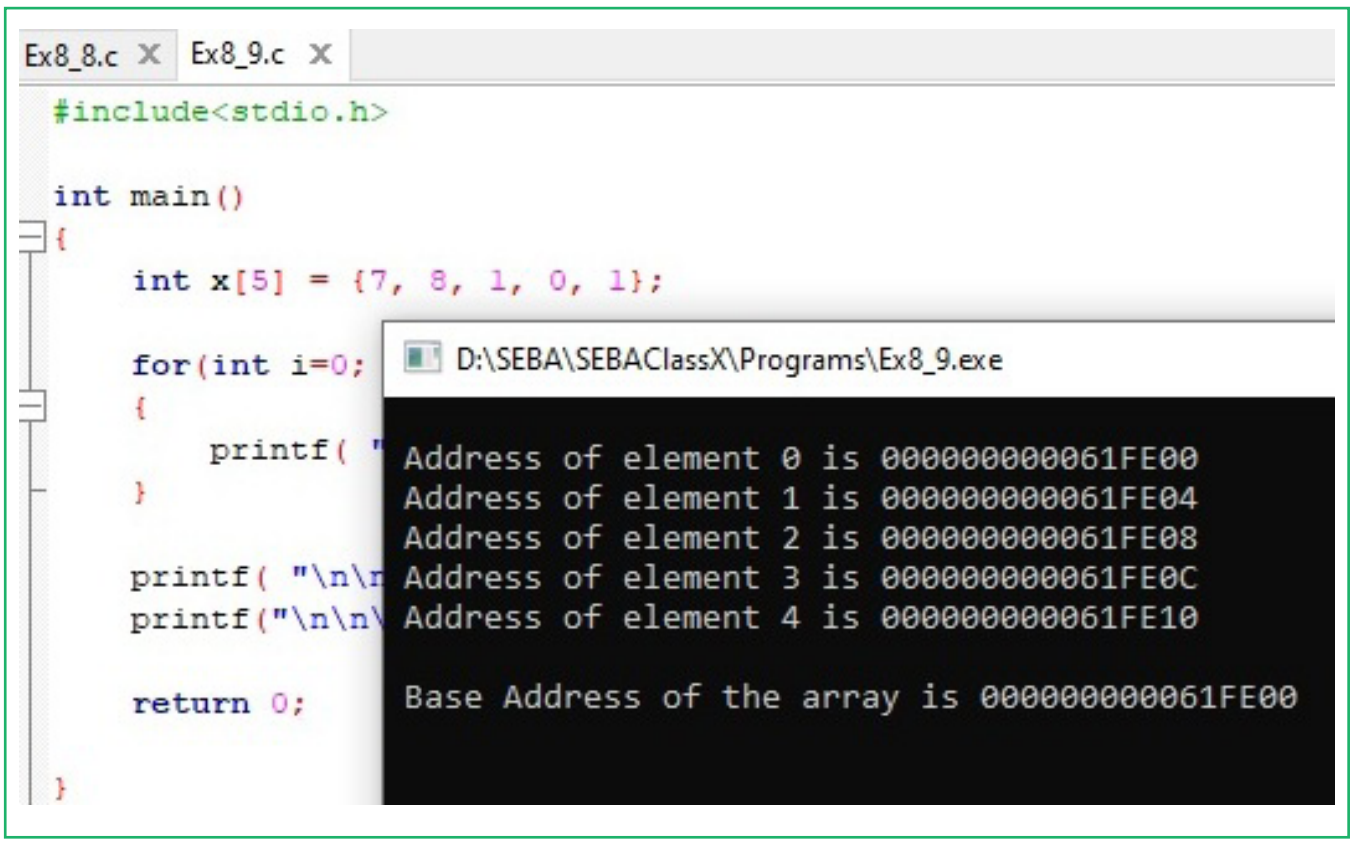


Figure 8.3: Output of Example 8.9.

The output of the program is shown in Figure 8.3. We see that there is a difference of 4 bytes between the addresses of the consecutive elements. We have also seen that the address of the first element, denoted by &x[0] is the same as the base address of the array, denoted by x.

The concept of a pointer teaches us that if we know the address of a variable, we can get its value using pointer. We will apply this concept to access the elements of the array.

The first element of the array can be accessed using *x because x in the address of the first element of the array.

We have also seen in Example 8.8 that if we increment the pointer value, it points to the next element. Thus (x+1) points to the next element of the array. Hence, the second element of the array can be accessed using *(x+1).

Thus, the element with index i of the array can be accessed using *(x+i).

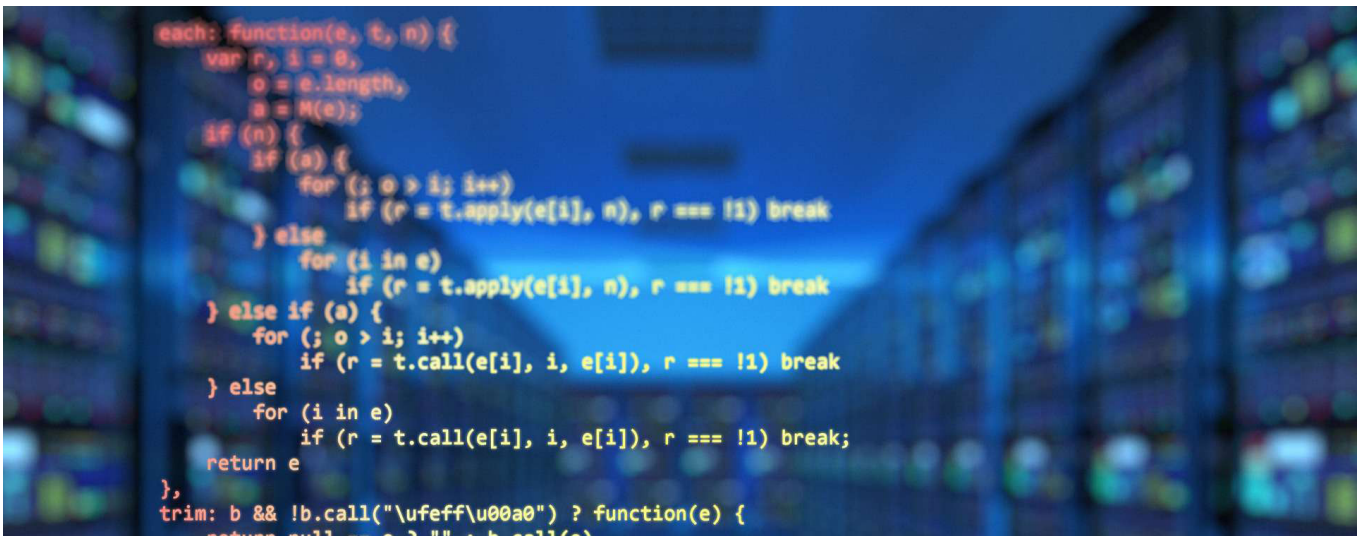
If we need to access all the elements one-by-one, we can use a loop as follows.

```
for(int i=0; i<5; i++)
{
        printf( "\n Value of element %d is %d", i, *(x+i) );
}
```

The program where the elements of an array are accessed using pointer is shown in Example 8.10. The output of the same is shown using Figure 8.4.

```
#include<stdio.h>

int main()
{
        int x[5] = {7, 8, 1, 0, 1};

        for(int i=0; i<5; i++)
        {
                printf( "\n Value of element %d is %d", i, *(x+i) );
        }

        return 0;
}
```

Example 8.10 A C program displaying the addresses of the array elements.



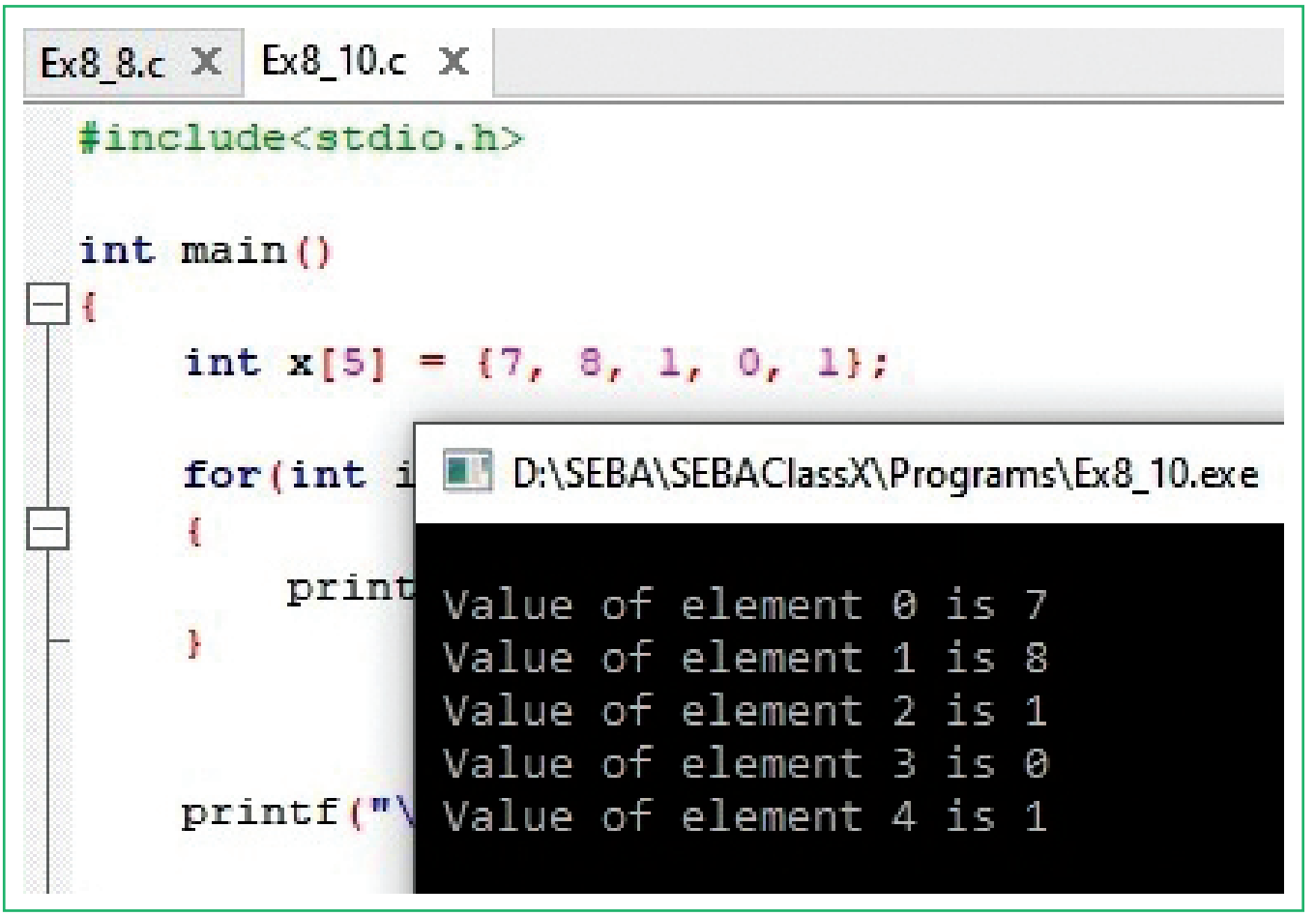


Figure 8.4: Output of Example 8.10.

8.4.2 Dynamic memory allocation

Do you remember the demerits of arrays? One of the demerits of arrays is that it is of fixed size and we need to decide the array size at the compile time. We specify the size of the array at the time declaration only.

```
int age[10];
```

For instance, the above declaration says the array age[ ] can store at max 10 elements. Compiler reserves these memory spaces at compile time itself.

If we want to decide and allocate the memory spaces at run time (that is, when the program executes), we call it **dynamic memory allocation**. And a pointer variable is used to achieve this.

We use a pointer and we allocate spaces to it when the program runs depending on our requirement. C supports a few library functions to allocate memory dynamically. We will use one such function **malloc()** in our programs.

The following code segment allocates spaces for 10 integers at the runtime to an int type pointer *ptr.

```
int *ptr;

ptr = (int*) malloc ( 10 * sizeof(int) );
```

We can use a variable in place of 10. The value of the variable can be taken from the keyboard at runtime. For instance, the following code segment will ask the user and accordingly allocate space.

```
int *age, personCount;

printf("\n How many persons are there? ");
scanf("%d", &personCount);

age = (int*) malloc (personCount * sizeof(int) );
```

The above code segment allocates a total of 4 times personCount number of bytes (assuming int takes 4 bytes). We have also seen that this is prefixed by (int*). This is called a **type-casting** operation. The syntax here indicates that the allocated space will be used to store int type variables.

Once the space is allocated to the pointer, we can use it as an array. When the user runs the program (maybe at some later point of time), he/she can give the input based on the requirement. Accordingly, that much amount of memory will be allocated. There is no wastage of memory. Thus this becomes an efficient technique.

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

int main()
{
        int *age, personCount;

        printf("\n How many persons are there? ");
        scanf("%d", &personCount);

        age = (int*) malloc ( personCount * sizeof(int) );

        for( int i=0; i<personCount; i++ )
        {
```

```
            printf( "\n Enter age of person %d ", i );
            scanf("%d", *(age + i) );
        }

        for( int i=0; i<personCount; i++ )
        {
            printf( "\n Age of person %d is %d", i, *(age + i) );

        }

        return 0;
}
```

Example 8.11 A C program showing dynamic memory allocation.

All the programs we have seen so far were using only the stdio.h header file. But the program in Example 8.11 uses another header file stdlib.h. This header file provides support for the malloc() library function.

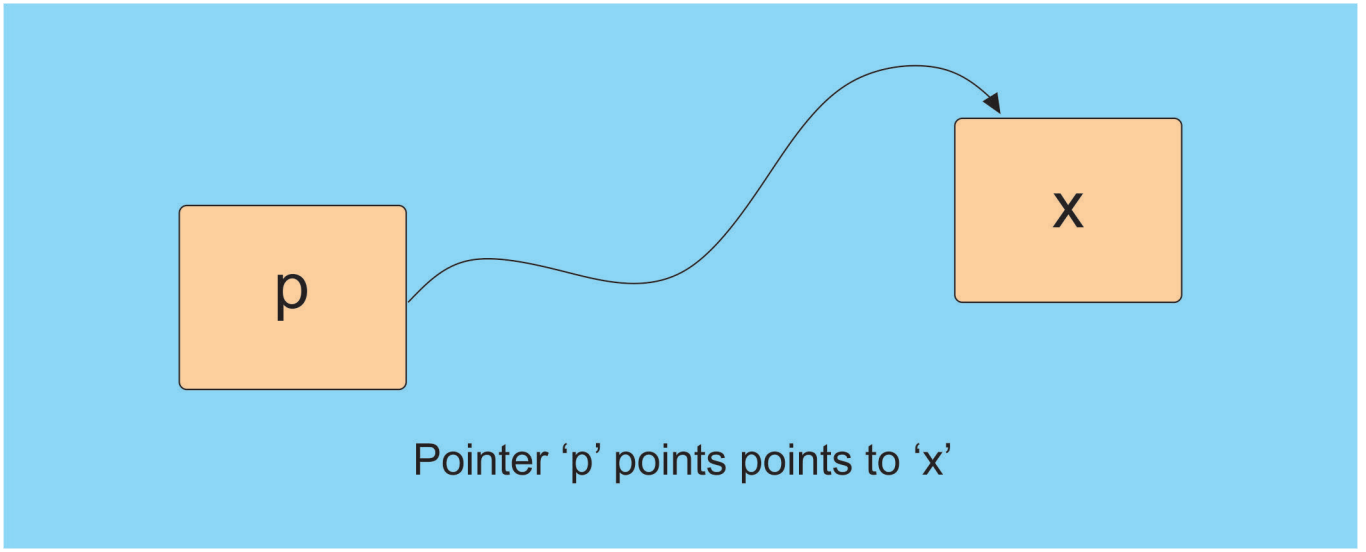
In case the system fails to allocate the requested amount of memory in the malloc() function, it returns NULL. We may also check this using a conditional statement. If the malloc() function returns NULL, we may terminate the program. Otherwise we may proceed normally. Example 8.12 shows this.

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

int main()
{
        int *age, personCount;

        printf("\n How many persons are there? ");
        scanf("%d", &personCount);

        age = (int*) malloc ( personCount * sizeof(int) );

        if( age = = NULL )
        {
```

```
            printf(" \n Sorry! Requested memory cannot be allocated.");
            return 0;
        }

        for( int i=0; i<personCount; i++ )
        {
            printf( "\n Enter age of person %d ", i );
            scanf("%d", (age + i) );
        }

        for( int i=0; i<personCount; i++ )
        {
            printf( "\n Age of person %d is %d", i, *(age + i) );

        }

        return 0;
}
```

Example 8.12 Another C program showing dynamic memory allocation.

In this chapter, we have introduced dynamic memory allocation technique using pointer. We have allocated memory only for int type variables in the examples. Dynamic memory allacation technique can be used for other types of variables as well. In the next chapter, we will use this technique to allocate memory for structure variables.



Exercise

1. How is a pointer variable different from a normal variable?

2. Why is dynamic memory allocation an efficient memory management technique?

3. How many bytes are needed to store an int pointer variable? Is it the same for a char pointer variable? Write a simple C program to explain your answer.

4. Write the output of the following code segment.

a.

```
int *ptr, x = 9;

ptr = &x;

printf("\n %d", (*ptr)++);
```

b.

```
int *ptr, x = 9;

ptr = &x;

printf("\n %d", (*ptr)++);

printf("\n %d", *ptr);
```

c.

```
int *ptr, x = 9;

ptr = &x;

int y = ++(*ptr);

printf("\n %d", y);
```

d.

```
char *ptr, x = 'A';

ptr = &x;

char y = *ptr;

printf("\n %c", y);
```

e.

```
char *ptr, x = 'A';

ptr = &x;

char y = (*ptr)++;

printf("\n %c", y);
```

f.

```
char *ptr, x = 'A';

ptr = &x;

char y = ++(*ptr);

printf("\n %c", y);
```

g.

```
char *ptr, x = 'A';

ptr = &x;

char *y;

y = ptr;

printf( "\n %c", ++(*y) );
```

5. Write a C program to dynamically allocate memory for an array to store 10 integers and display the first 5 out of them.

6. Write a C program to dynamically allocate memory for an array to store runs scored by Virat Kohli in the last ten ODI cricket matches. Write a function to find the maximum one.

7. Write a C program and define a function that takes the length of your name as an input parameter and then allocates memory dynamically to store your name. Write another function to display the name.

8. Write a C program to store some integer variables in an array. Then write functions to the following.

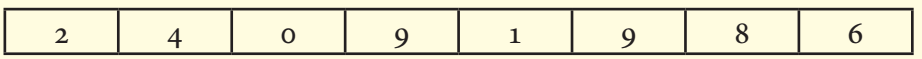
    1. To calculate the number of even numbers in the array.

    2. To dynamically allocate memory to a new array to store only the even numbers.

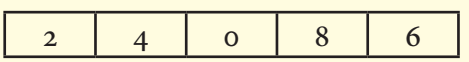
    3. To copy the even numbers from the first array to the second one.

Sample run:

First array



Second array



9. Write a C program to store some integer variables in an array. Then write functions to the following.

1. To calculate the number of non-zero elements that are divisible by 3.
2. To dynamically allocate memory to a new array to store only those elements.
3. To copy the selected elements from the first array to the second one.
4. To calculate the summation of these elements.

Sample run:

First array

2	4	0	9	1	9	8	6

Second array

9	9	6

Summation: 24

CHAPTER 9

Structure in C

In this chapter

We will first learn to create our own data type. Then we will learn several techniques to use them. Finally, we will develop a library management application using the concepts we learned so far.

9.1 INTRODUCTION

Suppose, we need to store various information about students in a C program and then we need to display the records.

For simplicity, let us say the information of a student consists of the roll number and age. We need to keep information about N such students.

The trivial solution to this problem is to declare two different arrays of size N - one to store roll number and another one to store age as shown in Figure 9.1.

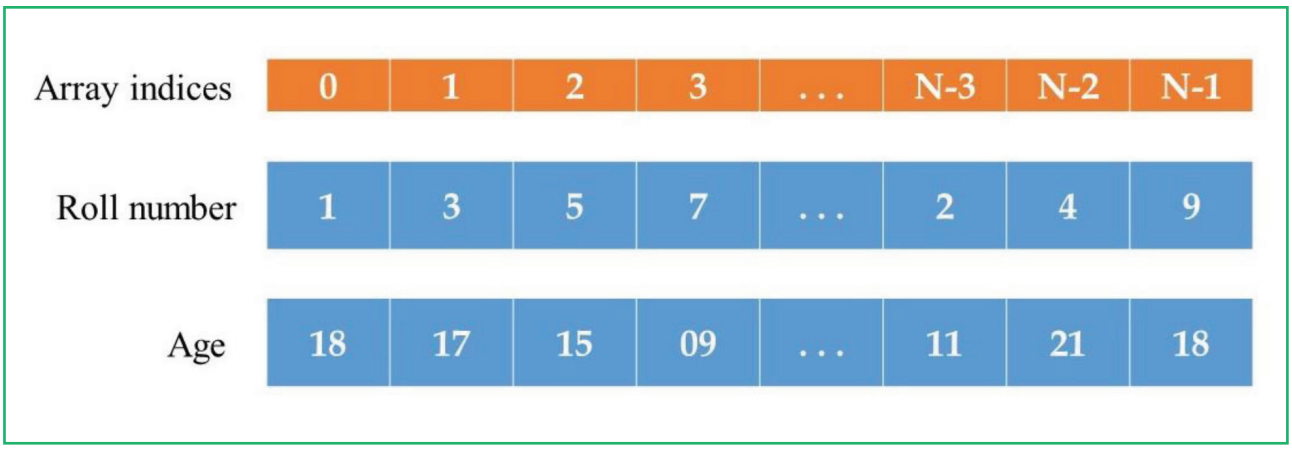


Figure 9.1 Two different arrays for storing student information.

A C program demonstrating the same is presented in Example 9.1. The program uses two int type arrays where memory is allocated statically. The first for loop takes us to visit all the elements of both the arrays to take inputs. The second for loop again goes through all the elements in both the arrays to display the values.

```
#include<stdio.h>

int main()
{
        int roll[20], age[20], studentCount;

        printf( "\n How many students?  " );
        scanf( "%d", &studentCount );

        for ( int index = 0; index < studentCount; index++ )
        {
                printf( " \n Enter roll for student %d ", index );
                scanf(" %d", &roll[index] );

                printf( " \n Enter age for student %d ", index );
                scanf( "%d", &age[index] );

        }

        printf( "\n Entered details are as follows: \n \n " );

        for ( int index = 0; index < studentCount; index++ )
        {
                printf( " \n Roll for student %d is %d", index, roll[index] );
                printf( " \n Age for student %d is %d", index, age[index] );

        }

        return 0;
}
```

Example 9.1 A C program where multiple arrays are used to store information about students.

The program looks simple. But this is not an efficient way to store information in computer memory. When we need the details about a particular student, we need to get the data from two different arrays and they may be stored far away from each other in computer memory.

The C programming language supports one efficient way to handle the scenario when we need to deal with multiple fields of different types. This is called **structure**. Structure is a user-defined data type that allows us to combine data of different types together. Structure helps to construct a complex data type that is more practical and meaningful.

Structure is somewhat similar to an array, but an array holds data of similar type only. But structure on the other hand, can combine data of different types.

9.2 DEFINING A STRUCTURE

The general syntax of a structure definition is as follows.

```
struct structure_name
{
        data_type member1;
        data_type member2;

        . . .

        data_type memeberN;
};
```

Here, **struct** is a keyword. Every field of a structure is called a member. A member can be of any data type. We enclose all the members within a pair of curly braces and put a semicolon (;) to denote the end of the structure. We also need to give a name to the structure we define. Once a structure is defined, we can use it as a new data type.

The following code segment declares a structure whose name is Student_Info. It has two members: roll and age, both are of type int.

```
struct Student_Info
{
        int roll;
        int age;
};
```

Now, if we want to use this structure to store information about students, we need to create variables of newly defined data type Student_Info. The following code segment creates two variables "student1" and "student2" of type Student_Info.

```
struct Student_Info student1, student2;
```

We need to prefix the declaration of a structure variable with the keyword "struct". Every variable has its individual storage to store the value for each field. Here the fields are roll and age. Thus every variable of type Student_Info has its individual roll and age value.

9.3 ACCESSING MEMBERS OF A STRUCTURE

Members of a structure can be accessed using a **dot operator** (.). The syntax is given below.

```
Structure_Variable_Name.Member_Name
```

For example, in order to access the roll field of student2, we need to use the following.

```
student2.roll
```

This (student2.roll) may be used as a separate variable such that we can display its value using printf or scan any value to it. We can also use this variable for other purposes. The following code segment will display the value stored in the roll field of student2.

```
printf( " %d ", student2.roll );
```

We present a complete C program in Example 9.2 where a student structure is created with two fields in it and then two structure variables are declared. We then assigned some values to them and displayed those values using printf() statements.

```
#include<stdio.h>

int main()
{
        struct Student_Info
        {
                int roll;
                int age;
        };

        struct Student_Info student1, student2;

        student1.roll = 1;
        student1.age = 17;

        student2.roll = 2;
```

```
    student2.age = 19;

    printf( "Roll and age of the students are: \n ");

    printf( "Student 1: Roll: %d  Age: %d", student1.roll, student1.age );
    printf( "\n Student 2: Roll: %d  Age: %d", student2.roll, student2.age );

    return 0;
}
```

Example 9.2 A C program demonstrating a simple structure.

We may also scan values to the structure variable fields from the keyboard. Example 9.3 shows this.

```
#include<stdio.h>

int main()
{
    struct Student_Info
    {
        int roll;
        int age;
    };

    struct Student_Info student1, student2;

    printf( "\n Enter roll of student 1: " );
    scanf("%d", &student1.roll );
    printf( "\n Enter age of student 1: " );
    scanf("%d", &student1.age );

    printf( "\n Enter roll of student 2: " );
    scanf("%d", &student2.roll );
    printf( "\n Enter age of student 2: " );
    scanf("%d", &student2.age );

    printf( "Roll and age of the students are: \n ");
```

```
        printf( "Student 1: Roll: %d  Age: %d", student1.roll, student1.age );
        printf( "\n Student 2: Roll: %d  Age: %d", student2.roll, student2.age );

        return 0;
}
```

Example 9.3 Another C program demonstrating a simple structure.

9.4 ACCESSING STRUCTURE MEMBERS WITH POINTERS

We have learned about pointers in Chapter 8. We used them with built-in data types (int, char, float). Now we will use pointers with the user-defined data type structure to access the members.

We have already seen use of a dot operator for accessing the members of the structure. When we use a pointer, we need to use **-> operator.**

If age is a member of a structure pointed by a pointer ptr, then we use **ptr -> age** to access the member age.

Now if we want to display its value, we will use the syntax we are using for other variables. Here we will consider ptr -> age to be a variable. As age is an integer, we use %d specifier as follows.

```
printf( " \n Age is %d", ptr->age );
```

Similar is the case for scanning values. The following statement will scan some integer values from the keyboard to the structure variable age.

```
scanf( "%d", &ptr->age );
```

Example 9.4 shows a simple C program where a pointer is used to scan and display the member variables of a structure Student_Info. We see that a pointer variable of type struct Student_Info is declared in line number 10. Then this pointer is made to point to a structure variable student (declared in line number 9) in line number 11.

Once the pointer ptr points to the structure variable, we use the notations discussed above for accessing the members of that structure.

```
#include<stdio.h>

int main()
{
        struct Student_Info
        {
                int roll;
                int age;
        };

        struct Student_Info student;
        struct Student_Info *ptr;

        ptr = &student;

        printf( " \n Enter roll of the student: " );
        scanf(" %d", &ptr->roll );

        printf( " \n Enter age of the student: " );
        scanf( "%d", &ptr->age );

        printf( "\n Entered details are as follows: \n \n " );

        printf( " \n Roll of the student is %d", ptr->roll );
        printf( " \n Age of the student is %d", ptr->age );

        return 0;
}
```

Example 9.4 C program demonstrating the use of a pointer with a structure.

Single pointer for multiple structure variables

We have already learned in Chapter 8 that we can use a single pointer to point to different variables at different instances of time. Here, we will do the same for structure variables.

In the program of Example 9.4, a single pointer was used to access the member variables of a structure variable student. If we have multiple structure variables (that is, more than one student record), we can use the same pointer for accessing their values.

Whenever we want to access the members of a structure variable, we need to make the pointer point to that specific variable. Thereafter, we can use them seamlessly.

Example 9.5 shows a C program where a single pointer *ptr is used to point to two structure variables student1 and student2. The pointer first points to student1 and it scans values from the keyboard. Then, it is made to point to student2. So it scans values for the next variable.

To display the student record, we do the same thing. First the pointer points to student1 (line number 22) and displays its member variables. Then, it points to student2 (line number 25) and displays its member variables.

```
#include<stdio.h>

int main()
{
        struct Student_Info
        {
                int roll;
                int age;
        };

        struct Student_Info student1, student2;
        struct Student_Info *ptr;

        ptr = &student1; // pointer points to student1

        printf( " \n Enter roll of the student 1: " );
        scanf(" %d", &ptr->roll );

        printf( " \n Enter age of the student 1: " );
        scanf( "%d", &ptr->age );

        ptr = &student2; // pointer points to student2

        printf( " \n Enter roll of the student 2: " );
        scanf(" %d", &ptr->roll );

        printf( " \n Enter age of the student 2: " );
        scanf( "%d", &ptr->age );
```

```
        printf( "\n Entered details are as follows: \n \n " );

        ptr = &student1; // pointer ptr points to student1 again
        printf( " \n Roll of the student 1 is %d", ptr->roll );
        printf( " \n Age of the student 1 is %d", ptr->age );

        ptr = &student2; pointer ptr now points to student2
        printf( " \n Roll of the student 2 is %d", ptr->roll );
        printf( " \n Age of the student 2 is %d", ptr->age );

        return 0;
}
```

Example 9.5 C program demonstrating the use of a single pointer with multiple structure variables.

In the above program, we see that there is no difference in the signatures while accessing a member variable for different structure variables. For instance, to access roll numbers, we use **ptr -> roll** for both **student**1 and **student2.**

9.5 PROBLEM SOLVING USING STRUCTURES

In this section, we will write programs for some common scenarios such as maintaining student record, employee record, book record, etc.

A. Storing student information with two integer fields

Let us first extend the program of Example 9.3 to store information of more than two numbers of students. We have already seen earlier that when we need to deal with more variables of the same type, we use array. We do the same thing here.

We declare an array of the variables of the newly defined structure type. Then we treat them as normal array variables. Example 9.4 shows the complete C program. The fields of the structure remain the same - roll and age.

```
#include<stdio.h>

int main()
{
```

```
        struct Student_Info
        {
                int roll;
                int age;
        };

        struct Student_Info student[10];
        int studentCount;

        printf( "\n How many students?  " );
        scanf( "%d", &studentCount );

        for ( int index = 0; index < studentCount; index++ )
        {
                printf( " \n Enter roll for student %d ", index );
                scanf(" %d", &student[index].roll );

                printf( " \n Enter age for student %d ", index );
                scanf( "%d", &student[index].age );

        }

        printf( "\n Entered details are as follows: \n \n " );

        for ( int index = 0; index < studentCount; index++ )
        {
                printf( " \n Roll for student %d is %d", index, student[index].roll );
                printf( " \n Age for student %d is %d", index, student[index].age );

        }

        return 0;
}
```

Example 9.6 A C program demonstrating an array of structure variables.

After declaring an array of sufficient size (line number 9), we asked the user to enter the number of students. Then, we used a for loop to access each field of every variable and to store values in them.

Here the structure variables are **student[0]**, **student[1]**, **student[2]**, etc. Each variable has two fields in it. For instance, student[0] has two fields - accessed as **student[0].roll** and **student[0].age.** Similarly, student[1] has two fields - accessed as **student[1].roll** and **student[1].age.**

In general, the fields of student[index] are to be accessed as student[index].roll and student[index].age. In the program, we used these notations to scan input in them and to display their values.

B. Student information with Name, Roll and Age

In the last example, we considered a structure that has only integer type of member variables. In this problem, we consider a more realistic situation where we will keep information about student name, roll number and age.

Roll number and age will remain as int type whereas name will be a sequence of characters (called as string - we introduced in Chapter 6).

The program is presented using Example 9.7. In this program, we have added a new field **(name)** in the definition of the structure Student_Info. So we are to take input in this new field. We have taken input using a gets() function.

```
#include<stdio.h>

int main()
{
        struct Student_Info
        {
                char name[30];
                int roll;
                int age;
        };

        struct Student_Info student[10];
        int studentCount;

        printf( "\n How many students?  " );
        scanf( "%d", &studentCount );

        for ( int index = 0; index < studentCount; index++ )
```

```
        {
                printf( " \n Enter name for student %d ", index );
                getchar();
                gets( student[index].name );

                printf( " \n Enter roll for student %d ", index );
                scanf(" %d", &student[index].roll );

                printf( " \n Enter age for student %d ", index );
                scanf( "%d", &student[index].age );

        }

        printf( "\n Entered details are as follows: \n \n " );

        for ( int index = 0; index < studentCount; index++ )
        {
                printf("\n\n Student %d: ", index+1 );

                printf("\n Name: ");

                puts( student[index].name );

                printf( " Roll: %d", student[index].roll );

                printf( " \n Age: %d", student[index].age );

        }

        return 0;
}
```

Example 9.7 Another C program demonstrating an array of structure variables.

In order to discard the previously scanned newline character from the input buffer, we have used a getchar() function before gets(). The remaining statements for input remain the same as Example 9.6.

After getting the inputs in the structure variables, we again run through a for loop to display the values stored in them. This time, to display the name of a student, we have used the puts() function. To display the other two fields, we have used printf() statements.

C. Keeping book information

In this problem, we are supposed to keep information about books. The information may contain the following.

- Title of the book
- Book ID
- Number of available copies
- Author of the book

The following table lists the book information and the corresponding structure member and data type.

Table 9. List of structure fields with their data types

Book information	Structure member	Member data type
Title of the book	title	Sequence of char
Book ID	bookID	int
Number of available copies	noOfCopies	int
Author of the book	author	Sequence of char

The program is given in Example 9.8. The look and the flow of the program is the same as the last one. We first declare a structure with appropriate members. Then we ask for the total number of books the user of the program wants to work on. Then, using the first for loop, we scan input from the keyboard. Using the second for loop, we display the values stored in the variables.

```
#include<stdio.h>

int main()
{
        struct Book_Info
        {
                char title[30];
                int bookID;
                int noOfCopies;
                char author[30];

        };

        struct Book_Info book[10];
        int totalBookCount;
```

```
        printf( "\n How many books in total?  " );
        scanf( "%d", &totalBookCount );


        for ( int index = 0; index < totalBookCount; index++ )
        {
                printf( " \n Enter title of the book %d ", index );
                getchar();
                gets(book[index].title);

                printf( " \n Enter Book ID %d ", index );
                scanf(" %d", &book[index].bookID );


                printf( " \n Enter total no of copies %d ", index );
                scanf(" %d", &book[index].noOfCopies );


                printf( " \n Enter author of the book %d ", index );
                getchar();
                gets(book[index].author);

        }

        printf( "\n Entered details are as follows: \n \n " );

        for ( int index = 0; index < totalBookCount; index++ )
        {
                printf("\n Book %d \n", index);
                printf(" Title: ");
                puts( book[index].title );
                printf( " ID: %d \n Available copies: %d ", book[index].bookID,
                book[index].noOfCopies );
                printf(" \n Author: ");
                puts( book[index].author );

        }

        return 0;
}
```

Example 9.8 A C program for handling information about books.

Similar to the previous program, for scanning char type inputs, we used gets() function and to display them, we used puts() function.

Here the individual members of the structure Book_Info are accessed as book[index].title, book[index].bookID, book[index].noOfCopies, and book[index].author where book[index] refers to a variable of that structure. We may also refer to it as one **record**. Following table shows some book records with some dummy values in the fields.

Table 9.2 Example of some book records

Title	Book ID	Available copies	Author
Adhalekha Dastabej	1	97	Dr. Mamoni Raisom Goswami
Halodhia soraye Baodhan Khay	2	65	Dr. Homen Borgohaien
Udangnifrai Gidingfinnanwi	3	89	Urkhao Gwra Brahma
Rongmilir Hanhi	4	90	Rongbong Terang
Short-Term Income Determination	5	70	Dr. B B Bhattacharya

Quick activity: Modify the above program to add two more fields in the book information - publisher name and year of publication.

D. Keeping book information - with an issue operation

In the previous problem, we have stored information about some books using structure. In this problem, let us consider the situation when we issue a book.

In case of issuing a book, we first need to know which book is to be issued. This is identified by the book ID field. Book ID is considered to be the primary key here (we have already studied the concept of primary key in the Database chapter).

Thus, to issue a book, we first ask for the ID of the book. Then we run through the records to see whether a book record exists with this book ID. We use a if clause in the program to check this.

```
if ( book[index].bookID = = id )
{
        . . .
}
```

If the condition becomes true, we know that there exists a book record with this book ID. In that case, we need to decrease the available quantity of that book by one. We perform the following operation in the program.

```
book[index].noOfCopies = book[index].noOfCopies - 1;
```

The complete C program is presented in Example 9.9. We can see that the structure is declared as a global variable along with the total number of records. We have used three user-defined functions for the following purposes.

Function name	Purpose
input()	To take input to the variables from the keyboard
output()	To display the records
issue()	To perform an issue operation for a book

As we have declared the structure as global, all the functions can access it. We have called the issue() function only once. This means that issue operation is performed only once for a book.

We have called the output() function from the issue() function (line number 10 of the function definition) to display the total book record after decreasing the available quantity of the book.

After we finished our work, we wrote a return statement to come out from the function (line number 11 of the function definition). We can do so because we have already found the book record and we performed the issue operation. If we do not return, the loop will unnecessarily iterate till the last record.

```
#include<stdio.h>

struct Book_Info
{
        char title[30];
        int bookID;
        int noOfCopies;
        char author[30];

};

struct Book_Info book[10];
int totalRecord;

void input()
{
        for ( int index = 0; index < totalRecord; index++ )
        {
```

```
            printf( " \n Enter title of the book %d ", index );
            gets(book[index].title);

            printf( " \n Enter Book ID %d ", index );
            scanf(" %d", &book[index].bookID );


            printf( " \n Enter total no of copies %d ", index );
            scanf(" %d", &book[index].noOfCopies );


            printf( " \n Enter author of the book %d ", index );
            getchar();
            gets(book[index].author);

        }
}

void output()
{
        printf( "\n Entered details are as follows: \n \n " );

        for ( int index = 0; index < totalRecord; index++ )
        {
            printf("\n Book %d \n\n", index);
            printf(" Title: ");
            puts(book[index].title);
            printf( " ID: %d \n Remaining copies: %d \n", book[index].bookID,
            book[index].noOfCopies );
            printf(" Authors: ");
            puts(book[index].author);

        }
}

void issue()
{
        int id;

        printf("\n Enter the ID of the book to issue: ");
        scanf("%d", &id);

        for ( int index = 0; index < totalRecord; index++ )
        {
```

```
                if( book[index].bookID == id )
                {
                        book[index].noOfCopies = book[index].noOfCopies - 1;

                        printf("\n One book issued with ID: %d", id);

                        output();

                        return;
                }
        }

}

int main()
{
        printf( "\n How many books in total?  " );
        scanf( "%d", &totalRecord );
        getchar();

        input();

        output();

        issue();

        return 0;
}
```

Example 9.9 A C program for handling information about books with an issue operation.

The following table shows the record details after we issue one book 'Rongmilir Hanhi'. That is, we perform an issue operation by calling function issue() for the book ID 4. Number of available copies of the book 'Rongmilir Hanhi' decreases by 1 (shown in red bold).

Table 9.3 Example of the book records after performing an issue operation for book ID 4

Title	Book ID	Available copies	Author
Adhalekha Dastabej	1	97	Dr. Mamoni Raisom Goswami
Halodhia Soraye Baodhan Khay	2	65	Dr. Homen Borgohaien
Udangnifrai Gidingfinnanwi	3	89	Urkhao Gwra Brahma
Rongmilir Hanhi	4	**89**	Rongbong Terang
Short-Term Income Determination	5	70	Dr. B B Bhattacharya

E. Keeping book information - with a return operation

In the last problem, we have considered issuing a book. In this problem, we will consider returning a book. Returning a book means someone had taken the book earlier and now the book is being returned back to the library.

Returning a book is similar to issuing. The only difference is that the available book copies will increase here instead of decrease.

We present the C program in Example 9.10. We see that all the portions of the program are the same except the returnBook() function. Accordingly, instead of calling issue() function, we called returnBook() function (line number 6 in main() function).

```
#include<stdio.h>

struct Book_Info
{
        char title[30];
        int bookID;
        int noOfCopies;
        char author[30];

};

struct Book_Info book[10];
int totalRecord;

void input()
{
        for ( int index = 0; index < totalRecord; index++ )
        {
                printf( " \n Enter title of the book %d ", index );
                gets(book[index].title);

                printf( " \n Enter Book ID %d ", index );
                scanf(" %d", &book[index].bookID );


                printf( " \n Enter total no of copies %d ", index );
                scanf(" %d", &book[index].noOfCopies );
```

```
                    printf( " \n Enter author of the book %d ", index );
                    getchar();
                    gets(book[index].author);

            }
        }

        void output()
        {
            printf( "\n Entered details are as follows: \n \n " );

            for ( int index = 0; index < totalRecord; index++ )
            {
                    printf("\n Book %d \n\n", index);
                    printf(" Title: ");
                    puts(book[index].title);
                    printf( " ID: %d \n Remaining copies: %d \n", book[index].bookID,
                    book[index].noOfCopies );
                    printf(" Authors: ");
                    puts(book[index].author);

            }
        }

void returnBook()
{
        int id;

        printf("\n Enter the ID of the book to return: ");
        scanf("%d", &id);

        for ( int index = 0; index < totalRecord; index++ )
        {
            if( book[index].bookID = = id )
            {
                book[index].noOfCopies = book[index].noOfCopies + 1;

                printf("\n One book is returned with ID: %d", id);

                output();
                return;
```

```
            }
        }
}

int main()
{
        printf( "\n How many books in total?  " );
        scanf( "%d", &totalRecord );
        getchar();

        input();

        output();

        returnBook();

        return 0;
}
```

Example 9.10 A C program for handling information about books with an issue operation.

The following table shows the record details when we perform a return operation for book ID 1. Number of available copies of the book 'Adhalekha Dastabej' increases by 1 (shown in red bold).

Table 9.3 Example of the book records after performing an issue operation for book ID 4

Title	Book ID	Available copies	Author
Adhalekha Dastabej	1	**98**	Dr. Mamoni Raisom Goswami
Halodhia soraye Baodhan Khay	2	65	Dr. Homen Borgohaien
Udangnifrai Gidingfinnanwi	3	89	Urkhao Gwra Brahma
Rongmilir Hanhi	4	89	Rongbong Terang
Short-Term Income Determination	5	70	Dr. B B Bhattacharya

F. Keeping book information - a mini library system

In the last two problems, we have seen the issue and return of books. In this problem, we aim to design an application that can support both issue and return. The application will keep on running and will prompt the user to enter options. Based on the option entered by the user, the application will perform the designated task. The following figure demonstrates one such running instance of the application.

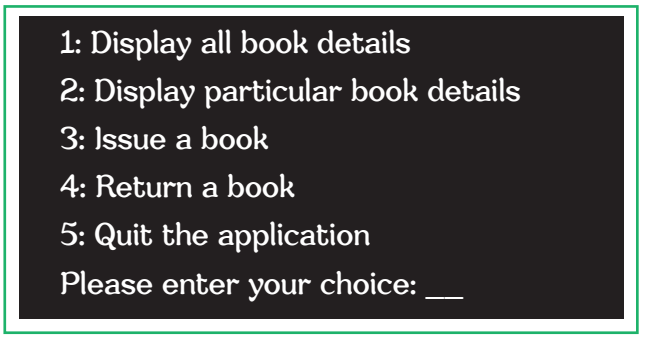


Figure 9.2: A running instance of the required application.

Now, when we want some statements to run continuously, we can use a while (1) loop as follows.

```
while (1)
{

    . . .

}
```

We need to place the options (as shown in Figure 9.2) inside the while loop so that they appear everytime. Once the user enters an option, we need to perform the required task associated with that option. For instance, the user enters 1, then we need to display all the records.

We can use a switch-case construct for this. In each case of the switch-case, we can call the appropriate function to do the required task. The following code segment shows this clearly.

We call the function displayAll() in case 1 and this will be executed when the user enters option 1. Similarly, when the user enters 4, it belongs to case 4 and we are calling the function returnBook().

```
while (1)
{
        printf(" \n 1: Display all book details" );
        printf(" \n 2: Display particular book details" );
        printf(" \n 3: Issue a book" );
        printf(" \n 4: Return a book" );
        printf(" \n 5: Quit the application" );
        printf(" \n Please enter your choice: " );
        scanf( "%d", &option );
```

```
        switch (option)
        {
        case 1:
                displayAll();
                break;

        case 2:
                displayOne();
                break;

        case 3:
                issueBook();
                break;

        case 4:
                returnBook();
                break;

        case 5:
                return 0;


        default:
                printf(" \n Invalid choice! ");

}
```

We present the complete C program in Example 9.11. All other portions of the program are similar to the previous ones.

```
#include<stdio.h>

struct Book_Info
{
        char title[30];
        int bookID;
        int noOfCopies;
        char author[30];
```

```
};

struct Book_Info book[10];
int totalRecord;

void input()
{
        for ( int index = 0; index < totalRecord; index++ )
        {
                printf( " \n Enter title of the book %d ", index );
                gets(book[index].title);

                printf( " \n Enter Book ID %d ", index );
                scanf(" %d", &book[index].bookID );


                printf( " \n Enter total no of copies %d ", index );
                scanf(" %d", &book[index].noOfCopies );


                printf( " \n Enter author of the book %d ", index );
                getchar();
                gets(book[index].author);

        }
}

void displayAll()
{
        printf( "\n The library has the following books: \n \n " );

        for ( int index = 0; index < totalRecord; index++ )
        {
                printf("\n Book %d \n\n", index);
                printf(" Title: ");
                puts(book[index].title);
                printf( " ID: %d \n Remaining copies: %d \n", book[index].bookID,
                book[index].noOfCopies );
                printf(" Authors: ");
                puts(book[index].author);
```

```
        }
}

void displayOne()
{
        int id;
        printf("\n Enter the ID of the book to issue: ");
        scanf("%d", &id);

        for ( int index = 0; index < totalRecord; index++ )
        {
                if( book[index].bookID == id )
                {
                        printf(" Title: ");
                        puts(book[index].title);
                        printf( " ID: %d \n Remaining copies: %d \n", book[index].bookID,
                        book[index].noOfCopies );
                        printf(" Authors: ");
                        puts(book[index].author);
                }
        }

}

void issueBook()
{
        int id;

        printf("\n Enter the ID of the book to issue: ");
        scanf("%d", &id);

        for ( int index = 0; index < totalRecord; index++ )
        {
                if( book[index].bookID = = id )
                {
                        book[index].noOfCopies = book[index].noOfCopies - 1;

                        printf("\n One book is returned with ID: %d", id);

                        return;
```

```
                }
            }

    }

    void returnBook()
    {
        int id;

        printf("\n Enter the ID of the book to return: ");
        scanf("%d", &id);

        for ( int index = 0; index < totalRecord; index++ )
        {
            if( book[index].bookID = = id )
            {
                book[index].noOfCopies = book[index].noOfCopies + 1;

                printf("\n One book is returned with ID: %d", id);

                return;
            }
        }
}

int main()
{
    printf( "\n How many books in total?  " );
    scanf( "%d", &totalRecord );
    getchar();

    input();

    int option;

    while (1)
    {
        printf(" \n 1: Display all book details" );
        printf(" \n 2: Display particular book details" );
        printf(" \n 3: Issue a book" );
```

```
            printf(" \n 4: Return a book" );
            printf(" \n 5: Quit the application" );
            printf(" \n Please enter your choice: " );
            scanf( "%d", &option );

            switch (option)
            {
            case 1:
                        displayAll();
                        break;

            case 2:
                        displayOne();
                        break;

            case 3:
                        issueBook();
                        break;

            case 4:
                        returnBook();
                        break;

            case 5:
                        return 0;

            default:
                        printf(" \n Invalid choice! ");

            }
      }

      return 0;
}
```

Example 9.11 A C program for a mini library application.

Quick activity: Extend the program of to handle some unusual conditions such as a user wants to issue a book but (i) it is not there in the library or (ii) the available quantity of the book is 0.

9.6 DYNAMIC MEMORY ALLOCATION FOR STRUCTURES

We have already learned about dynamic memory allocation in chapter 8. Here, we wish to apply the concept for structure. The primary objective of dynamic memory allocation is to allocate memory space to a pointer while the program runs.

In case of a structure, depending on the number of records the user has, we can allocate memory space. The following statement allocates space to the pointer *ptr for storing N structure records of type Student_Info.

```
ptr = (struct Student_Info*) malloc ( N * sizeof(struct Student_Info) );
```

Once the required space is allocated to the pointer, we can use it as a normal array as we did in Chapter 8. We may use either array subscripting (uses [ ]) or pointer base address (uses (ptr + index)) for accessing array elements. We rewrite the program of Example 9.6 using dynamic memory allocation and present in Example 9.12.

```
#include<stdio.h>

int main()
{
        struct Student_Info
        {
                int roll;
                int age;
        };

        struct Student_Info *ptr;
        int studentCount;

        printf( "\n How many students?  " );
        scanf( "%d", &studentCount );

        ptr = (struct Student_Info*) malloc ( studentCount * sizeof(struct Student_Info) );

        for ( int index = 0; index < studentCount; index++ )
        {
                printf( " \n Enter roll for student %d ", index );
                scanf(" %d", &(ptr+index)->roll );
```

```
                printf( " \n Enter age for student %d ", index );
                scanf( "%d", &(ptr+index)->age );

        }

        printf( "\n Entered details are as follows: \n \n " );

        for ( int index = 0; index < studentCount; index++ )
        {
                printf( " \n Roll for student %d is %d", index, (ptr+index)->roll );
                printf( " \n Age for student %d is %d", index, (ptr+index)->age );

    }

    return 0;
}
```

Example 9.12 Program in C to demonstrate dynamic memory allocation of structure variables.

The program allocates memory space for studentCount number of structure variables. Size of a structure variable is the size of the member variables of that structure.

Quick activities

1. Extend the above program to include student names in the structure.
2. Rewrite all the programs from Example 9.7 to 9.11 using dynamic memory allocation strategy.

Exercise

1. Define structure in the context of a programming language. How is it different from an array?

2. Is structure a built-in data type? Can we apply basic arithmetic operations such as addition, subtraction to structure variables? Show with a simple C program.

3. Identify the members of the structure from the below code segment.

```
struct Account
{
        char acNo[15] ;
        char ifsc[15];
        char acType[7];
        double balance;
        double minBalance;
};

struct Account account1, account2, account3, account[10];
```

4. Identify the structure variables from the below code segment.

```
struct Account
{
        char acNo[15] ;
        char ifsc[15];
        char acType[7];
        double balance;
        double minBalance;
};

struct Account account1, account2, account3, account[10];
```

5. Consider the structure below and write statements for the following.

a. to declare a variable of the structure

b. to display the age of the teacher

```
struct Teacher
{
        char name[30] ;
        int age;
};
```

6. Declare a pointer for structure Teacher (from Q No. 5) and dynamically allocate memory for 10 records.

7. Consider the structure below and write statements for the following.

a. to declare a pointer for the above structure and display the salary.

b. to declare a single pointer for two different variables of the higher structure and display the details of the employee whose salary is more.

```
struct Employee
{

        char name[30];
        double salary;
};
```

8. Rewrite the program of Q. No. 7 to facilitate dynamic memory allocation for N number of record where N in a user input.

9. Consider the below structure and design a simple banking system that supports the following operations.

a. opening of accounts (Hint: input to the variables)

b. displaying details based on account number

c. displaying all account details

d. displaying details of all accounts whose balance is more than 1000

e. depositing an amount to an account

f. withdrawing some amount from an account

```
struct Account
 {
        char acNo[15] ;
        char ifsc[15];
        char acType[7];
        double balance;
        double minBalance;
};
```

CHAPTER 10

An Introduction to Object Oriented Programming

In this chapter

You will learn what is programming paradigm. You will learn about procedural programming, important features of the procedural programming, its advantages and disadvantages. This chapter gives an introduction to OOP also. We will also discuss the important fundamental features of OOP, advantages and disadvantages of OOP. We will also discuss about the presence of java programming language in today's world.

10.1 PROGRAMMING PARADIGM

In the context of programming languages, the term paradigm means set of design principles that defines the program structure. It is an approach to programming.

A problem can be solved in many different ways or approaches. A programmer can evaluate different approaches in order to select the best option. So, a computer program can be developed using different programming languages that belong to different paradigms. Each approach to the solution will have some merits and demerits.

The programming paradigm is all about the writing style and organizing the program code in a specific way. It is framework that defines how the programmer can conceptualize and model complex problem to be solved. The computer programs are written using different programming languages. Each programming language has unique programming style that implements a specific programming paradigm. For example C language follows procedural programming paradigm. Whereas the C++, python and java are said to be object oriented programming paradigm.

There are many programming paradigms out of which the two most important programming paradigms are the Procedure oriented Programming and object Oriented Programming.

10.2 PROCEDURE ORIENTED PROGRAMMING (POP)

Procedure oriented programming can be defined as a programming model which is based upon the concept of calling procedure. Procedures, also known as routines, subroutines or functions, simply consist of a series of computational steps to be carried out. During a program's

execution, any given procedure might be called at any point, including by other procedures or itself. The procedural programming aims at dividing the large program into smaller programs called procedures.

Some of the popular procedural languages include C, FORTRAN, Pascal, BASIC, COBOL, ALGOL, COBOL.

In procedural programming, the program code is organized as set of procedures called functions. These functions operate on the program data called the variables. Each function consists of computational statements and solves a part of the problem. These functions can be called many times depending upon the algorithm of the program to solve a particular problem. For example, a program may involve collecting data from user, performing some kind of calculation on that data and printing the data on screen when is requested. Calculating, reading or printing can be written in a program with the help of different functions on different tasks.

POP is related with the conventional style. This approach is also known as the top-down approach. In this approach, a program is divided into functions that perform specific tasks. This approach is mainly used for medium-sized applications. Data is global, and all the functions can access global data. The basic drawback of the procedural programming approach is that data is not secured because data is global and can be accessed by any function. Program control flow is achieved through function calls and go to statements.

Procedure oriented programming basically contains multiple functions into the program. In below figure we have divided our program into multiple functions. We use flowchart or algorithm to show how the program is executed from one instruction to other instruction. It does not emphasise on the data.

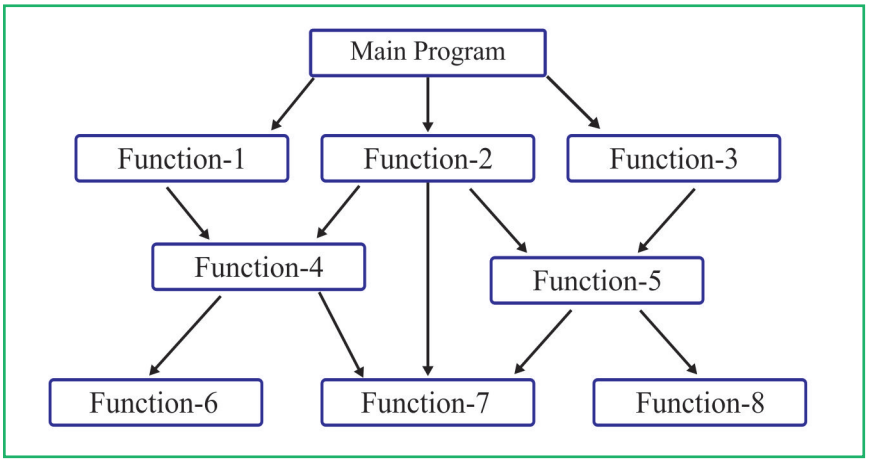


Figure : 1

In a multi-function program we use global variable to communicate between two functions. Global variable can be use by any function at any time while local variables are only used within

the function. But it creates problem in large program because we can't determine which global variables (data) are used by which function.

Also global variables are accessed by all the function so any function can change its value at any time so all the function will be affected.

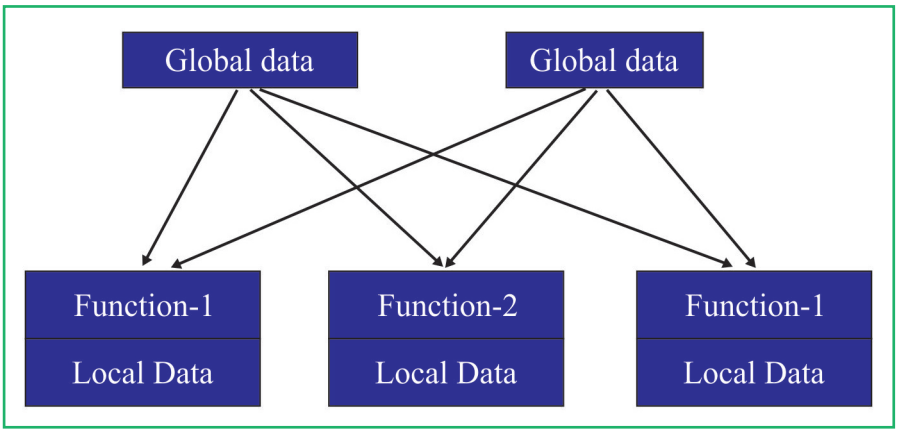


Figure : 2

10.3 CHARACTERISTICS OF PROCEDURE ORIENTED PROGRAMMING

a. It emphasis on algorithm.
b. Large programs are divided into smaller programs known as functions.
c. Function can communicate by global variable.
d. Data move freely from one function to another function.
e. Functions change the value of data at any time from any place. (Functions transform data from one form to another.)
f. It uses top-down programming approach.
g. Most procedural programming language offer extensive library of predefined functions.

10.4 ADVANTAGES AND DISADVANTAGES OF PROCEDURE ORIENTED PROGRAMMING

10.4.1 Advantages:

a. The procedural programming languages are relatively much easier to learn as first programming language for beginners.
b. The straight forward program organization makes it ideal choice as a general purpose language.
c. The use of standard library functions brings down significant reduction in the overall development cost and time.

10.4.2 Disadvantages:

a. The procedural programming is not suitable for large and complex software.
b. It is difficult to represent the real world objects.
c. It is difficult to protect the data from inadvertent changes.
d. The software maintenance is relatively difficult for a procedural programming software.
e. In procedural programming, functions are the most important components of the program and the data does not get the attention.

10.5 OBJECT ORIENTED PROGRAMMING (OOP)

The object oriented programming is an approach to the programming that is based on the concept of object and class. Objects contain data in the form of attributes and code in the form of methods. In object oriented programming, computer programs are designed using the concept of objects that interact with real world. The OOP programming paradigm was developed in order to overcome the limitations of the procedural programming. The procedural program structure consists of program data in the form of variables and the functions that operates on the data. The program data in the procedural programming is generally global. The global data can be accessed by any function. It is difficult to protect the data from such inadvertent changes. The unrestricted access to the program data makes it venerable because the data can be easily modified by any function. The object oriented programming approach provides the much needed solution to the limitations of the procedural programming. In OOP programming approach, it is much easier to control the access to the data. The main aim of OOP is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function. It also becomes much easier to model the real world objects in the OOP programming style.

10.5.1 Languages used in Object Oriented Programming:

Java, C++, C#, Python, PHP, JavaScript, Ruby, Perl, Objective-C, Dart, Swift, Scala, MATLAB. Simula is credited as being the first object oriented programming language.

10.6 CHARACTERISTICS OF OOP

i. Emphasis is on data rather than procedure.
ii. Programs are divided into objects.
iii. Data structures are designed such that they characterize the objects.
iv. Functions that operate on data of an object are tied together in the data structure.
v. Data is hidden and cannot be accessed by external functions.
vi. New data and functions can be easily added whenever necessary.
vii. Bottom-up approach is followed in program design.

10.7 BUILDING BLOCKS OF OOP

10.7.1 Object and Class

Object is a basic unit of Object Oriented Programming and represents the real life entities. An object is a thing in real world which has certain properties and methods. It may be any place, person, bank account, bill or any item.

An object consists of:

1. **State:** It is represented by attributes of an object. It also reflects the properties of an object.
2. **Behaviour:** It is represented by methods of an object. It also reflects the response of an object with other objects.
3. **Identity:** It gives a unique name to an object and enables one object to interact with other objects.
4. **Methods:** A method is a collection of statements that perform some specific task and return result to the caller. A method can perform some specific task without returning anything.

A class is a user defined blueprint or prototype from which objects are created. It represents the set of properties or methods that are common to all objects of one type.

10.8 FEATURES OF OOP

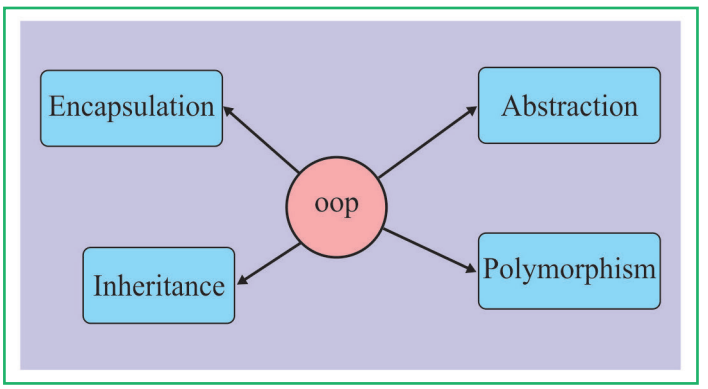


Figure : 3

The four features of object oriented programming are- encapsulation, abstraction, inheritance and polymorphism.

10.8.1 Encapsulation:

The most important feature of object oriented programming is encapsulation.

We can define encapsulation as the method of putting everything that is required to do the job, inside a capsule and presenting that capsule to the user. By encapsulation, all the necessary

data and methods are bind together and all the necessary details are hidden to the normal user. So encapsulation is the process of binding data members and methods of a program together to do a specific job, without revealing unnecessary details. A class is an example of encapsulation. It consists of data and methods that have been bundled into a single unit. Using the method of encapsulation, the programmer cannot directly access the data. Data is only accessible through the functions present inside the class.

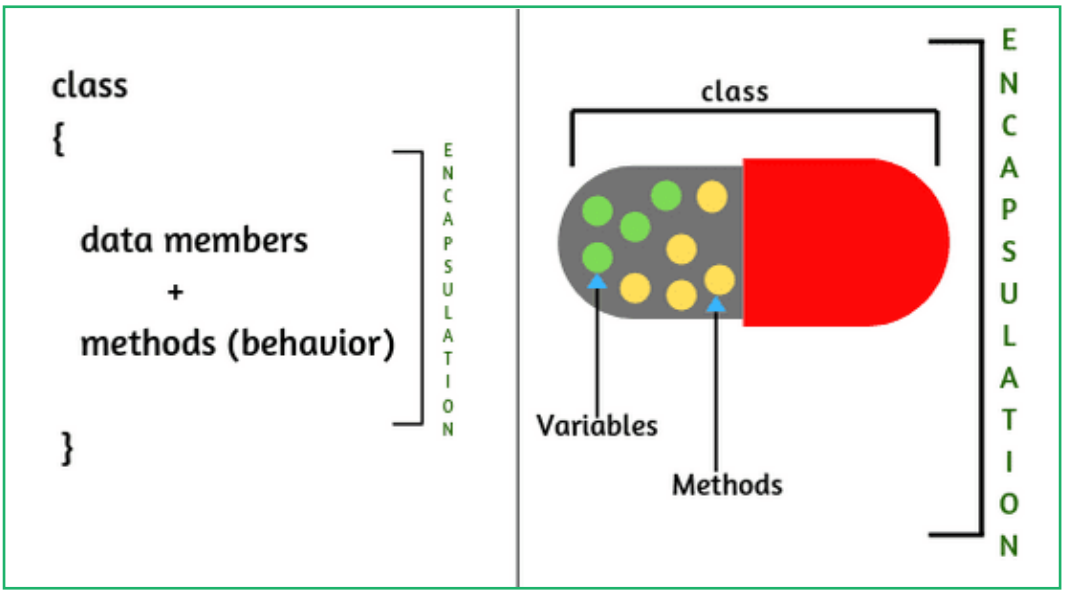


Figure : 4

Encapsulation can also be defined in two ways:

i. Data hiding: Encapsulation is the process of hiding unwanted data, such as restricting access to any member of an object.

ii. Data binding: Encapsulation is the process of binding the data members and the methods together as a whole, as a class.

Encapsulation adds security. Data and methods can be set to private, so they cannot be accessed outside the class. To get information about data in an object, public methods are used to access or update data. This adds a layer of security, where the developer chooses what data can be seen on an object by exposing that data through public methods in the class definition. Within classes, most programming languages have public, protected, and private sections. Public is the limited selection of methods available to the outside world, or other classes within the program. Protected is only accessible to child classes. Private can only be accessed from within that class.

10.8.2 Abstraction:

In abstraction, it displays only the important information by hiding the implementation part. Abstraction is the most important features of OOP. It allows us to show only essential data

or information to the user and hides the implementation details from the user. The concept of abstraction relates to the idea of hiding data that are not needed for presentation. It serves an important security role. By only displaying selected pieces of data, and only allowing data to be accessed through classes and modified through methods, data is protected from exposure.
Example: A real-world example of abstraction is driving a car. When we drive a car, we do not need to know how the engine works.

A driver only uses a small selection of tools. The engineering is hidden from the driver. To make a car work, a lot of pieces have to work under the hood, but exposing that information to the driver would be a dangerous distraction.

10.8.3 Polymorphism:

Poly means many and morphs mean form. So polymorphism means one name multiple forms. Polymorphism refers to a single function or multi-functioning operator performing in different ways.

The most common and simple use of polymorphism in OOP program is use of the same method name. It is possible to define many methods with the same method name but with different implementation. The correct method would be invoked depending on the manner in which the method is invoked.

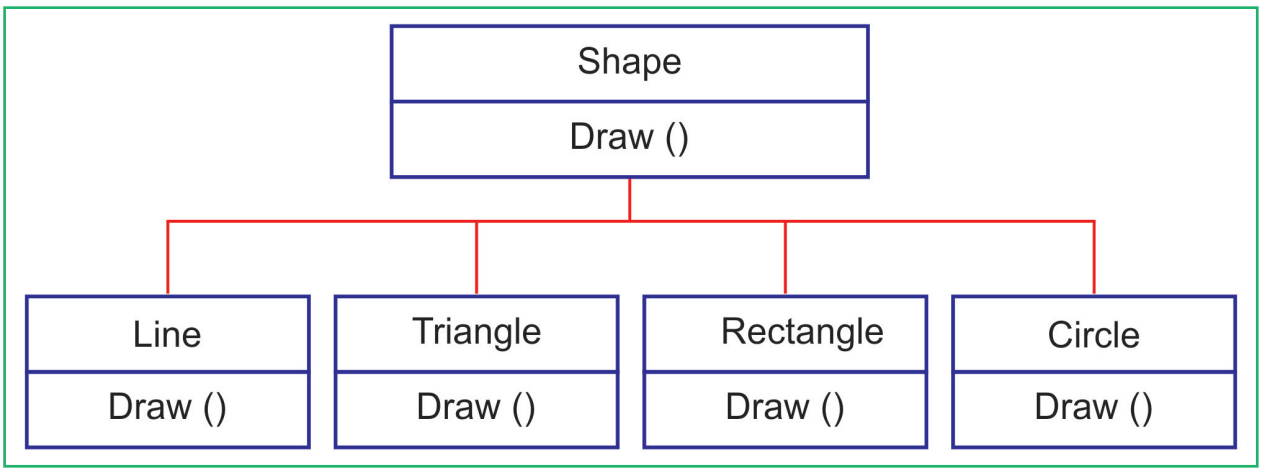


Diagram : 5

The above fundamental is known as the function overloading.

Similar way we can perform operator overloading in which one operator is used for multiple operation. For example '+' is a operator that is used for arithmetic operation the same operator we can use for the concatenation of string and add two structure elements.

The decision to invoke the appropriate method can either happen at compile time or it can happen during the run time. This is referred as compile time polymorphism and runtime polymorphism respectively.

10.8.4 Inheritance:

Inheritance is the process of acquiring or inheriting properties of one class to the other. The process of deriving one class from an existing class is called inheritance. The existing class is called the parent class and the new class is called the child class.

The inheritance is one of the most important fundamental principles in the oop programming. The inheritance basically deals with the re-usability of the program code.

By using inheritance, the programmer makes use of the existing code already previously written. The inheritance helps to reduce the overall development time for the software development.

Take one example of a school. Suppose the school management decided to make software based on the data of students only. The programmer made the software and decided to collect personal details like name, age, sex, address etc. After one year school management decides to incorporate data of teachers to the software. The programmer can add this extension within a small time as he can reuse many of the codes he had written earlier by making use inheritance.

10.9 ADVANTAGES OF OOP

The object oriented programming offers many advantages as compared to procedural programming.

a. OOP can support large scale and complex software development project.
b. OOP offers better data protection. In OOP, the data access permissions are tightly controlled.
c. It allows us the code re-usability.
d. Better representation of real world objects. The programmer can easily represent real world objects realistically into the program code.
e. Avoids unnecessary data exposure to the user by using abstraction.
f. Better software maintenance.
g. Enhance security.
h. Easy code modification. Easy redesign and extension of code that does not affect the other functionality.
i. Polymorphism offers lot of flexibility in OOPs.

10.10 DISADVANTAGES OF OOP

a. With OOP, classes tend to be overly generalised.
b. The OOP program design is tricky.

c. One needs to do proper planning and proper design for oop programming.

d. Programmer should be well skilled.

e. OOPs programs are bigger when compared to other programs. The length of the programmes developed using OOP language is much larger than the procedural approach. Since the programme becomes larger in size, it requires more time to be executed that leads to slower execution of the programme.

f. Takes more time to solve problems.

10.11 DIFFERENCE BETWEEN PROCEDURAL PROGRAMMING AND OBJECT ORIENTED PROGRAMMING

Let us now understand the difference between the procedural programming and object oriented programming.

Procedural Programming	Object Oriented Programming
Program is divided into objects	Program is divided into functions.
Bottom-up approach.	Top-down approach.
It uses access specifier.	It doesn't use access specifier.
Procedural programming does not have any proper way for hiding data so it is ***less secure.***	Object oriented programming provides data hiding so it is ***more secure.***
Inheritance property is used.	Inheritance is not allowed.
Encapsulation is used to hide the data.	No data hiding.
It is difficult to represent the real world objects.	It is much easier to represent the real world objects in OOP.
Adding new data and functions is not easy.	Adding new data and function is easy.
Overloading is not possible.	Overloading is possible.
Function is more important than data.	Data is more important than function.

KEYWORDS LEARNED IN THIS CHAPTER

PARADIGM POP OOP OBJECT CLASS

ENCAPSULATION ABSTRACTION INHERITANCE

POLYMORPHISM FUNCTION OVERLOADING OPERATOR OVERLOADING

Exercise:

I. SHORT ANSWER QUESTIONS:

a. What do you mean by programming paradigm?
b. Define object.
c. Define class.
d. What is encapsulation?
e. What is data hiding?
f. What is polymorphism?
g. Name four object oriented programming languages.
h. Name two procedure oriented programming languages.
i. Name the first object oriented language.
j. What is abstraction?

II. LONG ANSWER QUESTIONS:

a. Mention four characteristics of procedure oriented programming.
b. Mention two advantages and two disadvantages of procedure oriented programming
c. Explain four features of OOP.
d. List four advantages of OOP.
e. Differentiate between procedure oriented programming and object oriented programming.

CHAPTER 11

Case Studies

1. Deep is a medical student. His professor has asked him to create a document for his thesis and also create a presentation on recent development in medical field for a seminar on different dates in different places.

   But Deep has PC at home but no laptop to carry his presentation for seminar. His professor has told him to provide with the laptop in the seminar. Now, another problem with Deep is that his Office is not working in his PC.

   So, he is looking for the solution to his problems.
   a. Where he would create the document for his thesis?
   b. Where he would create the presentation for the seminar?
   c. What is the benefit of using such platform?
   d. Where he would keep the documents so that he can avail the documents from anywhere (if USB is not working)?
   e. Where he schedules the seminar and get reminders and why?
   f. On the way to catch his flight to the airport, he lost his way. Suggest an App that Deep can use to find the location of the airport.

2. Rohan wants to take his parents for Char Dhams yatra. But he confused how to book tickets and book hotels as there may be rush, and he can't take his age-old parents move here and there.

   His friend has given suggestion for online booking. But Rohan has never used such system.

   Can you help him with the following?

   a. What is advantage of online ticketing?
   b. Why is online booking important?
   c. What are the disadvantages of booking online?
   d. What are requirements of online booking?
   e. He is confused which websites to search for booking online tickets. Can you help him with some website names?

3. Nabanita is fond of buying different kinds of goods. She used to go to different shops and malls to buy things. But she wants some solutions to buy goods sitting at home. Then her friend told her about e-commerce. She likes the idea, but she has some queries. Can you solve her queries?

   a. What is the difference between traditional shopping and ecommerce?
   b. What are the advantages of e-commerce?
   c. Which are the best e-commerce sites?
   d. What are the possible risks of e-commerce websites?
   e. What are the risks of online transactions?

4. Aditya is a freelancer. She used to post different works on the Internet. After that she heard of blog. She had some queries regarding blog. Solve her the following queries.

   a. What exactly is a blog?
   b. How do she start a blog?
   c. Do bloggers make money?
   d. What are the different types of blogs?
   e. How did she choose a topic for her blog?
   f. How she can be a successful blogger?

5. Roshmi is from a middle-class family. She has completed her graduation and started her work in an IT firm. To get promotion in her job, she has to upgrade her academic skills. But she can't do regular courses. So, can you help her how can she upgrade her academic skills. Give complete guidance to her.

*****