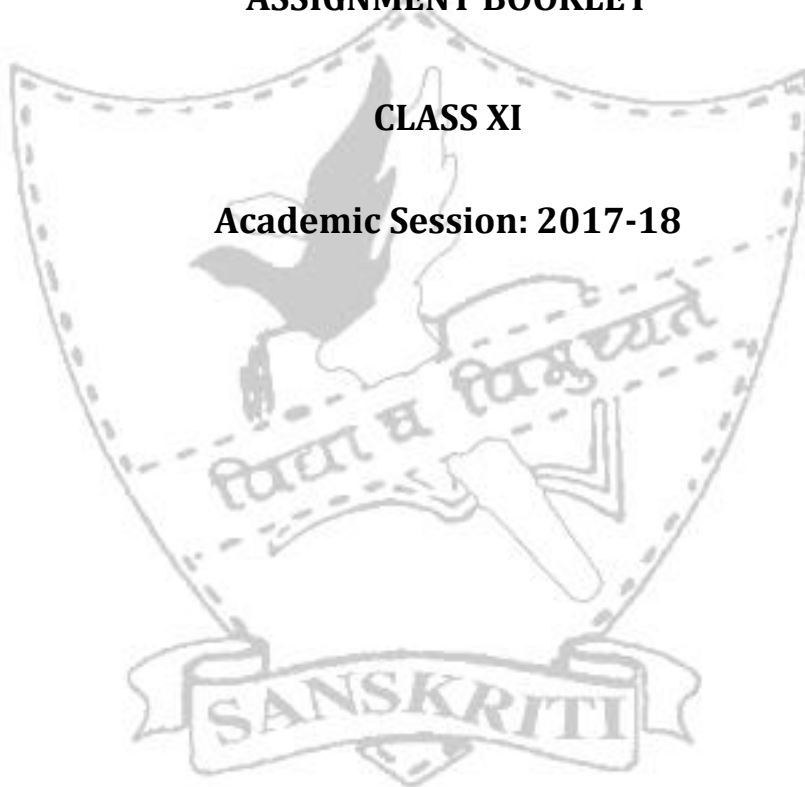


COMPUTER SCIENCE C++(083)

ASSIGNMENT BOOKLET

CLASS XI

Academic Session: 2017-18



Index SheetClass XI

S.No.	Assignment #	Topic	Page #
1.	Assignment No. 1	Computer Organisation	4
2.	Assignment No. 2	Data Representation	6
3.	Assignment No : 3	Getting Started With C++	7
4.	Assignment No : 4	Data Handling	9
5.	Assignment No : 5	Operators and Expressions	10
6.	Assignment No : 6	Flow of Control	12
7.	Assignment No : 7	Arrays	16
8.	Assignment No : 8	Functions	18
9.	Assignment No : 9	Structures	22
10.	Program List No. 1	Getting Started With C++	26
11.	Program List No. 2	Selection Statements and Loops	27
12.	Program List No. 3	Nested Loops	28
13.	Program List No. 4	1-D Arrays and Functions	30
14.	Program List No. 5	2-D Arrays and Structures	32
15.	Project Assignment		33

Computer Science SyllabusClass XI**April**

Chap I, II, IV - Computer Basics
Chap III - Data Representation

May

Chap VI - Getting Started With C++
Chap VII - Data Handling
Chap VIII - Operators and Expressions

July

Chap XIV - Programming Methodology
Chap X - Flow of Control

August

Chap XII - 1-D Arrays
2-D Arrays
Address Calculation in arrays

September

Revision
First Term Examination

October

Sorting and Searching in 1-D Arrays

November

Chap XI - Functions

December

Chap XIII - Structures
Project work

January

Chap V - OOP Concepts

February

Revision

Unit-1

Handout - Computer Fundamentals

Objective:

- ❑ To impart in-depth knowledge of computer related basic terminologies.
- ❑ To inculcate the skills of implementation of basic theory in troubleshooting the software & hardware problems.

What is a Computer?

Computer is an advanced electronic device that takes raw data as input from the user and processes these data under the control of set of instructions (called program) and gives the result (output) and saves output for the future use. It can process both numerical and non-numerical (arithmetic and logical) calculations.

A computer has four functions:

- a. accepts data
- b. processes data
- c. produces output
- d. stores results

Input
Processing
Output
Storage

Input (Data):

Input is the raw information entered into a computer from the input devices. It is the collection of letters, numbers, images etc.

Process:

Process is the operation of data as per given instruction. It is totally internal process of the computer system.

Output:

Output is the processed data given by computer after data processing. Output is also called as Result. We can save these results in the storage devices for the future use.

Computer System

All of the components of a computer system can be summarized with the simple equations. $\text{COMPUTER SYSTEM} = \text{HARDWARE} + \text{SOFTWARE} + \text{USER}$

- Hardware = Internal Devices + Peripheral Devices
All physical parts of the computer (or everything that we can touch) are known as Hardware.
- Software = Programs
Software gives "intelligence" to the computer.
- USER = Person, who operates computer.

Generations of Computers:

First Generation (1940-56):

The first generation computers used **vacuum tubes** & **machine language** was used for giving the instructions. These computers were **large in size** & their programming was difficult task. The **electricity consumption** was very **high**. Some computers of this generation are ENIAC, EDVAC, EDSAC & UNIVAC-1.

Second Generation (1956-63):

In 2nd generation computers, **vacuum tubes were replaced by transistors**. They required only 1/10 of power required by tubes. This generation computers generated less heat & were reliable. The first operating system developed in this generation. Programming in both **machine as well as assembly language**.

The Third Generation (1964-71):

The 3rd generation computers replaced transistors with **Integrated circuit** known as chip. From Small scale integrated circuits which had 10 transistors per chip, technology developed to MSI circuits with 100 transistors per chip. These computers were **smaller, faster & more reliable with lower power consumption**. High level languages invented in this generation.

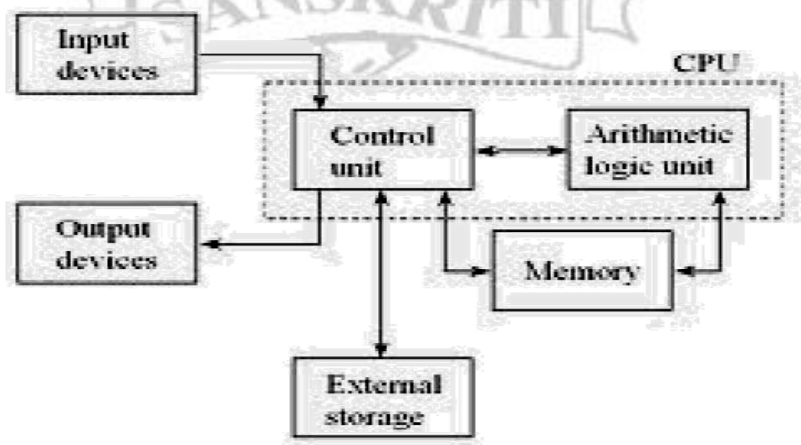
The Fourth Generation (1972- present):

LSI & VLSI were used in this generation. As a result **microprocessors** came into existence. The computers using this technology are known to be Micro Computers. High capacity hard disks were invented. There is great development in **data communication**.

The Fifth Generation (Present & Beyond):

Fifth generation computing devices, based on **artificial intelligence**, are still in development, though there are some applications, such as voice recognition, that are being used today. The use of **parallel processing** and **superconductors** is helping to make artificial intelligence a reality. Quantum computation and molecular and nanotechnology will radically change the face of computers in years to come.

ARCHITECTURE OF COMPUTER



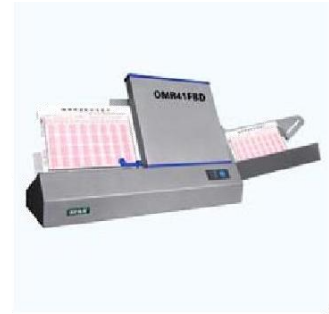
Input Devices: Those devices which help to enter data into computer system. Eg. Keyboard, Mouse, Touchscreen, Barcode Reader, Scanner, MICR, OMR etc.



Bar code Reader
evaluation)



MICR used in Bank



OMR(Used for answer sheet

Output Devices: Those devices which help to display the processed information are called output devices. Eg. Monitor, Printer, Plotter, Projector



Printer



Plotter



Projector

CENTRAL PROCESSING UNIT (CPU)

The main component to make a computer operate is the computer chip or microprocessor. This is referred to as the Central Processing Unit (CPU) and is housed in the computer case. Together, they are also called the CPU. It performs arithmetic and logic operations. The CPU (Central Processing Unit) is the device that interprets and executes instructions.



Types of Computers:

On the basis of working principle

a) Analog Computer

An analog computer is a form of computer that uses *continuous* physical phenomena such as electrical, mechanical, or hydraulic quantities to model the problem being solved.

Eg: Thermometer, Speedometer, Petrol pump indicator, Multimeter



b) Digital Computer

A computer that performs calculations and logical operations with quantities represented as digits, usually in the binary number system.

c) Hybrid Computer (Analog + Digital)

A combination of computers those are capable of inputting and outputting in both digital and analog signals. A hybrid computer system setup offers a cost effective method of performing complex simulations. The instruments used in medical science lies in this category.

On the basis of Size:

a) Super Computer

Supercomputers are the fastest, expensive and are employed for specialized applications that require immense amounts of mathematical calculations such as weather forecasting. Other uses of supercomputers include animated graphics, fluid dynamic calculations, nuclear energy research, and petroleum exploration. PARAM, Pace & Flosolver are the supercomputer made in India.



b) Mainframe Computer

A very large and expensive computer capable of supporting hundreds, or even thousands, of users simultaneously. In some ways, mainframes are more powerful than supercomputers because they support more simultaneous programs. But supercomputers can execute a single program faster than a mainframe.

c) Mini Computer

A mid-sized computer. In size and power, minicomputers lie between *workstations* and *mainframes*. A minicomputer is a multiprocessing system capable of supporting from 4 to about 200 users simultaneously. Generally, servers are comes in this category.

d) Micro Computer

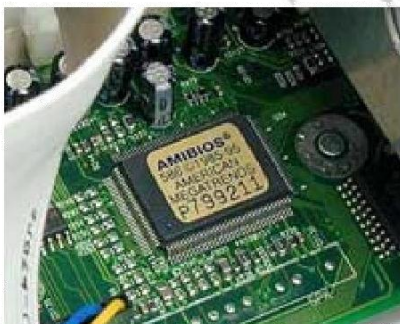
- i. **Desktop Computer:** a personal or micro-mini computer sufficient to fit on a desk.
- ii. **Laptop Computer:** a portable computer complete with an integrated screen and keyboard. It is generally smaller in size than a desktop computer and larger than a notebook computer.
- iii. **Palmtop Computer/Digital Diary /Notebook /PDAs:** a hand-sized computer. Palmtops have no keyboard but the screen serves both as an input and output device.

e) Workstations

A terminal or desktop computer in a network. In this context, workstation is just a generic term for a user's machine (client machine) in contrast to a "server" or "mainframe."



Memory: It facilitates the remembrance power to computer system. It refers to the physical devices used to store programs (sequences of instructions) or data (e.g. program state information) on a temporary or permanent basis for use in a computer or other digital electronic device. The term primary memory is used for the information in physical systems which are fast (i.e. RAM), as a distinction from secondary memory, which are physical devices for program and data storage which are slow to access but offer higher memory capacity. Primary memory stored on secondary memory is called virtual memory. Primary Memory can be categorized as Volatile Memory & Non-Volatile Memory.

Random Access Memory (RAM)

Volatile memory is computer memory that requires power to maintain the stored information. Most modern semiconductor volatile memory is either Static RAM or dynamic RAM.

SRAM retains its contents as long as the power is connected and is easy to interface to but uses six transistors per bit.

Dynamic RAM is more complicated to interface to and control and needs regular refresh cycles to prevent its contents being lost. However, DRAM uses only one transistor and a capacitor per bit, allowing it to reach much higher densities and, with more bits on a memory chip, be much cheaper per bit. SRAM is not worthwhile for desktop system memory, where DRAM dominates, but is used for their cache memories..

Read Only Memory (ROM)

Non-volatile memory is computer memory that can retain the stored information even when not powered. Examples of non-volatile memory are flash memory and ROM/PROM/EPROM/EEPROM memory (used for firmware such as boot programs).

Cache Memory:

Cache memory is random access memory (RAM) that a computer microprocessor can access more quickly than it can access regular RAM. As the microprocessor processes data, it looks first in the cache memory and if it finds the data there (from a previous reading of data), it does not have to do the more time-consuming reading of data from larger memory. It is of two types- L1 cache is on the same chip as the microprocessor. L2 is usually a separate static RAM (SRAM) chip.

Secondary Memory:

- A. Hard Disk (Local Disk)
- B. Optical Disks: CD-R, CD-RW, DVD-R, DVD-RW
- C. Pen Drive
- D. Floppy Disks
- F. Memory Cards
- G. External Hard Disk
- H. Blu Ray Disk

Blu-Ray Disk:

Blu-ray (not Blue-ray) also known as Blu-ray Disc (BD), is the name of a new optical disc format. The format offers more than five times the storage capacity of traditional DVDs and can hold up to 25GB on a single-layer disc and 50GB on a dual-layer disc. While current optical disc technologies such as DVD, DVD±R, DVD±RW, and DVD-RAM rely on a red laser to read and write data, the new format uses a blue-violet laser instead, hence the name Blu-ray.

Units of Memory:

The smallest unit is bit, which mean either 0 or 1.

1 bit	= 0 or 1
1 Byte	= 8 bit
1 Nibble	= 4 bit
1 Kilo Byte	= 1024 Byte= 2^{10} Byte
1 Mega Byte	= 1024 KB= 2^{10} KB
1 Gega Byte	= 1024 MB= 2^{10} MB
1 Tera Byte	= 1024 GB= 2^{10} GB
1 Peta Byte	= 1024 TB= 2^{10} TB
1 Exa Byte	= 1024 PB= 2^{10} PB
1 Zetta Byte	= 1024 EB= 2^{10} EB
1 Yotta Byte	= 1024 ZB= 2^{10} ZB

Bootling

The process of loading the system files of the operating system from the disk into the computer memory to complete the circuitry requirement of the computer system is called bootling.

Types of Bootling:

There are two types of bootling:

- **Cold Bootling:** If the computer is in off state and we boot the computer by pressing the power switch 'ON' from the CPU box then it is called as cold bootling.
- **Warm Bootling:** If the computer is already 'ON' and we restart it by pressing the 'RESET' button from the CPU box or CTRL, ALT and DEL key simultaneously from the keyboard then it is called warm bootling.

Software

Software, simply are the computer programs. The instructions given to the computer in the form of a program is called Software. Software is the set of programs, which are used for different purposes. All the programs used in computer to perform specific task is called Software.

Types of software**1. System software:****a) Operating System Software**

DOS, Windows XP, Windows Vista, Unix/Linux, MAC/OS X etc. **b) Utility Software**

Windows Explorer (File/Folder Management), Compression Tool, Anti-Virus Utilities, Disk Defragmentation, Disk Clean, BackUp, WinZip, WinRAR etc...

c) Language Processors

Compiler, Interpreter and Assembler

2. Application software:**a) Package Software**

MS. Office, Adobe (Dreamweaver, Flash, PhotoShop)

b) Tailored or Custom Software

School Management system, Inventory Management System, Payroll system, financial system etc.

Operating system

Operating system is a platform between hardware and user which is responsible for the management and coordination of activities and the sharing of the resources of a computer. It hosts the several applications that run on a computer and handles the operations of computer hardware.

Functions of operating System:

- Processor Management
- Memory Management
- File Management
- Device Management

Types of Operating System:

- **Real-time Operating System:** It is a multitasking operating system that aims at executing real-time applications. Example of Use: e.g. control of nuclear power plants, oil refining, chemical processing and traffic control systems, air
- **Single User Systems:** Provides a platform for only one user at a time. They are popularly associated with Desk Top operating system which runs on standalone systems where no user accounts are required. Example: DOS.
- **Multi User Systems:** Provides regulated access for a number of users by maintaining a database of known users. Refers to computer systems that support two or more simultaneous users. Another term for multi-user is time sharing. Ex: All mainframes are multi-user systems. Example: Unix
- **Multi-tasking and Single-tasking Operating Systems:** When a single program is allowed to run at a time, the system is grouped under the single-tasking system category, while in case the operating system allows for execution of multiple tasks at a time, it is classified as

a multi-tasking operating system.

- **Distributed Operating System:** An operating system that manages a group of independent computers and makes them appear to be a single computer is known as a distributed operating system. Distributed computations are carried out on more than one machine. When computers in a group work in cooperation, they make a distributed system.

Commonly used operating systems

UNIX: A popular multi-user, multitasking operating system developed at Bell Labs in the early 1970s. UNIX was one of the first operating systems to be written in a high-level programming language, namely C. This meant that it could be installed on virtually any computer for which a C compiler existed.

LINUX: A freely-distributable open source operating system that runs on a number of hardware platforms. The Linux kernel was developed mainly by Linus Torvalds and it is based on Unix. Because it's free, and because it runs on many platforms, including PCs and Macintoshes, Linux has become an extremely popular alternative to proprietary operating systems.

Windows: Microsoft Windows is a series of graphical interface operating systems developed, marketed, and sold by Microsoft. Microsoft introduced an operating environment named Windows on November 20, 1985 as an add-on to MS-DOS in response to the growing interest in graphical user interfaces (GUIs). The most recent client version of Windows is Windows 10; the most recent server version is Windows Server 2016 R2; the most recent mobile version is Windows Phone 10.

SOLARIS: Solaris is a Unix operating system originally developed by Sun Microsystems. It superseded their earlier SunOS in 1993. **Oracle Solaris**, as it is now known, has been owned by Oracle Corporation since Oracle's acquisition of Sun in January 2010.

BOSS: BOSS (Bharat Operating System Solutions) GNU/Linux distribution developed by C-DAC (Centre for Development of Advanced Computing) derived from Debian for enhancing the use of Free/ Open Source Software throughout India. This release aims more at the security part and comes with an easy to use application to harden your Desktop.

Mobile OS: A mobile operating system, also called a mobile OS, is an operating system that is specifically designed to run on mobile devices such as mobile phones, smartphones, PDAs, tablet computers and other handheld devices. The mobile operating system is the software platform on top of which other programs, called application programs, can run on mobile devices.

Android: Android is a Linux-based mobile phone operating system developed by Google. Android is unique because Google is actively developing the platform but giving it away for free to hardware manufacturers and phone carriers who want to use Android on their devices.

Symbian: Symbian is a mobile operating system (OS) targeted at mobile phones that offers a high-level of integration with communication and personal information management (PIM) functionality. Symbian OS combines middleware with wireless communications through an integrated mailbox and the integration of Java and PIM functionality (agenda and contacts). The

Symbian OS is open for third-party development by independent software vendors, enterprise IT departments, network operators and Symbian OS licensees.

LANGUAGE PROCESSORS

Since a computer hardware is capable of understanding only machinelevel instructions, So it is necessary to convert the HLL into Machine Level Language. There are three Language processors:

- A. **Compiler:** It is translator which converts the HLL language into machine language in one go. A Source program in High Level Language gets converted into Object Program in Machine Level Language.
- B. **Interpreter:** It is a translator which converts the HLL language into machine language line byline. It takes one statement of HLL and converts it into machine code which is immediately executed. It eliminates the need of separate compilation/run. However, It is slow in processing as compare to compiler.
- C. **Assembler:** It translates the assembly language into machine code.

Microprocessor:

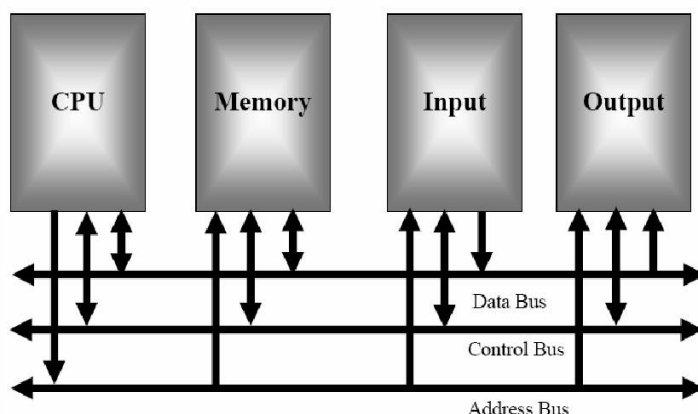
A microprocessor is a semiconductor chip, which is manufactured using the Large Scale integration (LSI) or Very Large Scale Integration (VLSI), which comprises Arithmetic Logic Unit, Control unit and Central Processing Unit (CPU) fabricated on a single chip.

Other Terminologies:

Registers: A register is a very small amount of very fast memory that is built into the CPU in order to speed up its operations by providing quick access to commonly used values. For example, if two numbers are to be multiplied, both numbers must be in registers, and the result is also placed in a register.

Bus:

A collection of wires through which data is transmitted from one part of a computer to another. This is a bus that connects all the internal computer components to the CPU and main memory. All buses consist of two parts -- an address bus and a data bus. The data bus transfers actual data whereas the address bus transfers information about where the data should go. The control bus is used by the CPU to direct and monitor the actions of the other functional areas of the computer. It is used to transmit a variety of individual signals (read, write, interrupt, acknowledge, and so forth) necessary to control and coordinate the operations of the computer.



Clock speed: Also called *clock rate*, the speed at which a microprocessor executes instructions. Every computer contains an internal clock that regulates the rate at which instructions are executed and synchronizes all the various computer components. The CPU requires a fixed number of clock ticks (or *clock cycles*) to execute each instruction. The faster the clock, the more instructions the CPU can execute per second.

Clock speeds are expressed in megahertz (MHz) or gigahertz (GHz).

16 bit Microprocessor: It indicates the width of the registers. A 16-bit microprocessor can process data and memory addresses that are represented by 16 bits. Eg. 8086 processor

32 bit Microprocessor: It indicates the width of the registers. A 32-bit microprocessor can process data and memory addresses that are represented by 32 bits. Eg. Intel 80386 processor, Intel 80486

64 bit Microprocessor: Eg. Pentium dual core, core 2 duo.
128 bit Microprocessor: It indicates the width of the registers. A 128-bit microprocessor can process data and memory addresses that are represented by 128 bits. Eg. Intel core i7

Difference between RISC & CISC architecture

RISC (*Reduced Instruction Set Computing*):

1. RISC system has reduced number of instructions.
2. Performs only basic functions.
3. All HLL support is done in software.
4. All operations are register to register.

CISC (*Complex Instruction Set Computing*):

1. A large and varied instruction set.
2. Performs basic as well as complex functions.
3. All HLL support is done in Hardware.
4. Memory to memory addressing mode

EPIC (Explicitly Parallel Instruction Computing):

It is a 64-bit microprocessor instruction set, jointly defined and designed by Hewlett Packard and Intel, that provides up to 128 general and floating point unit registers and uses *speculative loading*, *predication*, and *explicit parallelism* to accomplish its computing tasks

PORTS: A port is an interface between the motherboard and an external device. Different types of ports are available on motherboard as serial port, parallel port, PS/2 port, USB port, SCSI port etc.

Serial port (COM Port): A serial port transmit data one bit at a time. Typically on older PCs, a modem, mouse, or keyboard would be connected via serial ports. Serial cables are cheaper to make than parallel cables and easier to shield from interference. It is also called as communication port.

Parallel Port (LPT ports): It supports parallel communication i.e. it can send several bits simultaneously. It provides much higher data transfer speed in comparison with serial port. It is also called Line Printer Port.

USB (Universal Serial Bus): It is a newer type of serial connection that is much faster than the old serial ports. USB is also much smarter and more versatile since it allows the "daisy chaining" of up to 127 USB peripherals connected to one port. It provides plug & play communication.

PS/2 Port : PS/2 ports are special ports for connecting the keyboard and mouse to some PC systems. This type of port was invented by IBM.

FireWire Port : The IEEE 1394 interface, developed in late 1980s and early 1990s by Apple as FireWire, is a serial bus interface standard for high-speed communications and isochronous real-time data transfer. The 1394 interface is comparable with USB and often those two technologies are considered together, though USB has more market share.

Infrared Port: An IR port is a port which sends and receives infrared signals from other devices. It is a wireless type of port with a limited range of 5-10ft.

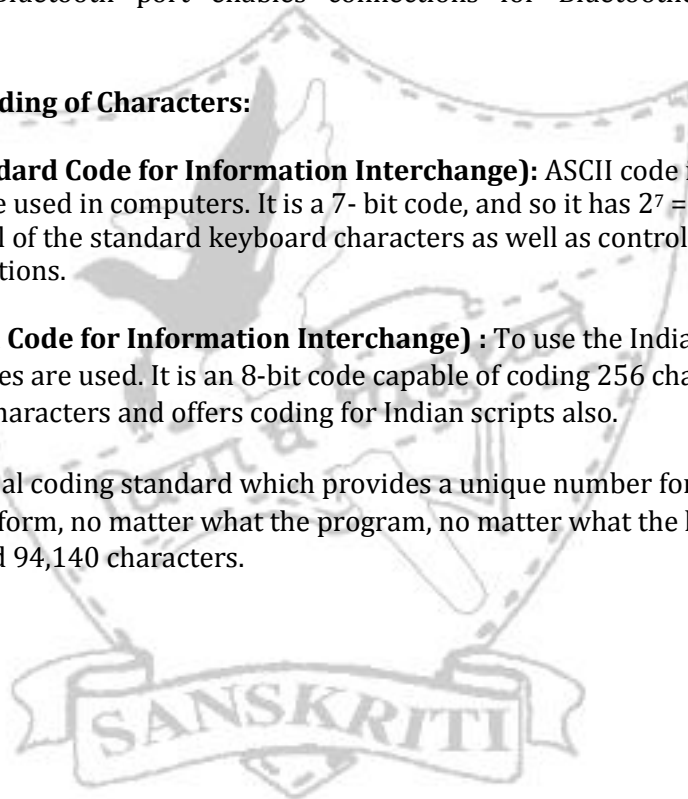
Bluetooth: Bluetooth uses short range radio frequencies to transmit information from fixed and mobile devices. These devices must be within the range of 32 feet, or 10 meters for Bluetooth to effectively work. A Bluetooth port enables connections for Bluetooth-enabled devices for synchronizing.

Internal Storage encoding of Characters:

ASCII (American Standard Code for Information Interchange): ASCII code is most widely used alphanumeric code used in computers. It is a 7-bit code, and so it has $2^7 = 128$ possible code groups. It represents all of the standard keyboard characters as well as control functions such as Return & Linefeed functions.

ISCII (Indian Standard Code for Information Interchange) : To use the Indian language on computers, ISCII codes are used. It is an 8-bit code capable of coding 256 characters. ISCII code retains all ASCII characters and offers coding for Indian scripts also.

Unicode: It is a universal coding standard which provides a unique number for every character, no matter what the platform, no matter what the program, no matter what the language. Unicode version 3.1 represented 94,140 characters.



Data Representation

Handout - NUMBER SYSTEMS

A. Decimal Number System:

Decimal Number system composed of 10 numerals or symbols. These numerals are 0 to 9. Using these symbols as digits we can express any quantity. It is also called base-10 system. It is a positional value system in which the value of a digit depends on its position.

These digits can represent any value, for example: **754**.

$$7 \cdot 10^2 + 5 \cdot 10^1 + 4 \cdot 10^0 = 700 + 50 + 4 = 754$$

base
digit position

The value is formed by the sum of each digit, multiplied by the **base** (in this case it is **10** because there are 10 digits in decimal

system) in power of digit position (counting from zero):

B. Binary Number System:

In Binary Number system there are only two digits i.e. 0 or 1. It is base-2 system. It can be used to represent any quantity that can be represented in decimal or other number system. It is a positional value system, where each binary digit has its own value or weight expressed as power of 2.

The following are some examples of binary numbers: 101101_2 11_2 10110_2

Conversion from Decimal to Binary or Binary to Decimal

Integer $(45)_{10} \rightarrow (X)_2$

Div	Quotient	Remainder	Binary Number (X)
45 / 2	22	1	1
22 / 2	11	0	01
11 / 2	5	1	101
5 / 2	2	1	1101
2 / 2	1	0	01101
1 / 2	0	1	101101

$(45)_{10} \rightarrow (101101)_2$

Fractional Part $(0.182)_{10} \rightarrow (X)_2$

Div	Product	Integer value	Binary Number (X)
0.182×2	0.364	0	0.0
0.364×2	0.728	0	0.00
0.728×2	1.456	1	0.001
0.456×2	0.912	0	0.0010
0.912×2	1.824	1	0.00101
0.824×2	1.648	1	0.001011
0.648×2	1.296	1	0.0010111

$(0.182)_{10} \rightarrow (0.0010111)_2$ (After we round and cut the number)

Conversion from Binary to Decimal $(X)_2 \rightarrow (X)_{10}$

$(101101.0010111)_2 \rightarrow (X)_{10}$

Multiply each digit

$$1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 0 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} + 0 \cdot 2^{-4} + 1 \cdot 2^{-5} + 1 \cdot 2^{-6} + 1 \cdot 2^{-7} = 32 + 0 + 8 + 4 + 0 + 1 + 0 + 0 + 0.125 + 0 + 0.03125 + 0.015625 + 0.007813 = (45.179688)_{10}$$

C. Octal Number System:

It has eight unique symbols i.e. 0 to 7. It has base of 8. Each octal digit has its own value or weight expressed as a power of 8.

Convert from decimal to octal $(X)_{10} \rightarrow (X)_8$

Integer Part $(45)_{10} \rightarrow (X)_8$

Div	Quotient	Remainder	Octal Number (X)
45 / 8	5	5	5
5 / 8	0	5	55

$(45)_{10} \rightarrow (55)_8$

Fractional Part $(0.182)_{10} \rightarrow (X)_8$

Mul	Product	Integer	Binary Number (X)	
$0.182 * 8$	1.456	1		0.1
$0.456 * 8$	3.648	3		0.13
$0.648 * 8$	5.184	5		0.135
$0.184 * 8$	1.472	1		0.1351
$0.472 * 8$	3.776	3		0.13513

$(0.182)_{10} \rightarrow (0.13513)_8$ (After we round and cut the number)

Convert from octal to decimal $(X)_8 \rightarrow (X)_{10}$

$(55.13513)_8 \rightarrow (X)_{10}$

We multiply each digit

$$5 * 8^1 + 5 * 8^0 + 1 * 8^{-1} + 3 * 8^{-2} + 5 * 8^{-3} + 1 * 8^{-4} + 3 * 8^{-5} + 6 * 8^{-6} =$$

$$40 + 5 + 0.125 + 0.03125 + 0.009766 + 0.000244 + 0.0001 + 0.0000229 = (45.1663829)_{10}$$

D. Hexadecimal Number System:

The hexadecimal system uses base 16. It has 16 possible digit symbols. It uses the digits 0 through 9 plus the letters A,B,C,D,E,F as 16 digit symbols. Each hexadecimal digit has its own value or weight expressed as a power of 16.

Table to remember

Decimal	Binary	Hexadecimal	Octal
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	8	10
9	1001	9	11
10	1010	A	12
11	1011	B	13
12	1100	C	14
13	1101	D	15
14	1110	E	16
15	1111	F	17
16	10000	10	20

Convert from decimal to hexadecimal $(X)_{10} \rightarrow (X)_{16}$ **Integer $(45)_{10} \rightarrow (X)_{16}$**

Div	Quotient	Remainder	Hex Number (X)
45 / 16	2	13	D (Since 13 decimal is D in hexadecimal)
2 / 16	0	2	2D

 $(45)_{10} \rightarrow (2D)_{16}$ **Fractional Number** **$(0.182)_{10} \rightarrow (X)_{16}$**

Mul	Product	Integer	Binary Number (X)
$0.182 * 16$	2.912		0.2
$0.912 * 16$	14.592	14	0.2E
$0.592 * 16$	9.472	9	0.2E9
$0.472 * 16$	7.552	7	0.2E97
$0.552 * 16$	8.832	8	0.2E978
$0.832 * 16$	13.312	13	0.2E978D

 $(0.182)_{10} \rightarrow (0.2E978D)_{16}$ (After we round and cut the number)**Convert from hexadecimal to decimal $(X)_{16} \rightarrow (X)_{10}$** $(2D.2E978D)_{16} \rightarrow (X)_{10}$ We multiply each digit

$$2 * 16^1 + 13 * 16^0 + 2 * 16^{-1} + 14 * 16^{-2} + 9 * 16^{-3} + 7 * 16^{-4} + 8 * 16^{-5} + 13 * 16^{-6} = 32 + 13 + 0.125 + 0.0546875 + 0.00219727 + 0.00010681 + 0.00000762 + 0.00000077 = (45.18199997)_{10}$$

Convert from binary to octal: For this conversion make the group of three digits from right to left before decimal & left to right after decimal then assign the specific octal value. (Given in table above)

 $(110101000.0101010)_2 \rightarrow (X)_8$

110 101 000 .0101010

6 5 0 .5 2

 $(110101000.0101010)_2 = (650.52)_8$

Convert from binary to hexadecimal: This conversion make the group of four digits from right to left before decimal & left to right after decimal then assign the specific Hexadecimal value. (Given in the table above)

 $(110101000.0101010)_2 \rightarrow (X)_{16}$

0001 1010 1000 .0101 0000

1 A 8 . A 8

 $(110101000.0101010)_2 \rightarrow (1A8.A8)_{16}$

Convert from hexadecimal to octal and binary: In this conversion write the binary of specific digit. For Octal three digit binary & for Hexadecimal four digit binary.

Convert from octal to binary $(650.52)_8 \rightarrow (X)_2$ 6 5 0 . 5 2 110101 000 . 101 010 $(650.52)_8 \rightarrow (110101000.101010)_2$	Convert from hexadecimal to binary $(1A8.A8)_{16} \rightarrow (X)_2$ 1 A 8 . A 8 0001 1010 1000 . 1010 1000
---	---

Assignment No. 1**Computer Organisation**

1. Which electronic device invention brought revolution in earlier computers?
2. Which memory is responsible for booting of system?
3. Where do you find analog computers in daily life?
4. What do you mean by term firmware?
5. What do you mean by language processors? Why we need it?
6. Give any example of hybrid computer in daily life.
7. Can we think of a computer system without operating system? Justify your answer.
8. Fifth generation of computer is a symbol of intelligence. Why?
9. Which is better for translator & why? Compiler or Interpreter. What do you mean by Defragmentation?
10. Write the full forms of RISC & CISC?
11. Which port a mouse should be connected?
12. What do you mean by LPT port?
13. What is difference between a USB & Firewire Port?
14. What is cache memory?
15. Answer the following:

(a) What is a Computer?

(b) Explain Input and Output Devices with examples.

(c) State whether the following are input/ output device. Also state the purpose of each device :

i. Key board	ii. Speaker
iii. Mouse	iv. Camera
v. Printer	vi. Scanner
vii. Plotter	viii. Joy Stick
ix. Monitor	x. MIC

(d) Fill in the following Memory Units relation Ships:

- i. 1 Byte = _____ Bits = 2^{\dots} Bits
- ii. 1 KiloByte = _____ Bytes = 2^{\dots} Bytes
- iii. 1 MegaByte = _____ KiloBytes = 2^{\dots} KiloBytes
- iv. 1 GigaByte = _____ MegaBytes = 2^{\dots} MegaBytes
- v. 1 TeraByte = _____ GigaBytes = 2^{\dots} GigaBytes
- vi. 1 TeraByte = _____ MegaBytes = 2^{\dots} MegaBytes
- vii. 1 MegaByte = _____ Bits = 2^{\dots} Bits

16. Answer the following:

(a) What do you understand by term Primary Memory?

(b) Differentiate between RAM and ROM.

(c) Define Secondary Memory with an example.

(d) What are Secondary Storage Devices? Give five examples of same.

(e) Explain the purpose of a UPS.

(f) What are drivers? Explain with the help of examples.

(g) Write brief note on the different types of Printers and their specific features that differentiate them from each other.

17. Answer the following:

(a) Explain the purpose of a CPU. Describe its various parts.

(b) Describe the booting up process of a computer in detail.

(c) Both RAM and ROM- BIOS are primary Memory. True/ False?

(d) Define the following (with examples):

- i) System Software
- ii) Application Software
- iii) Utility Software

18. Answer the following:

(a) What is an Operating System? Give examples.

(b) State the Key functions of an Operating System?

(c) What do you understand by the term GUI? How is it different from a non- GUI operating System?

(d) Differentiate between Microsoft Windows and DOS.

(e) Differentiate between **LINUX and UNIX**.

Assignment No. 2

Data Representation

1. Perform the following conversions :

- a) $(123.36)_{10} = ()_2$
- b) $(1001001.1001)_2 = ()_{10}$
- c) $(456.42)_8 = ()_{10}$
- d) $(945.64)_{10} = ()_8$
- e) $(532.67)_8 = ()_2$
- f) $(10010101.110011)_2 = ()_8$
- g) $(AF10.2E9)_{16} = ()_{10}$
- h) $(1001.10)_{10} = ()_{16}$
- i) $(EA52.B71)_{16} = ()_2$
- j) $(101101011.110011)_2 = ()_{16}$

Note: Show the conversion procedure step by step

2. Represent the following decimal numbers using sign and magnitude form : 17 and -17.

3. Represent the following decimal numbers using One's complement form : 22 and -22.

4. Represent the following decimal numbers using Two's complement form : 122 and -122.

5. Convert the following:

- i. 101001.0101 to decimal
- ii. $(236)_8$ to Binary
- iii. $(266)_{10}$ to Hexadecimal
- iv. $(AF2)_{16}$ to Binary
- v. 0101110.1010110 to Hexadecimal

UNIT-2 Introduction to C++

C++ CHARACTER SET:

Character set is a set of valid characters that a language can recognize. A character can represent any letter, digit, or any other sign. Following are some of the C++ character set.

Letters	A to z and a to z
Digits	0 -9
Special symbols	+ - * ^ \ [] { } = ! < > . ' ' ; : & #
White space	Blank space, horizontal tab(- >), carriage return , newline, form feed.
Other characters	256 Ascii characters as data or as literals.

TOKENS:

The smallest lexical unit in a program is known as token. A token can be any keyword, Identifier, Literals, Punctators, Operators.

KEYWORDS :

These are the reserved words used by the compiler. Following are some of the Keywords.

auto	continue	float	New	signed	volatile
short long	class	struct	Else	inline	goto
delete	friend	private	typedef	void	template
catch	register	sizeof	Union		

IDENTIFIERS:

An arbitrary name consisting of letters and digits to identify a particular word. C++ is a case sensitive language as it treats upper and lower case letters differently. The first character must be a letter. The underscore counts as a letter.

Pen time580 s2e2r3 _dos _HJ13_JK

LITERALS:

The data items which never change their value throughout the program run. There are several kind of literals:

- ☑ Integer constant
- ☑ Character constant
- ☑ Floating constant
- ☑ String constant.

Integer constant :

Integer constant are whole numbers without any fractional part. An integer constant must have at least one digit and must not contain any decimal point. It may contain either + or -. A number with no sign is assumed as positive.

e.g 15, 1300, -58795.

Character Constant:

A character constant is single character which is enclosed within single quotation marks. e.g 'A'.

Floating constant:

Numbers which are having the fractional part are referred as floating numbers or real constants. It may be a positive or negative number. A number with no sign is assumed to be a positive number.
e.g 2.0, 17.5, -0.00256

String Literals:

It is a sequence of letters surrounded by double quotes. E.g "abc".

PUNCTUATORS:

The following characters are used as punctuators which are also known as separators in C++:

Punctuator	Name	Function
[]	Brackets	These indicate single and multidimensional array subscripts
()	Parenthesis	These indicate function calls and function parameters.
{ }	Braces	Indicate the start and end of compound statements.
;	Semicolon	This is a statement terminator.
:	Colon	It indicates a labeled statement
*	Asterisk	It is used as a pointer declaration
...	Ellipsis	These are used in the formal argument lists of function prototype to indicate a variable number of arguments.
=	Equal to	It is used as an assigning operator.
#	Pound sign	This is used as preprocessor directives.

OPERATORS:

These are those lexical units that trigger some computation when applied to variables and other objects in an expression. Following are some operators used in C++

Unary operators: Those which require only one operand to trigger. e.g. &, +, ++, --, !.

Binary operators: These require two operands to operate upon. Following are some of the Binary operators.

Arithmetic operators :

+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Remainder

Logical Operators :

&&(logical AND)

|| (Logical OR)

Relational Operator:

< less than	> Greater than
<= Less than equal to.	>= greater than equal to.
==equal to	!= not equal to.

Assignment Operator:

=	Assignment Operator
+=	Assign sum
-=	Assign difference
*=	Assign Product
/=	Assign quotient
%=	Assign Remainder

Conditional operator (?)

The conditional operator evaluates an expression returning a value if that expression is true and a different one if the expression is evaluated as false.

Its format is:

condition ? result1 : result2

e.g `7==5 ? 4 : 3` // returns 3, since 7 is not equal to 5.

Comma operator (,)

The comma operator (,) is used to separate two or more expressions that are included where only one expression is expected. When the set of expressions has to be evaluated for a value, only the rightmost expression is considered.

For example, the following code:

```
a = (b = 3, b + 2);
```

Would first assign the value 3 to b, and then assign b+2 to variable a. So, at the end, variable a would contain the value 5 while variable b would contain value 3.

Explicit type casting operator

Type casting operators allow you to convert a datum of a given type to another. There are several ways to do this in C++. The simplest one, which has been inherited from the C language, is to precede the expression to be converted by the new type enclosed between parentheses () :

```
int i;
```

```
float f = 3.14; i = (int) f;
```

The previous code converts the float number 3.14 to an integer value (3), the remainder is lost. Here, the typecasting operator was (int). Another way to do the same thing in C++ is using the functional notation: preceding the expression to be converted by the type and enclosing the expression between parentheses:

```
i = int (f);
```

Both ways of type casting are valid in C++.

sizeof()

This operator accepts one parameter, which can be either a type or a variable itself and returns the size in bytes of that type or object:

```
a = sizeof (char);
```

This will assign the value 1 to a because char is a one-byte long type.

The value returned by sizeof is a constant, so it is always determined before program execution.

Input Output (I/O) In C++**The cout Object:**

The cout object sends to the standard output device. cout sends all output to the screen i.e monitor.

The syntax of **cout** is as follows: `cout << data;`

e.g

```
cout << a;           ( here a can be any variable)
```

The cin operator :

The cin operator is used to get input from the keyboard. When a program reaches the line with cin, the user at the keyboard can enter values directly into variables.

The syntax of **cin** is as follows: `cin>>variablename;`

e.g

`cin>>ch; (here ch can be any variable)`

Basic structure of a C++ program:

Following is the structure of a C++ program that prints a string on the screen:

```
#include<iostream.h>
void main ()
{
cout<<" Computer Science for Class XI";
}
```

The program produces following output:

Computer Science for Class XI

The above program includes the basic elements that every C++ program has. Let us check it line by line

#include<iostream.h> : This line includes the preprocessor directive include which includes the header file `iostream.h` in the program.

void main () :this line is the compilation starts with the **main ()**. Every C++ programs compilation starts with the **main ()**.

void Every C++ programs is the keyword used when the function has no return values.

{ : this is the start of the compound block of `main ()`.

cout<<" Computer Science for Class XI"; : this statement prints the sequence of string " **Computer Science for Class XI**" into this output stream i.e. on monitor.

Every statement in the block will be terminated by a semicolon (;) which specifies compiler the end of statement.

COMMENTS in a C++ program.:

Comments are the line that compiler ignores to compile or execute. There are two types of comments in C++.

1. **Single line comment:** This type of comment deactivates only that line where comment is applied. Single line comments are applied with the help of "`//`".

e.g `// cout<<tomorrow is holiday` the above line is starting with `//` so compiler wont access this line.

2. **Multi line Comment:** This type of comment deactivates group of lines when applied. This type of comments are applied with the help of the operators "`/*`" and "`*/`". These comment mark with `/*` and end up with `*/`. This means all statements that fall between `/*` and `*/` is considered even though it is spread across many lines.

e.g `#include<iostream.h>`
`int main ()`
`{cout<< " hello world";`
`/* this is the program to print`
`hello world.For`

demonstration of comments
*/}

In the above program the statements between /* and */ will be ignored by the compiler.

CASCADING OF OPERATOR:

When shift operators (<< and >>) are used more than one time in a single statement then it is called as cascading of operators. **e.g. cout<< roll<< age<<endl;**

DATATYPES IN C++:

A datatype is just an interpretation applied to string of bytes .

Data in C++ are of two types:

1. Simple /Fundamental datatypes .
2. Structures/Derived datatypes.

Simple /Fundamental data types:

When programming, we store the variables in our computer's memory, but the computer has to know what kind of data we want to store in them, since it is not going to occupy the same amount of memory to store a simple number than to store a single letter or a large number, and they are not going to be interpreted the same way.

The memory in our computers is organized in bytes. A byte is the minimum amount of memory that we can manage in C++. A byte can store a relatively small amount of data: one single character or a small integer (generally an integer between 0 and 255). In addition, the computer can manipulate more complex data types that come from grouping several bytes, such as long numbers or non-integer numbers.

Next you have a summary of the basic fundamental data types in C++, as well as the range of values that can be represented with each one:

Name	Description	Size	Range	
char	Character	1byte	signed: -128 to 127	unsigned: 0 to 255
int	Integer	2bytes	signed: -32768 to 32767	unsigned: 0 to 65535
Float	Floating point number	4 bytes	3.4×10^{-38} to $3.4 \times 10^{38}-1$	
Double	Double precision floating point number	8 bytes	1.7×10^{-308} to $1.7 \times 10^{308}-1$	

Derived Data Types:

The datatypes which are extracted / derived from fundamental data types are called derived datatypes. These datatypes can be derived by using the declaration operator or punctuators for e.g Arrays, function, Pointer, Class , Structure, union etc.

Class :A class represents a group of similar objects. To represent class in C++ it offers a userdefined datatypes called CLASS .Once a Class has been defined in C++, Object belonging to that class can easily be created. A Class bears the same relationship to an object that a type does to a variable.

STRUCTURE:

A Structure is a collection of variables of same/different data types referenced under one name. The access to structure variables is by default global i.e. they can be accessed publicly throughout the program.

UNION :

A memory location shared between two different variables of different datatypes at different times is known as Union. Defining union is similar as defining the structure.

References:

A reference is an alternative name for an object. A reference variable provides an alias for a previously defined variable. A reference declaration consists of base type, an & (ampersand), a reference variable name equated to a variable name .the general syntax form of declaring a reference variable is as follows.

Datatype &ref_variable = variable_name;

Where Datatype is any valid C++ datatype, ref_variable is the name of reference variable that will point to variable denoted by variable_name.

e.g int a= 10; int&b= a;

then the values of **a is 10** and the value of **b is also 10;**

Constant :

The keyword const can be added to the declaration of an object to make that object constant rather than a variable. Thus the value named constant cannot be altered during the program run.

Syntax:-

const type name=value;

Example:-

const int uage=50; // it declares a constant named as **uage** of type integer that holds value 50.

Preprocessor Directives:

#include is the preprocessor directive used in C++ programs. This statement tells the compiler to include the specified file into the program. This line is compiled by the processor before the compilation of the program.

e.g #include<iostream.h>

the above line includes the header file **iostream** into the program for the smooth running of the program.

Compilation and Linking

Compilation refers to the processing of source code files (.c, .cc, or .cpp) and the creation of an 'object' file. This step doesn't create anything the user can actually run. Instead, the compiler merely produces the machine language instructions that correspond to the source code file that was compiled. For instance, if you compile (but don't link) three separate files, you will have three object files created as output, each with the name <filename>.o or <filename>.obj (the extension will depend on your compiler). Each of these files contains a translation of your source code file into a machine language file -- but you can't run them yet!

You need to turn them into executables your operating system can use. That's where the linker comes in.

Linking refers to the creation of a single executable file from multiple object files. In this step, it is common that the linker will complain about undefined functions (commonly, `main` itself). During compilation, if the compiler could not find the definition for a particular function, it would just assume that the function was defined in another file. If this isn't the case, there's no way the compiler would know -- it doesn't look at the contents of more than one file at a time.

The linker, on the other hand, may look at multiple files and try to find references for the functions that weren't mentioned.

ERRORS:

There are many types of error that are encountered during the program run. following are some of them:

1. **Compiler error:** The errors encountered during the compilation process are called Compiler error. Compiler error are of two types:
 - Syntax error.
 - Semantic error.

Syntax Error: Syntax error is the one which appears when we commit any grammatical mistakes. These are the common error and can be easily corrected.

These are produced when we translate the source code from high level language to machine language.

e.g. `cout<<endl;` This line will produce a syntax error as there is a grammatical mistake in the word `cout`

Semantic error: These errors appear when the statement written has no meaning.

e.g. `a + b = c;` ; this will result a semantically error as an expression should come on the right hand side of an assignment statement.

2. **Linker Errors:** Errors appear during linking process e.g if the word `main` written as `mian` . The program will compile correctly but when link it the linking window will display errors instead of success.
3. **Run Time error:** An abnormal program termination during execution is known as Runtime Error. e.g. If we are writing a statement $X = (A + B) / C$; the above statement is grammatically correct and also produces correct result. But what happen if we gave value 0 to the variable `c`, this statement will attempt a division by 0 which will result in illegal program termination. Error will not be found until the program will be executed because of that it is termed as run time error.
4. **Logical Error.:** A logical error is simply an incorrect translation of either the problem statement or the algorithm.
e.g : $\text{root1} = -b + \sqrt{b^2 - 4ac} / (2a)$
the above statement is syntactically correct but will not produce the correct answer because the division have a higher priority than the addition, so in the above statement division is performed first, then addition is performed but in actual practice to do addition performed then divide the resultant value by $(2a)$.

Manipulators :

Manipulators are the operators used with the insertion operator `<<` to modify or manipulate the way data is displayed. There are two types of manipulators **`endl`** and **`setw`**.

1. **The endl manipulator** :The endl manipulator outputs new line . It takes the compiler to end the line of display.

```
cout<< " Sanskriti School"<<endl;  
cout<< " Payroll and Employee Department";
```

The output of the above code will be

```
Sanskriti School  
Payroll and Employee Department
```

2. **The setw Manipulator** : The **setw** manipulator causes the number (or string) that follows it in the stream to be printed within a field **n** characters wide where n is the arguments to **setw (n)**.

Increment and Decrement Operators in C++:

The increase operator (++) and the decrease operator (--) increase or reduce by one the value stored in a variable. They are equivalent to +=1 and to -=1, respectively. Thus:

```
C++;  
C +=1;  
C=C+1;
```

are all equivalent in its functionality: the three of them increase by one the value of C.

A characteristic of this operator is that it can be used both as a prefix and as a suffix. That means that it can be written either before the variable identifier (++a) or after it (a++). Although in simple expressions like a++ or ++a both have exactly the same meaning, in other expressions in which the result of the increase or decrease operation is evaluated as a value in an outer expression they may have an important difference in their meaning:

In the case that the increase operator is used as a prefix (++a) the value is increased **before** the result of the expression is evaluated and therefore the increased value is considered in the outer expression;

Example 1

```
B=3;  
A =++B;           // (here A contains 4, B contains 4).
```

In case that it is used as a suffix (a++) the value stored in **a** is increased **after** being evaluated and therefore the value stored before the increase operation is evaluated in the outer expression.

Example 2

```
B=3;  
A=B++;           // (here a contains 3, B contains 4).
```

In Example 1, B is increased before its value is copied to A. While in Example 2, the value of B is copied to A and then B is increased.

Assignment No : 3
Getting Started With C++

- Q1. What are C++ Tokens? Name them.
- Q2. What is the difference between Key Words and Identifiers. Give Examples.
- Q3. Define Variables. Give Example.
- Q4. List all the Arithmetic Operators.
- Q5. List all the Logical Operators.
- Q6. List all the Relational Operators.
- Q7. What is the difference between :
- i) = and ==
 - ii) '1' and 1
 - iii) 'A' and "A"
 - iv) << and >>
- Q8. What is the importance of main() in C++?
- Q9. Why do we need to include header files in a C ++ program?
- Q10. What is the importance of #include statement in C++?
- Q11. What do the angular <> brackets signify in include statement?
- Q12. What is Code Generation? Explain all the steps in details.
- Q13. What happen during Compiling?
- Q14. What happen during Linking?
- Q15. What is the use of the following Keys/ Key combination in C++?
- i) F2
 - ii) Alt+ F9
 - iii) Ctrl+F9
 - iv) Alt + F5
 - v) Alt + F3
 - vi) Alt +X
- Q16. Differentiate between Syntax errors and Symantic Errors.
- Q17. Differentiate between Logical errors and Run Time Errors.
- Q18. Calculate the length of the following :
- i) 'Computer'
 - ii) "Sanskriti\n\tSchool"
- Q19. Mention in each case, whether it is a valid identifier or not. State the reason in case of invalidity.
- i) Math marks
 - ii) x1y2z3
 - iii) int
 - iv) Score

- Q20. Identify the errors in the following C++ statements :
- (i) Int a= 10;
 - (ii) char s = "String";
 - (iii) b + c = a + n;
- Q21. Rewrite the following code after correcting all the syntax errors. Underline your corrections:
- ```
/* WAP to subtract two numbers
void main()
{
 int x=20; y = 35;
 z = y - x;
 cout>>"Difference = ">>z;
}
```

#### **Assignment No : 4**

#### **Data Handling in C++**

- Q1. Differentiate between Primitive and Non Primitive data types.
- Q2. Name four basic data types in C++. Also state their memory storage size.
- Q3. What are data type modifiers? Name them.
- Q4. What are character data types often termed as integer data types?
- Q5. What are C++ constants? Name them, also give one example of each.
- Q6. What would be the output of the following C++ code:
- ```
#include<iostream.h>
void main()
{
    char ch ='A';
    int x = ch;
    cout<<x<<" "<<ch<<" "<<ch+2;
}
```
- Q7. Rewrite the following code after correcting all the syntax errors. Underline your corrections:
- ```
#include<iostream.h>
void main()
{
 int x = 'A';
 char a = 65;
 float f =10.5;
 int y = x+a;
 float p = f+a;
 cout<<y<<" "<<p;
}
```

**Assignment No : 5**  
**Operators and Expressions**

- Q1. Differentiate between Unary and binary operators.
- Q2. Which is the ternary Operator of C++? Explain with the help of an example.
- Q3. What is Type Casting? Explain with the help of an example.
- Q4. What is Implicit Type conversion? Explain with the help of an example.
- Q5. What will be the output of the following C++ expressions :
- i) `cout<< "n++ = "<< n++ - --n;` ( if n=16 initially)
  - ii) `char grade = ++a >=90 ? 'A' : 'B';` ( if a=100 initially)  
`cout<<grade;`
  - iii) `cout<<x==10;` ( if x=5 initially)
  - iv) `cout<<k++ <<"\t"<<++k<<endl;` ( if k=4 initially)
  - v) `int a=6, b=3;`  
`int y= b++ * a/b * b/a+5;`  
`cout<<y;`
  - vi) `int a=16, b = 4;`  
`int k = b/a;`  
`float h = b/a;`  
`cout<<k<< ","<<h;`
- Q6. Rewrite the following code after correcting all the syntax errors. Underline your corrections:
- i) `include<iostream.h>`  
`void main`  
`{ constint x;`  
`cout<< "Enter a number : ";`  
`cin>> x;`  
`cout<< "The square of <<x<<is = <<x*x;`  
`}`
  - ii) `# include<iostreem.h>`  
`void main()`  
`{ clrscr();`  
`int x=2, y=0;`  
`(x=2)? 5* 5 : 6*6`  
`}`
  - iii) `Include<iostream.h>`  
`void main ()`  
`{`  
`float a=5.5;`  
`charch=a;`  
`int x==2;`  
`a = x * ch;`  
`x = ch + 32;`  
`int k = x%a;`  
`getch(); }`

- Q7. What would be the result of following relational expressions? Show all the steps of execution :
- i)  $!(p) \&\& q$
  - ii)  $(p > q) || (p + q < 25/p)$
  - iii)  $(p > !q) \&\& (!p)$
- When
- a)  $p=1$  and  $q=0$
  - b)  $p=2$  and  $q=-5$

### Programming Methodology

Writing good program is a skill. This can be developed by using the following guidelines.

1. Meaningful Names for identifiers.
2. Ensure clarity of expression
3. Use of comments and indentations
4. Insert blank lines and blank spaces
5. Statements on separate lines

### **Characteristics of a Good Program**

1. Effective and efficient
2. User friendly
3. Self-documenting code
4. Reliable
5. Portable

### **Problem solving methodology and techniques**

1. Understand the problem well
2. Analyze the program
3. Code program
4. Test and Debug program
5. Implementation & Documentation

**Algorithm :-** Algorithm is a step-by-step process of solving a well-defined computational problem. An algorithm is an effective method expressed as a finite list of well-defined instructions for calculating a function, starting from an initial state and initial input. Thus, a step-by-step procedure to solve the problem is called algorithm.

Example 1: Let us take one simple day-to-day example by writing algorithm for making „Maggi Noodles“ as a food.

Step 1: Start

Step 2: Take pan with water

Step 3: Put pan on the burner

Step 4: Switch on the gas/burner

Step 5: Put maggi and masala

Step 6: Give two minutes to boil

Step 7: Take off the pan

Step 8: Take out the maggi with the help of fork/spoon

Step 9: Put the maggi on the plate and serve it

Step 10: Stop. Further, the way of execution of the program shall be categorized into three ways: (i) sequence statements; (ii) selection statements; and (iii) iteration or looping statements. This is also called as “control structure”.

**Selective Statements:** In this program, some portion of the program is executed based upon the conditional test. If the conditional test is true, compiler will execute some part of the program, otherwise it will execute the other part of the program

Example2 : Write an algorithm to check whether a person is eligible to vote? (Age more than or equal to 18 years makes one eligible to vote).

Step 1: Start

Step 2: Take age and store it in age

Step 3: Check age value, if age  $\geq 18$  then go to step 4 else step 5

Step 4: Print "Eligible to vote" and go to step 6

Step 5: Print "Not eligible to vote"

Step 6: Stop

**Iterative statements:** In some programs, certain set of statements are executed again and again based upon conditional test. i.e. executed more than one time. This type of execution is called "looping or iteration".

Example 3: Write an algorithm to print all natural numbers up to "n".

Step 1: Start

Step 2: Take any number and store it in n.

Step 3: Store 1 in I

Step 4: Check I value, if  $I \leq n$  then go to step 5 else go to step 8







Step 5: Print I

Step 6: Increment I value by 1

Step 7: Go to step 4

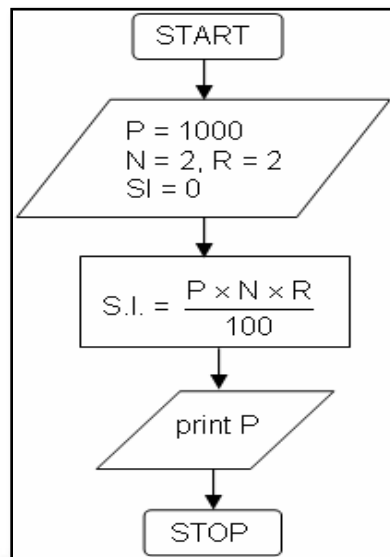
Step 8: Stop

**Flowchart:-** We can also show steps of an algorithm in graphical form by using some symbols. This is called flowcharting. Thus, Flowchart is a pictorial view of algorithm. Flowchart Symbols:-Some of the standard symbols along with respective function(s) that are used for making flowchart are as follows:

| Symbol                                                                              | Name                             | Description                                                                           |
|-------------------------------------------------------------------------------------|----------------------------------|---------------------------------------------------------------------------------------|
|  | Rectangular or action            | A process or action such as calculation, input/output, assignment etc.                |
|  | Oval                             | Represents a complete algorithm Begin/Start, End/Stop.                                |
|  | Diamond or decision              | Which indicates that a decision to be made such as choice between YES or NO.          |
|  | Flowlines                        | Indicates the order in which the actions are to be performed.                         |
|  | Small circle or connector symbol | When describing a portion of a complete algorithm continued from or will continue on. |
|  | Input or output                  | The data or input/output.                                                             |

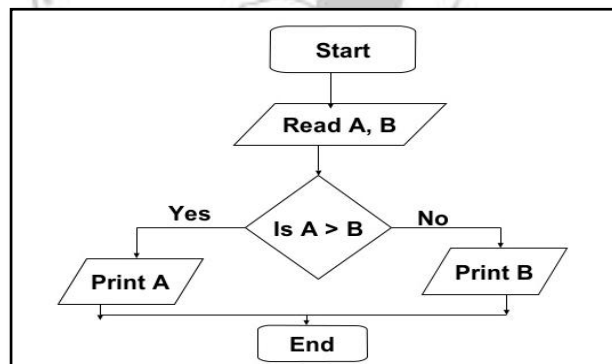
**Example :** Draw a flowchart to find the simple interest. (Sequence)

Solution:



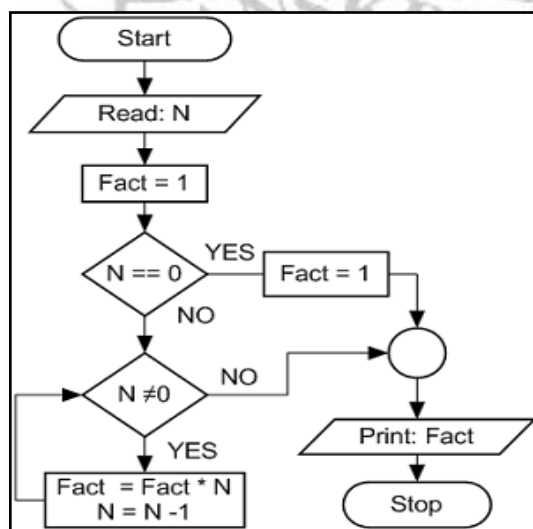
**Example :** Draw a flowchart to find bigger number among two numbers (selective)

Solution:



**Example :** Draw a flow chart to find factorial of any number.

Solution:





## UNIT-3

### Flow of Control

#### Linear Flow

A simple linear program is one which has a linear flow of execution of programming logic/statements. The logic of program flows (or better they are termed as control flow) from top to bottom out of set of statements.

#### If-Else statement

An **if** statement can be followed by an optional **else** statement, which executes when the boolean expression is false.

##### Syntax:

The syntax of an if...else statement in C++ is:

```
if(boolean_expression)
{
 // statement(s) will execute if the boolean expression is true
}
else
{
 // statement(s) will execute if the boolean expression is false
}
```

If the boolean expression evaluates to **true**, then the **if block** of code will be executed, otherwise **else block** of code will be executed.

#### Switch..case construct

Consider a situation in which, only one block of code needs to be executed among many blocks. This type of situation can be handled using nested if...else statement but, the better way of handling this type of problem is using switch...case statement.

##### Syntax of switch

```
switch (n)
{
case constant1:
 code/s to be executed if n equals to constant1; break;
case constant2:
 code/s to be executed if n equals to constant2; break;
.
.
.
default:
 code/s to be executed if n doesn't match any of the cases;
}
```

The value of **n** is either an integer or a character in above syntax. If the value of **n** matches constant in case, the relevant codes are executed and control moves out of the switch statement. If the **n** doesn't matches any of the constant in case, then the default statement is executed and control moves out of switch statement.

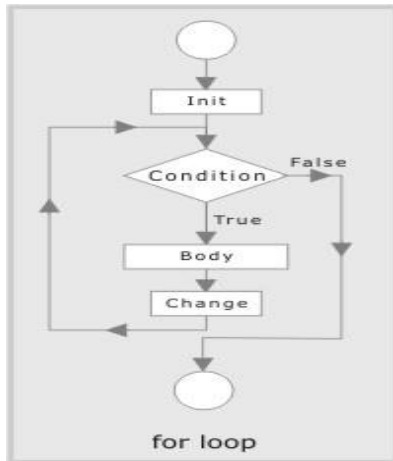
### For Loop

A **for** loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

#### **Syntax:**

The syntax of a for loop in C++ is:

```
for (init; condition; increment)
{
 statement(s);
}
```



### While Loop

A **while** loop statement repeatedly executes a target statement as long as a given condition is true.

#### **Syntax:**

The syntax of a while loop in C++ is:

```
while(condition)
{ statement(s);
}
```

Here, **statement(s)** may be a single statement or a block of statements. The **condition** may be any expression, and true is any non-zero value. The loop iterates while the condition is true.

When the condition becomes false, program control passes to the line immediately following the loop.

### Do-While Loop

Unlike **for** and **while** loops, which test the loop condition at the top of the loop, the **do...while** loop checks its condition at the bottom of the loop.

A **do...while** loop is similar to a while loop, except that a do...while loop is guaranteed to execute at least one time.

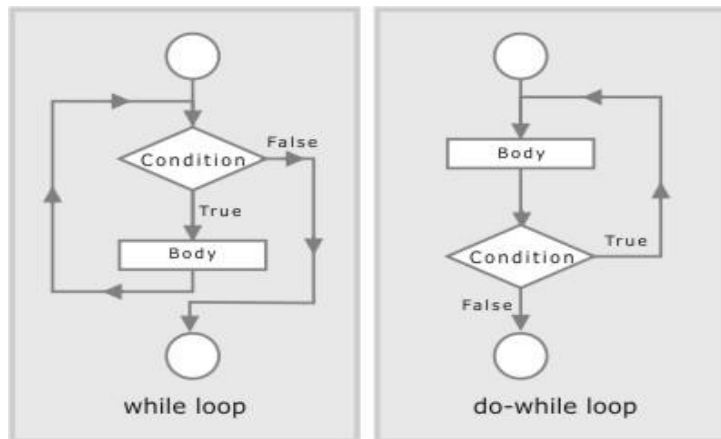
#### **Syntax:**

The syntax of a do...while loop in C++ is:

```
do
{ statement(s);
}while(condition);
```

Notice that the conditional expression appears at the end of the loop, so the statement(s) in the loop execute once before the condition is tested.

If the condition is true, the flow of control jumps back up to do, and the statement(s) in the loop execute again. This process repeats until the given condition becomes false.



**Jump Statements :** The jump statements unconditionally transfer program control within a function.

- goto statement
- break statement
- continue statement

**The goto statement** - goto allows to make jump to another point in the program.

gotopqr;

**pqr:** pqr is known as label. It is a user defined identifier. After the execution of goto statement, the control transfers to the line after label pqr.

**The break statement** - The break statement, when executed in a switch structure, provides an immediate exit from the switch structure. Similarly, you can use the break statement in any of the loop. When the break statement executes in a loop, it immediately exits from the loop.

**The continue statement** - The continue statement is used in loops and causes a program to skip the rest of the body of the loop.

```
while (condition)
```

```
{
```

```
Statement 1;
```

```
 If (condition)
```

```
 continue;
```

```
 statement; }
```

The continue statement skips rest of the loop body and starts a new iteration.

**The exit ( ) function** - The execution of a program can be stopped at any point with exit ( ) and a status code can be informed to the calling program. The general format is  
exit (code) ;

where code is an integer value. The code has a value 0 for correct execution. The value of the code varies depending upon the operating system

### Assignment No : 6

#### Flow of Control

- Q1. Explain the working of **for** loop with the help of an example.  
Q2. Differentiate between loops and selection statements.  
Q3. Differentiate between :  
i) while and do.... while loops  
ii) break and continue  
iii) if... else and switch...case

Give explanatory example for each.

- Q4. State the outputs of the following program segments :

```
i) int k=10;
for(; k>=1; k- -)
{
 k++;
}
cout<<k;
```

```
ii) int a=b=4;
while (a==b)
{
 if (a%2==0) ++a;
 if (b/2==2) ++b;
}
cout<<a<< " :: " <<b;
```

```
iii) int n = 12345, s=0, c=0;
while (n!=0)
{
 x=n%10;
 s+=x*pow(10,c);
 n=n/10;
}
cout<<s;
```

```
iv) for(int i=10 ; i<=50 ; i++);
cout<<i;
```

```

v) charch = 'a';
int x=0;
while(ch!= 'e' && x<10)
{
 cout<<ch+x;
 x++;
 ch++;
}

```

Q5. Explain what would be the difference in the outputs of the following C++ code segments:

i)

|                                                                                                                            |                                                                                                                                |                                                                                                                                             |
|----------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------|
| <pre> int x =120; for( ; x&lt;=130;x++ )     if (x%2) a++;     if (x%2==1 &amp;&amp; a%2==0)         cout&lt;&lt;x; </pre> | <pre> int x =120; for( ; x&lt;=130;x++ ) {     if (x%2) a++;     if (x%2==1 &amp;&amp; a%2==0)         cout&lt;&lt;x; } </pre> | <pre> int x =120; for( ; x&lt;=130;x++ )     if (x%2) a++;     else         if (x%2==1 &amp;&amp; a%2==0)             cout&lt;&lt;x; </pre> |
|----------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------|

ii)

|                                                                                                         |                                                                                                             |
|---------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------|
| <pre> int n =3465, S=0; while(n%2==0&amp;&amp; n&gt;0) {     S+=n%2;     n=n/2; } cout&lt;&lt;S; </pre> | <pre> int n =3465, S=0; do {     S+=n%2;     n=n/2; } while(n%2==0&amp;&amp; n&gt;0); cout&lt;&lt;S; </pre> |
|---------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------|

iii)

|                                                                                                                                                                                                                                                                                                              |                                                                                                                                                                                                                                                                                                                                                         |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre> int a=8, b=10, ans; cout&lt;&lt; "1. Calculate area\n"; cout&lt;&lt; "2. Calculate perimeter \n"; cout&lt;&lt; "3. Exit program\n"; cin&gt;&gt;ans; switch(ans) {     case 1 : cout&lt;&lt; a*b;     case 2 : cout&lt;&lt; 2*(a+b);     case 3 : exit(0);     default : cout&lt;&lt; "error"; } </pre> | <pre> int a =8, b=10, ans; cout&lt;&lt; "1. Calculate area\n"; cout&lt;&lt; "2. Calculate perimeter \n"; cout&lt;&lt; "3. Exit program\n"; cin&gt;&gt;ans; switch(ans) {     case 1:  cout&lt;&lt; a*b;               break;     case 2:  cout&lt;&lt; 2*(a+b);               break;     case 3:  exit(0);     default : cout&lt;&lt; "error"; } </pre> |
| <p>When the value for <u>ans</u> is inputted as: 1<br/> When the value for <u>ans</u> is inputted as: 2<br/> When the value for <u>ans</u> is inputted as: 3<br/> When the value for <u>ans</u> is inputted as: 4</p>                                                                                        |                                                                                                                                                                                                                                                                                                                                                         |



Q6. Rewrite the following C++ program after removing the syntax errors, underline the corrections:

```
i) # include <iostreamh>
 void main();
 {
 int a,b;
 cout<<"Enter two numbers ;
 cin<<a<<b;
 for(i=0; i>-7; i- -)
 { if (a>b)
 a++;
 }
 else
 - -b;
```

```
ii) # include <iostream.h>
 void main();
 { float r;
 int w=90;
 while w>60
 { r = w-60;
 Switch (r)
 {20 :cout<<"Lower Range";
 30 :cout<<"Middle Range"
 40 :cout<<"High Range";
 }
 w=/10;
 }
```

Q7. State the outputs of the following program segments :

```
i) for(int I = 1; i<=6; I++)
 {
 for (int j = 5; j<=1; j++)
 cout<<I*j<<"\t";
 cout<<"\n";
 I++;
 }

ii) char a='A';
 int x;
 while (a!= 'E')
 { x = a;
 do
 { cout<<x<<"\t";
 x++;

 }while(x<=70);
 cout<<"\n"
 a++;
 }
```

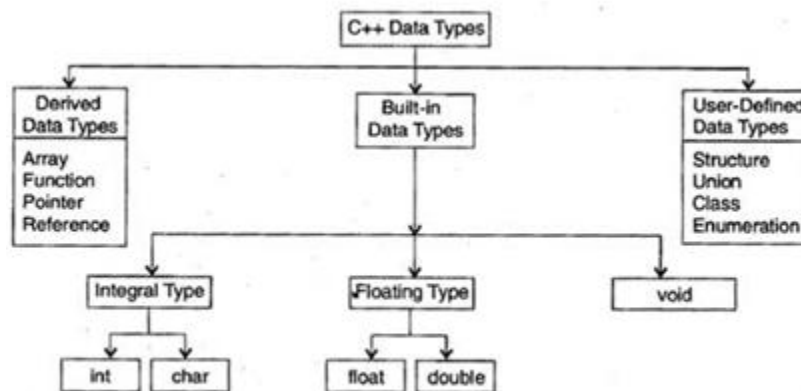
```

iii) longint n = 5, x = 3, s=0,c=0;
 for (inti = 0; i<=n; i++)
 {
 c = 1;
 for(int j =1;j<=i;j++)
 c *= x;
 s+=c;
 }
 cout<<s;

```

## ARRAYS

Array is a derived data type (Data types that are derived from the built-in data types are known as derived data types. The various derived data types provided by C++ are *arrays, functions, references* and *pointers*.)



Various Data Types in C++

➤ Arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.

➤ They are a way to group a number of items into a larger unit.

➤ They can have data items of simple types like int or float or even of user defined types like structures and objects.

- Elements of these arrays will be referred to as `arrayname[n]`, where `n` is the element number in the array.
- Element numbers in `[ ]` are called as subscripts or indices. The subscript or index of an element designates its position in the array's ordering.
- The array subscripts start with number 0 and not with 1. That is, array `A[5]` will be having elements: `A[0]`, `A[1]`, `A[2]`, `A[3]`, `A[4]`

### Types of arrays:

1. One dimensional array - comprised of finite homogeneous elements.
2. Multi-dimensional array - comprised of elements each of which is itself an array. For example -> a two dimensional array

### One Dimensional Array

- It is the simplest form of array.
- The array has to be given a name and its elements are referred to by their subscripts or indices. C++ array's index numbering starts with 0
- An array has to be first defined before it can be used. An array definition specifies a *variable type* and a *name* along with one more feature *size* to specify how many data items the array will contain. The size must be an integer value or integer constant without any sign.

For example: `type array-name[size];`

```
int marks[50];
```

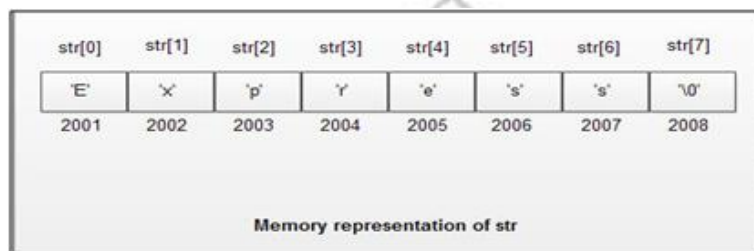
### Memory Representation of 1-D Array

- The elements of 1-D array are stored in contiguous memory location in their index order.
- Thereby the memory used by an array will depend on its data type that we used while declaring it.
- For example: char grade[8] -> In this array, each element will be a char type and its size will be defined as 1 byte (since a character size is 1 byte).
- An array like int age[5] -> will have each element of int type and its size will be 2 bytes.
- Therefore, the amount of storage required to hold an array is directly related to its size and type.

total bytes = sizeof(type) \* size\_of\_array

So, for char grade[8] =  $1 * 8 = 8$  bytes

int age[5] =  $2 * 5 = 10$  bytes



### Address calculation of 1-D Array:

The notation of an array is:

Array\_name[lower bound L, upper bound U]

Arrayname[size];

where size specifies the number of elements in the array and the subscript (index) value ranges from 0 through size-1.

Length of the array = UB - LB + 1

Address of element with subscript I =  $B + S * (I - LB)$

where,

B: Base Address (Address of the very first element)

S: Size of an array element

LB: Lower Bound of array

UB: Upper Bound of array

Example 1: Given an array A[0:15]. If B=1000 and S=2, then calculate the address of A[10]

**Solution:**

Base Address -> B=1000

Size of element -> S=2

LB=0, UB=15

Address of A[I] =  $B + S(I - LB)$

=  $1000 + 2(10 - 0)$

=  $1000 + 2(10) = 1000 + 20 = \underline{1020}$

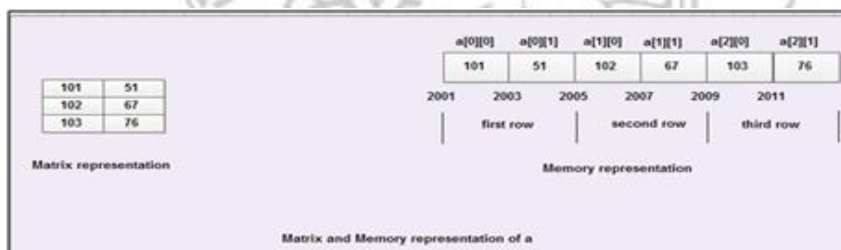
**Two-Dimensional (2-D) Arrays:**

- is an array where each element is itself an array.
- $A[m][n]$  means an array  $A$  with  $M$  rows and  $N$  columns i.e.  $M \times N$  elements. For example:  $A[3][2]$  would mean an array  $A$  with 3 rows and 2 columns i.e.  $3 \times 2 = 6$  elements.
- is always written as *type array\_name[rows][columns];* i.e. `int sale[3][2];`
- Each element of the array `sale` will be referred as `sale[0][0]`, `sale[0][1]`, `sale[1][0]`, and so on
- To process 2-D arrays, we used nested loops. One loop will process rows and the other will process columns. The outer loop is for rows and the inner loop is for columns. For example:

|       | Column 0             | Column 1             |
|-------|----------------------|----------------------|
| Row 0 | <code>a[0][0]</code> | <code>a[0][1]</code> |
| Row 1 | <code>a[1][0]</code> | <code>a[1][1]</code> |
| Row 2 | <code>a[2][0]</code> | <code>a[2][1]</code> |

**Memory Representation of a 2-D Array**

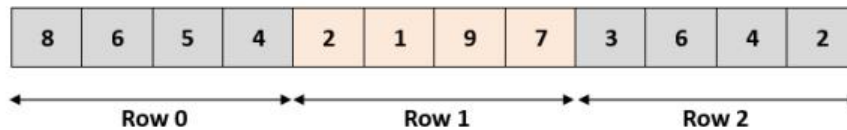
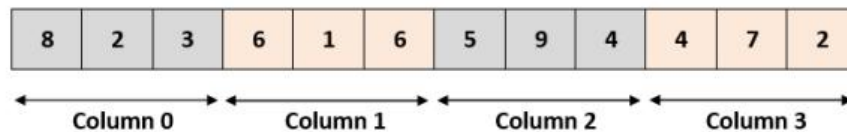
- A 2-D array is stored in a row-column matrix, where the 1st index indicates the row and the 2nd indicates the column.
- For example `int sales[4][5]` will be having  $4 \times 5 = 20$  elements in the memory.
- The amount of storage required to hold a 2-D array is also dependent upon its *base type, number of rows and number of columns*.
- The formula to calculate the total number of bytes for the above array will be :
- Total bytes = number of rows  $\times$  number of columns  $\times$  size of (base type)  
i.e.  $4 \times 5 \times 2 = 40$  bytes
- For array `double P[6][4]`, total number of bytes will be :  $6 \times 4 \times 8 = 192$  bytes.

**Implementation of 2-D array in Memory:**

Elements of a 2-D arrays in memory are allocated contiguous locations. The array elements are stored linearly using one of the following two methods:

|           |   | Column Index |   |   |   |
|-----------|---|--------------|---|---|---|
|           |   | 0            | 1 | 2 | 3 |
| Row Index | 0 | 8            | 6 | 5 | 4 |
|           | 1 | 2            | 1 | 9 | 7 |
|           | 2 | 3            | 6 | 4 | 2 |

Two-Dimensional Array

**Row-Major (Row Wise Arrangement)****Column-Major (Column Wise Arrangement)**

1. **Row Major:** Using this method a 2-D array is stored with all the elements of first row in sequence followed by elements of second row and so on.

Address of the  $[I, J]$ th element =  $B + S * [(I - L_r) * N + (J - L_c)]$  where

B: Base Address (address of very first element)

S: Size of an array element

$L_r$  : Lower bound of row (first row number)

$L_c$  : Lower bound of column (first column number)

N : No of columns

**Example:** An array `mat[15][30]` is stored in the memory with each element requiring 8 bytes of storage. If the base address of V is 5300, find out memory location of `mat[8][12]`, if the array is stored along the row.

**Answer:**

Base address  $B = 5300$

Size of elements  $S = 8$  bytes

Number of rows  $M = 15$

Number of columns  $N = 30$

Lower bound of row  $L_r = 0$

Lower bound of column  $L_c = 0$

In row major implementation

Address of the  $[I, J]$ th element =  $B + S * [(I - L_r) * N + (J - L_c)]$

Address of `mat[8][12]` =  $5300 + 8 * [(8 - 0) * 30 + (12 - 0)]$

=  $5300 + 8 * [8 * 30 + 12]$

=  $5300 + 8 * [240 + 12]$

=  $5300 + 8 * 252$

=  $5300 + 2016$

= 7316

2. **Column Major:** Using this method a 2-D array is stored with all the elements of first column in sequence followed by elements of second column and so on.



Address of the  $[I,J]$ th element =  $B + S*[(I-L_r) + (J-L_c) * M]$  where

B: Base Address (address of very first element)

S: Size of an array element

$L_r$  : Lower bound of row (first row number)

$L_c$  : Lower bound of column (first column number)

M : No of rows

**Example:** An array  $A[15][35]$  is stored in the memory along with column with each of its elements occupying 8 bytes of storage. Find out the base address and address of an element  $A[2][5]$ , if the location  $A[5][10]$  is stored at the address 4000.

**Answer:**

Let the Base address be B

Size of elements  $S=8$  bytes

Number of rows  $M= 15$

Number of columns  $N=35$

Lower bound of row  $L_r =0$

Lower bound of column  $L_c =0$

In column major implementation

Address of the  $[I,J]$ th element =  $B + S*[(I-L_r) + (J-L_c) * M]$

Address of mat  $[5][10] = B + 8*[(5-0) + (10-0) * 15]$

$4000 = B + 8*[5 + 10*15]$

$4000 = B + 1240$

$B = 4000 - 1240$

$B = 2760$

Address of mat  $[2][5] = 2760 + 8*[(2-0) + (5-0) * 15]$

$= 2760 + 8*[2 + 5*15]$

$= 2760 + 616$

$= 3376$

**Note:**

Usually number of rows and columns of a matrix are given ( like  $A[20][30]$  or  $A[40][60]$  ) but if it is given as  $A[L_r - - - - U_r, L_c - - - - U_c]$ . In this case number of rows and columns are calculated using the following methods:

Number of rows (M) will be calculated as  $= (U_r - L_r) + 1$

Number of columns (N) will be calculated as  $= (U_c - L_c) + 1$

And rest of the process will remain same as per requirement (Row Major Wise or Column Major Wise).

**Examples:**

Q 1. An array  $X [-15.....10, 15.....40]$  requires one byte of storage. If beginning location is 1500 determine the location of  $X [15][20]$ .

Number or rows say  $M = (U_r - L_r) + 1 = [10 - (- 15)] + 1 = 26$

Number or columns say  $N = (U_c - L_c) + 1 = [40 - 15]] + 1 = 26$

**Matrices as 2-D Arrays:**

- Matrix is a set of  $mn$  numbers arranged in the form of an array of  $m$  rows and  $n$  columns. Such a matrix is called  $m \times n$  ( $m$  by  $n$ ) matrix.
- Are represented using a 2-D array. For example: `intMat[5][4]`; means a matrix of 5 rows and 4 columns.
- Like 2-D arrays, to declare matrices we have to use a nested loop.

```
int Mat[5][4]; //5 rows , 4 columns
for(int i=0;i<5;i++) //track of rows
{
 for(int j=0;j<4;j++) //track of columns
 {
 cout<<"Enter element for row= "<<i+1<<" and column= "<<j+1;
 cin>>Mat[i][j];
 }
}
```

**Algebra of Matrices:****Rules to be followed:**

1. Addition and Subtraction
  - > It is possible only on 2 identically dimensioned matrices.
  - > If  $A[m][n]$  and  $B[p][q]$  are 2 given matrices, before performing addition or subtraction, we should check that  $m=p$  and  $n=q$
2. Multiplication
  - > If  $A[m][n]$  and  $B[p][q]$  are 2 given matrices, before performing addition or subtraction, we should check that  $n=p$

**Array of Strings:**

- It is a 2D character array
- The size of the 1st index (rows) determines the number of strings and the size of second index(columns) determines the maximum length of each string.
- For example: `char strngs[10][51]`; will mean an array of 10 string each of which can hold 50 characters. (Here the 2nd index is given 51, to take care of the null character '\0')

**Array Initialization**

- Arrays can be initialized at the time of declaration too, just as we initialize variables. For example:

```
typearray_name[size1]={value list};
int days[5]={23,26,31,28,30};
```

- The value list is separated by a comma, and each value will have the same data type as defined for the array.
- In the above example, `days[0] =23`, `days[1]=26` and so on.
- Character arrays can also defined as the same -> `char apr[10]="Program";`
- 2D arrays can also be initialized in the same way. For example,

```
int cube[5][2]={1,10,2,3,20,16,29,26,12,11};
i.e cube[0][0]=1, cube[0][1]=10, cube[1][0]=2 and so on
```

OR

```
char star[7][11]={"Sunday","Monday","Tuesday","Wednesday",
```

```
"Thursday","Friday","Saturday");
```

- C++ allows to skip the size of the array in the array initialization statement. It then automatically creates an array big enough to hold all the initializers present.
- This is called as **unsized array**.
- C++ automatically calculates the dimensions of unsized arrays.

For examples:

```
char S12[]="First String";
```

```
intval[]={2,3,4,5,6,7,8,9} //If you skip the size, you must give a list of initializers so
 that C++ can calculate the size of array y counting them.
```

```
float amount[]={2341.22,345.23,6357.12}
```

```
int cube[][2]={1,10,2,3,20,16,29,26,12,11};
```

- The advantage of this declaration is that we can lengthen or shorten the value list without changing the array dimensions.

### **Calling function with arrays:**

- Arrays are passed by reference and the array name is automatically converted into array pointer.
- C++ interprets an array name as the address of some element
- When an array is used as an argument to a function, only the address of the array gets passed, not a copy of the entire array. When you call a function with an array name, a pointer to the 1st element in the array is passed into the function. That means the parameter declaration must be of compatible type.

**For example:**

- 1) ***The receiving parameter of the array may itself be declared as an array with its size.***
- 2) ***The receiving parameter may be declared as an unsized array***
- 3) ***The receiving parameter can be declared as a pointer***

### **Function Arguments for a Multi-dimensional array**

- When the function argument is an array of more than 1 dimension, we have to declare the size of the dimensions.
- But the size of the first dimension is optional.

For example:

```
void check(int mat[5][6]);
```

OR//both are valid and parameter mat will be converted to array  
//pointer of int type automatically.

```
void check(int mat[][6]);
```

## Searching

We can use two different search algorithms for searching a specific data from an array

1. Linear search algorithm
2. Binary search algorithm

### Linear search algorithm

In Linear search, each element of the array is compared with the given item to be searched for. This method continues until the searched item is found or the last item is compared.

```
#include<iostream.h>
int linear_search(int a[], int size, int item)
{
 int i=0;
 while(i<size&& a[i]!=item)
 i++;
 if(i<size)
 return i; //returns the index number of the item in the array else
 return -1; //item not present in the array so it returns -1 since -1 is not a legal
 //index no.
}
void main()
{
 int b[8]={2,4,5,7,8,9,12,15},size=8;
 int item;
 cout<<"enter a number to be searched for";
 cin>>item;
 int p=linear_search(b, size, item); //search item in the array b
 if(p!=-1)
 cout<<item<<" is not present in the array"<<endl;
 else
 cout<<item <<" is present in the array at index no "<<p;
}
```

In linear search algorithm, if the searched item is the first element of the array then the loop terminates after the first comparison (best case), if the searched item is the last element of the array then the loop terminates after size time comparisons (worst case) and if the searched item is middle element of the array then the loop terminates after size/2 time comparisons (average case). For large size array linear search not an efficient algorithm but it can be used for an unsorted array.

### **Binary Search algorithm**

Binary search algorithm is applicable for already sorted array only. In this algorithm, to search for the given item from the sorted array (in ascending order), the item is compared with the middle element of the array. If the middle element is equal to the item then index of the middle element is returned, otherwise, if item is less than the middle item then the item is present in first half segment of the array (i.e. between 0 to middle-1), so the next iteration will continue for first half only, if the item is larger than

the middle element then the item is present in second half of the array (i.e. between middle+1 to size-1), so the next iteration will continue for second half segment of the array only. The same process continues until either the item is found (search successful) or the segment is reduced to the single element and still the item is not found (search unsuccessful).

```
#include<iostream.h>
int binary_search(int a[], int size, int item)
{
 int first=0,last=size-1,middle; while(first<=last)
 {
 middle=(first+last)/2;
 if(item==a[middle])
 return middle; // item is found else if(item<a[middle])
 last=middle-1; //item is present in left side of the middle element
 else
 first=middle+1; // item is present in right side of the middle element
 }
 return -1; //given item is not present in the array, here, -1 indicates unsuccessful
 search
}
void main()
{
 int b[8]={10,14,19,26,31,33,35,42,44},size=10;
 int item;
 cout<<"enter a number to be searched for";
 cin>>item;
 int p=binary_search(b, size, item); //search item in the array b
 if(p!=-1)
 cout<<item<<" is not present in the array"<<endl;
 else
 cout<<item <<" is present in the array at index no "<<p;
 }
}
```

Using the above algorithm, let us search for a value 31 in this sorted array.

First, we shall determine half of the array by using this formula –

$$\text{mid} = \text{low} + (\text{high} - \text{low}) / 2$$

Here it is,  $0 + (9 - 0) / 2 = 4$  (integer value of 4.5). So, 4 is the mid of the array.

Now we compare the value stored at location 4, with the value being searched, i.e. 31. We find that the value at location 4 is 27, which is not a match. As the value is greater than 27 and we have a sorted array, so we also know that the target value must be in the latter half of the array.

|    |    |    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|----|----|
| 10 | 14 | 19 | 26 | 27 | 31 | 33 | 35 | 42 | 44 |
|    |    |    |    |    | ↓  |    |    |    |    |
| 10 | 14 | 19 | 26 | 27 | 31 | 33 | 35 | 42 | 44 |
| 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  |
| 10 | 14 | 19 | 26 | 27 | 31 | 33 | 35 | 42 | 44 |
| 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  |



We change our low to mid + 1 and find the new mid value again.

$\text{low} = \text{mid} + 1$

$\text{mid} = \text{low} + (\text{high} - \text{low}) / 2$

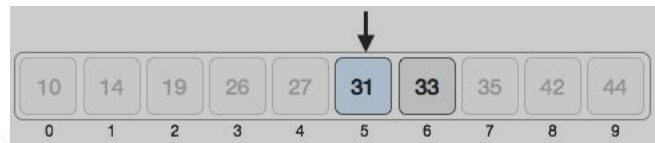
Our new mid is 7 now. We compare the value stored at location 7 with our target value 31.



The value stored at location 7 is not a match, rather it is less than what we are looking for. So, the value must be in the lower part from this location.



Hence, we calculate the mid again. This time it is 5. We compare the value stored at location 5 with our target value. We find that it is a match.



We conclude that the target value 31 is stored at location 5.

Binary search halves the searchable items and thus reduces the count of comparisons to be made to very less numbers.



### Sorting in Arrays

Sorting is nothing but storage of data in sorted order, it can be in ascending or descending order. The term Sorting comes into picture with the term Searching. Sorting arranges data in a sequence which makes searching easier.

We can use different sort algorithms for sorting a data in an array:

1. Bubble Sort
2. Selection Sort
3. Insertion Sort

### Bubble Sort

This sorting algorithm is comparison-based algorithm in which each pair of adjacent elements is compared and the elements are swapped if they are not in order.

We take an unsorted array for our example.



Bubble sort starts with very first two elements, comparing them to check which one is greater.



Now, value 33 is greater than 14, so it is already in sorted locations.

Next, we compare 33 with 27.



We find that 27 is smaller than 33 and these two values must be swapped.

The new array should look like this –



Next we compare 33 and 35. We find that both are in already sorted



positions.

Then we move to the next two values, 35 and 10.

We know then that 10 is smaller 35. Hence they are not sorted.

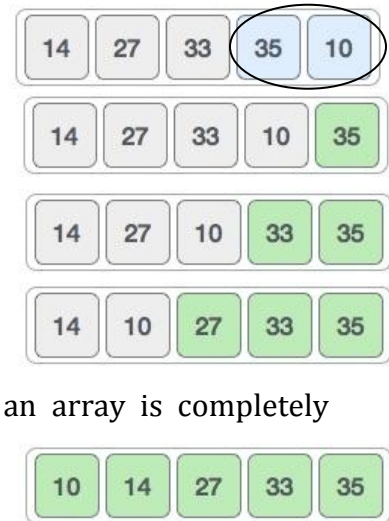
We swap these values. We find that we have reached the end of the array. After one iteration, the array should look like this –

To be precise, we are now showing how an array should look like after each iteration. After the second iteration, it should look like

this-

Notice that after each iteration, at least one value moves at the end.

And when there's no swap required, bubble sorts learns that an array is completely sorted.



### Function for Bubble sort

```
#include<iostream.h>
void BubbleSort(int a[],int size)
{ int temp; // Bubble Sort Starts Here
 for(int i=0; i<=size; i++)
 { for(int j=0; j<size; j++)
 { if(a[j]>a[j+1]) //Comparing adjacent elements
 { temp=a[j];
 //Swapping element in if statement
 a[j]=a[j+1];
 a[j+1]=temp; }
 }
 }
}
```

### Selection Sort

Selection sort is a simple sorting algorithm. This sorting algorithm is an in-place comparison-based algorithm in which the list is divided into two parts, the sorted part at the left end and the unsorted part at the right end. Initially, the sorted part is empty and the unsorted part is the entire list.

The smallest element is selected from the unsorted array and swapped with the leftmost element, and that element becomes a part of the sorted array. This process continues moving unsorted array boundary by one element to the right.

Consider the following depicted array as an example.



For the first position in the sorted list, the whole list is scanned sequentially. The first position where 14 is stored presently, we search the whole list and find that



10 is the lowest value. So we swap 14 with 10.  
After one iteration 10, which happens to be the minimum value in the list, appears in the first position of the sorted list.

For the second position, where 33 is residing, we start scanning the rest of the list in a linear manner. We find that 14 is the second lowest value in the list and it should appear at the second place. We swap these values.

After two iterations, two least values are positioned at the beginning in a sorted manner.

The same process is applied to the rest of the items in the array. Following is a pictorial depiction of the entire sorting process –

### Function for Selection Sort

```
void selectionSort(int a[], int size)
{
 int i, j, min, temp;
 for(i=0; i < size-1; i++)
 {
 min = i; //setting min as i
 for(j=i+1; j < size; j++)
 {
 if(a[j] < a[min])
 //if element at j is less than element at min position
 {
 min = j; //then set min as j
 }
 }
 temp = a[i];
 a[i] = a[min];
 a[min] = temp;
 }
}
```

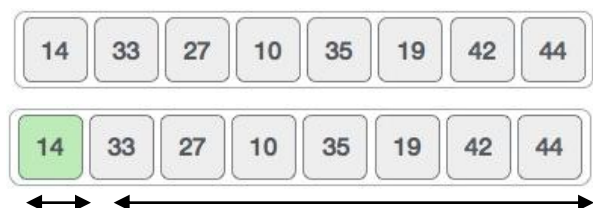
### Insertion Sort

Here, a sub-list is maintained which is always sorted. For example, the lower part of an array is maintained to be sorted. An element which is to be 'insert'ed in this sorted sub-list, has to find its appropriate place and then it has to be inserted there. Hence the name, insertion sort.

The array is searched sequentially and unsorted items are moved and inserted into the sorted sub-list (in the same array).

We take an unsorted array for our example.

Initially the first element alone comprises the sorted array and hence is sorted by itself. For



now, 14 is in sorted sub-list.

Sorted

Unsorted

Insertion sort moves ahead and tries to find the correct place for the next element 33 which is to the right of 14. Hence no movement.



Next iteration 27 is the current element and its correct position is between 14 and 33.



It shifts 33 right and moves 27 between 14 and 33 hence, the sorted sub-list remains sorted after swapping.



Next, it finds the correct position for 10 which will be at the left of 14.



We shift 33, 27 and 14 right and insert 10 right in the beginning. So, by the end of third iteration, we have a sorted sub-list of 4 items.



Unsorted

### Function for Insertion Sort

```
void insertionSort(int a[], int size)
```

```
{ int i, j, temp;
 for (i = 1; i <= size; i++)
 { j = i-1;
 temp = a[i];
 while (j >= 0 && temp < a[j]) //shift elements right till you find a smaller element
 { a[j+1] = a[j];
 j--;
 }
 a[j+1] = temp;
 }
}
```

### Assignment No : 7

Arrays

- Q1. Define arrays. Give one example.  
 Q2. What is the base type of an array?  
 Q3. What are strings?  
 Q4. Which character terminates the string?  
 Q5. State the outputs of the following program segments :

a) 

```
chararr [] = "Technology";
for (inti= strlen(arr)-1; i>=0 ; i--)
arr[i] = arr[i+1];
cout<< "The output is : " <<arr;
```

b) 

```
charstr [] = "ComPutErS";
intarr[15];
for(int j=0; j<strlen(str);j++)
{
if((int)str[j]%2==0)
arr[j]= (int)str[j];
else
if(islower(str[j]))
arr[j]=1;
else
arr[j]=0;
}
for(int j=0; j<strlen(str);j++)
{
cout<<arr[j]<< ",";
}
```

c) 

```
int a [5] = {3, 10, 1, 20, 25 } ;
inti, j, k, m;
i = a[0]++;
j = a[i++];
k = ++ a[2];
m = ++a[k] + a[i]++ ;
cout<<i<< " "<<j<< " "<<k<< " "<<m;
```

Q 6. Rewrite the following program after removing all the errors. Underline the corrections:

a) 

```
include<iostream.h>
void main()
{
intarr ={ '3', '4', '6', '7', '8' };
for(int l= 1; l<5; l++)
{
gets(arr[l];
}
}
```



```
b) #include<iostream.h>
void main()
{
 char name [] ="string" ;
 for(int I= 1; I<5; I++)
 {
 If(arr[I]=="i") puts(arr[I]);
 }
}
```

Q 7. Give Array declarations for the following :

- To store prices of 50 items.
- To store the scores of 25 players in 3 different matches.
- To store the names of 15 employees of a firm.
- To store the dates of admission of 5 patients in a hospital. Initialise this array with some valid data.

Q 8. Write the statements (only) to carry out each of the following operations :

- To count the number of positive elements in an array of 20 integers.
- To count the frequency of alphabet 'a' in a string.
- To replace all blank spaces by '!' in a string.

Q 9. Write all the passes to perform:

- Bubble Sort
- Selection Sort

on the following Array :

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| A | e | X | C | a | Z | y | R |
|---|---|---|---|---|---|---|---|

## FUNCTIONS



A function is a subprogram that acts on data and often returns a value.

- >Makes program more readable
- >can be invoked in other parts of the program
- >reduces program size

### Types of functions

1. Built in functions
2. User defined functions

### Built in functions

- >use standard library of C++.
- >A library is a collection of subprograms used to develop software.
- >They are not independent programs. They are called Helper code and are used in other independent programs.
- >C++ library of functions store functions of same category (e.g. mathematical, string, etc) under separate files known as header files.

I] `stdio.h` -> Header files provide function prototypes definition, definitions for library functions.

e.g `getchar()`, `putchar()`, `gets()`, `puts()`

II] `string.h` -> declares several string manipulation and memory manipulation routines.

e.g `strcat`, `strcpy`, `strlen`, `strcmp`

III] `math.h` -> declares prototypes for the math function and math error handlers.

e.g `fabs()`, `pow()`, `sqrt()`, `sin()`, `cos()`, `abs()`

IV] `stdlib.h` -> declares several commonly used routines like conversion routines, search/sort routines

e.g `randomize()`, `random()`

V] `iostream.h` -> declares basic C++ stream I/O routines

e.g `get()`, `put()`, `getline()`, `write()`,

VI] `iomanip.h` -> declares C++ streams I/O manipulators

e.g -> `setw`, `setprecision`

Character functions: Use header file -> **cctype.h**

| Name                   | Description                                                                         |
|------------------------|-------------------------------------------------------------------------------------|
| <code>isalnum()</code> | Checks if the given argument is alphanumeric character and returns a non zero value |
| <code>isalpha()</code> | Checks if the given argument is an alphabet and returns a non zero value            |
| <code>isdigit()</code> | Checks if the given argument is a digit(0-9) and returns a non zero value           |
| <code>isupper()</code> | Checks if the given argument is a uppercase alphabets and returns a non zero value  |
| <code>islower()</code> | Checks if the given argument is a lowercase alphabets and returns a non zero value  |
| <code>tolower()</code> | Returns the lowercase of the parameter.                                             |

|           |                                         |
|-----------|-----------------------------------------|
| toupper() | Returns the uppercase of the parameter. |
|-----------|-----------------------------------------|

String functions: -> use header **string.h**

|    |                                              |                                                                                                                 |
|----|----------------------------------------------|-----------------------------------------------------------------------------------------------------------------|
| 1. | char strcat(char str1, const str2)           | Concatenates a copy of str2 to the end of str1. str1 must have enough reserved elements to hold both strings.   |
| 2. | int strcmp(const char str1, const char str2) | Alphabetically compares 2 strings and returns -ve value if str1<str2, 0 if str1=str2 and +ve value if str1>str2 |
| 3. | char strcpy(char str1, const str2)           | Copies the contents of str2 into str1. str1 must have enough reserved elements to hold str2.                    |
| 4. | int strlen(char str)                         | Returns the len of the string str (doesn't count the Null -> a string is terminated with a null value \n)       |

Mathematical Functions -> use header file **math.h**

|    |        |                                                             |
|----|--------|-------------------------------------------------------------|
| 1. | fabs() | Returns a float absolute value -> fabs(1.0) gives 1.0       |
| 2. | pow()  | Returns base raised to exp power -> pow(2,2) gives answer 4 |
| 3. | sqrt() | Returns the square root of a number -> sqrt(81) gives 9     |
| 4. | sin()  | Returns sin of an argument. Value of arg in radians         |
| 5. | cos()  | Returns cos of an argument. Value of arg in radians         |
| 6. | abs()  | Returns an absolute value -> abs(1.0) gives 1               |

Standard Input/Output functions -> use header file **stdio.h**

|    |           |                                                                                                                            |
|----|-----------|----------------------------------------------------------------------------------------------------------------------------|
| 1. | gets()    | Accepts a string of characters entered at the keyboard and places them in a string variable. -> char name[21]; gets(name); |
| 2. | puts()    | Writes the string on the screen and advances the cursor to the newline. -> puts(name)                                      |
| 3. | getchar() | Reads a character from the keyboard -> char ch; ch=getchar();                                                              |
| 4. | putchar() | Prints a character on the screen -> putchar(ch);                                                                           |

Stream Input/Output functions -> use header file **iostream.h**

|    |       |                                                                                                                         |
|----|-------|-------------------------------------------------------------------------------------------------------------------------|
| 1. | get() | Fetches a single character and stores it in a character variable. It is used with <b>cin</b> . -> char ch; cin.get(ch); |
|----|-------|-------------------------------------------------------------------------------------------------------------------------|

|    |           |                                                                                                                                                                                                                                                                                                                                                                                                 |
|----|-----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 2. | put()     | Used to output a character at a time. Used with <b>cout</b> . ->cout.put(ch)                                                                                                                                                                                                                                                                                                                    |
| 3. | getline() | Handles line oriented input function. It reads a line of text. Syntax ->cin.getline( <i>line</i> , <i>size</i> ) where <i>line</i> refers to the variable name and <i>size</i> refers to the maximum <i>size-1</i> characters that can be stored in it. For example : char name[30]; cin.getline(name,30); It can read white spaces and new line characters which <i>cin&gt;&gt;name</i> cannot |
| 4. | write()   | Handles line oriented output function. Displays an entire line. For example ->cout.write(name,30). But if you give cout.write(name, 10), it will display only the 1st 10 characters from the variable char name[30];                                                                                                                                                                            |

Other functions ->use header file **stdlib.h**

|    |                      |                                                                                                                                                                                                  |
|----|----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1. | random( <i>num</i> ) | It generates a random number within range 0 to num-1. For example random(10) will generate numbers between 0-9. But this will generate the same random numbers even if you rerun the code.       |
| 2. | randomize()          | Initialises the random number generator with a random number. It will generate different random numbers everytime. randomize() is a built in function, it has no arguments and returns no value. |

A **function** is a subprogram that acts on data and often returns a value.

- >Makes program more readable
- >can be invoked in other parts of the program
- >reduces program size

Types of functions are:

1. Built in functions
2. User defined functions

**User Defined functions:**

- > are functions created by you - the programmer.
- >are created as per the requirements of the program
- >the main() is called as the calling code
- >each function has its own name. When that name is encountered in a program, that function is executed. When the execution of the function is over, execution returns back to calling code.
- >it is basically a group of program statements that can be invoked from other parts of the program as when required

**1. Define a function**

- function must be defined before using it.
- can be used anywhere in the program.
- written as  
type function-name (parameter list)

{body of function }

- type will specify the type of value that the return statement of the function returns. (can be any valid C++ data type)
- If no type is mentioned, compiler assumes that the function returns an integer value.
- parameter list is a comma separated list of variables of a function referred to as its arguments. A function can be without any parameters too.
- A function definition must have a return statement.

Example:

```
#include<iostream.h>
void display()
{
 cout<<"Welcome to functions";
}
void main()
{
 display();
}
```

## 2. Function Prototype

- describes the function interface to the compiler by giving details such as number and type of arguments and type of return values.
- a declaration of a function made (\*A declaration introduces a function name to the program where as a definition is a declaration that also tells the program what the function is doing and how it is doing)

Example:

```
#include<iostream.h>
void display(); ----->prototype (tells the program that the function is defined somewhere)
void main()
{
 display();
}
void display()
{
 cout<<"Welcome to functions";
}
```

## 3. Use of void

- Used as a return type for functions that do not return a value.  
for e.g. void display(); or type function\_name(void) -> means empty parameter list
- By default, the type specifier of a function will take int if nothing is mentioned.
- Global prototype ->
- Local prototype ->

Example:

```
#include<iostream.h>
int sum(int n1, int n2); ----->prototype (declaration)
void main(){
 int result;
 result=sum(20,30) -----> calling of function
 cout<<"Th result is "<<result<<"\n";
}
int sum(int a, int b) -----> (definition of function)
{
 int s;
 cout<<a<<"\n";
 cout<<b<<"\n";
}
```

```
s=a+b;
return s; }
```

#### 4. Accessing/ Calling a function

➤ function is called by providing the function name & its parameter list

Program to print a cube of a number :

```
#include<iostream.h>
void main()
{
float cube(float); -----> prototype
float x, y;
cout<<"Enter the number ";
cin>>x;
y= cube(x);
cout<<"The cube of number "<<x<<"is "<<y;
}
float cube(float a)
{
float n;
n=a*a*a;
return n;
}
```

#### Default Arguments:

➤ Default arguments can also be given in function.

➤ For e.g float interest(float principal, int time, float rate=0.50); So, while calling this function we can say si\_int=interest(50000,2) where si\_int is a float variable. In this the 3rd argument is not mentioned which means that it will take the default value from the prototype i.e. rate=0.50

➤ Impt: Any argument cannot have a default value unless all arguments appearing on its right side have their default values

Examples:

float interest(float principal, int time, float rate=0.50); -->correct

float interest(float principal, int time=2, float rate); -->incorrect

float interest(float principal=2000, int time=2, float rate); -->incorrect

float interest(float principal, int time=2, float rate=0.50); -->correct

float interest(float principal=2000, int time=2, float rate=0.50); -->incorrect

➤ Constant values can also be given in a function. But the function cannot change these values. For e.g.intsum(constint a, constint b);

**5. Call by Value** ->In this method, copies of values of actual parameters into formal parameters are made. That means the function creates its own copy of argument values and then uses them.

->Changes are not reflected back to the original values.

->Function does not have access to the original variables(actual parameters)

->Advantage ---> one cannot alter the variables that are used to call the function because any change that occurs inside function is on the function's copy of argument value. The original value remains intact.

->Any change in the formal parameter is not reflected back to actual parameter.

**6. Call by Reference** --> Instead of passing a value to the function being called, a reference to the original variable is passed.

- A reference is an Alias (a different name) for a predefined variable.
- Basically the same variable's value can be accessed by either original variable's name and the reference name. Both refer to the same storage area.
- When a function is called by reference, then the formal parameters become references to the actual parameters in the calling function.
- It does NOT create its own copy of original values, it REFERS to the original values only by different names i.e references.
- The called function works with original data and any change in the values gets reflected to the data.
- This method is useful in situations where the values of the original values are to be changed using a function.
- To use call by reference & operator should be used in the prototype and declaration.
- Function can also be invoked using mix of both methods. For example to calculate Simple Interest we can use function such as  
`intsimpleint(float, int&, int&);`  
 Here, the 1st parameter(Principal) is passed with value and the other 2 parameters (time and rate) are passed with reference.

## 7. The Return Statement

- It not only terminates the function's execution but also passes the control back to the calling function.
- An immediate exit from the function is caused as soon as a return statement is encountered and the control passes back to the operating system
- It is used to return a value to the calling code.
- All functions except those of type void return a value.
- There are 3 types of functions in C++:
  - ➔ Computational Functions - functions that calculate or compute some value and return the computed value. e.g `sqrt()`, `cos()`
  - ➔ Manipulative Functions - functions that manipulate information and return a success or failure code. If value 0 is returned, it means successful operation.
  - ➔ Procedural Functions - functions that perform an action and have no explicit return value, e.g `exit()` function is a procedural function.
- When the type of function is not explicitly declared, C++ assumes it to be int type. Hence, if functions are returning values of different data types, explicit type specifier must be given.
- Return by Reference : A function can also return by reference. For example:
 

```
float&min (float &a, float &b)
{
```



```

if (a<b)
return a;
else
return b;
}

```

Here the function min() will return a reference to a float type of variable i.e. it returns a reference to a or b rather than returning values. A function call min(x,y) will give a reference to either a or b depending on which one is lesser of the two. It can then assign a value to that reference. i.e min(x, y)= -5; will assign -5 to the lesser of the two , x or y.

### 8. Scope Determination:

➤ The program part in which a particular piece of code or a data value can be accessed is called as a Variable Scope.

➤ Function Scope : (a) If a function's declaration (sum() )is made within another function (calculate() ), then the function sum() is locally available to function calculate() only and nowhere else. These function scope is a local prototype.

(b) If the same sum() function is declared outside all the other functions such as main(), it can be accessed from any functions in the file. These functions are called global prototypes.

➤ Variable Scope : (a) If a variable declaration appears outside all the functions, it is said to be global variable. This variable is available to all functions and blocks defined in the file. A global variable comes into existence when the program execution starts and destroyed when the program terminates. They hold values throughout program execution. (b) If a variable is defined within a function, they are called as local variables. A local variable comes into existence when the function is entered and is destroyed upon exit. A local variable cannot hold its values between function calls.

➤ Lifetime : The time interval for which a particular variable or data value lives in memory is called Lifetime or variable or data value. This means -> a variable's life time is its parent-block-run i.e. as long as its parent block is executing, the variable lives in the memory.

➤ A local variable can have the same name as that of a global variable. In such a case, the function having a local variable with the same name of a global variable, cannot access that global variable. For example:

```

#include<iostream.h>
int a=20;
void main()
{
int a=50;
cout<<a;
}

```

Output shown will be 50

Here the global variable is hidden and the value of the local variable is taken. If you want to use the global variable inside a function, you have to use the scope operator (::).

```

#include<iostream.h>
int a=20;
void main()
{
int a=50;
}

```

```
cout<<::a<<"\n"<<a
}
```

Output shown will be

20

50

### **Assignment No : 8**

#### **Functions**

Q 1. What are Functions? Give Example.

Q 2. What is Function Prototype? Give Example.

Q 3. Differentiate between Call by Value and Call by Reference. Give explanatory Example.

Q 4. Give the output of the following program :

i) 

```
void display(int x, int y=5)
{
 charstr[20] = "WORLD"
 for(int l=0; l< x; l++)
 { for(int j= l; j<y; j++)
 cout<<str[j];
 cout<<"\n";
 }
}
void main()
{
 inti=3, j =5;
 display(j);
 cout<< "New Display";
 display(i ,j);
}
```

ii) 

```
#include<iostream.h>
int a=10;
void main()
{
 void demo(int&, int, int);
 int a=20, b=5;
 demo(::a, a, b);
 cout<<::a<<a<<b<<endl;
}
void demo(int&x, int y, int z=0)
{
 a+=x;
 y*=a;
 z=a+y;
 cout<<x<<y<< z<<endl;
}
```

```

iii) void show(int&y)
 {
 static int x=0;
 cout<<x++<<"\t";
 x=x-y++;
 }
 void main()
 {
 int k=10;
 show(k);
 cout<<k<<"\n";
 show(k);
 cout<<k<<"\n";
 show(k);
 }

iv) void Change(char msg [], intlen)
 {
 for(int c= 0; c<len; c++)
 {
 if(islower(msg[c]))
 msg[c] = msg[c] - 32;
 else if(isupper (msg[c]))
 msg[c] = msg[c] + 32;
 else if(isdigit (msg[c]))
 msg[c] = msg[c] + 1;
 else
 msg [c] = '*';
 }
 }
 void main ()
 {
 charstr [] = "Pre - Boards 2005-06", t;
 Change(str, strlen(str));
 puts(str);
 for (int c = 0, R = strlen(str) -1; c<strlen(str)/2 ;c++, R--)
 {
 t = str[c] ;
 str [c] = str [R];
 str [R] = t ;
 }
 puts(str);
 }

v) float calc(int&y, float x=12.5)
 {
 y*=x;
 cout<<x<<"\t"<<y ++<<"\t"<<--y<<endl;
 return x;
 }

```

```

void main()
{
 int k=10; float n=2.5;
 n=calc(k);
 cout<<k<<"\t"<<n<<"\n";
 k=calc(k,n);
 cout<<k<<"\t"<<n<<"\n";
}

```

Q 5. Rewrite the following codes after removing the errors. Underline your corrections.

i)        # include<iostream.h>  
 int Z=122;  
 void main()  
 {  
     int x=12; y=89;  
     z+=x;  
     puts( Z);  
 }  
 int add()  
 {  
     Z= x+y;

ii)        # include<iostream.h>  
 int calculate(int s=0, y=2, z)  
 void main()  
 {  
     int x=1,y=3;  
     z=calculate(1,2,3);  
     z=calculate(z);  
 }  
 int calculate(int s=0, y=2, z)  
 {  
     return(s\*y\*z);  
 }

Q 6. Write only the function Prototypes for the following :

- i)        To return the sum of all elements of a matrix.
- ii)       To display all prime numbers which are greater than **N** and less than **M**.

### Structures

A data structure is a group of data elements grouped together under one name. These data elements, known as *members* can have different types and different lengths. Data structures can be declared in C++ using the following syntax:

```

structtype_name {

```

```
member_type1 member_name1;
member_type2 member_name2;
member_type3 member_name3;
.
.
} object_names;
```

Where type\_name is a name for the structure type, object\_name can be a set of valid identifiers for objects that have the type of this structure. Within braces {}, there is a list with the data members, each one is specified with a type and a valid identifier as its name.

For example:

```
struct product {
int weight;
double price;
};
```

```
product apple;
product banana, melon;
```

This declares a structure type called product, and defines it having two members: weight and price, each of a different fundamental type. This declaration creates a new type (product), which is then used to declare three objects (variables) of this type: apple, banana, and melon. The member of the Structure is always Accessed with the help of the .(dot) Operator or if we want to access a variable of the structure then we must use the .(dot) Operator. The Member of the Structure are always Accessed through the Structure variables or For Accessing the Elements of the Structure First you have to create the instance of the Structure then with the help of the Structures you can use the any variable that is defined in the Structure.

for declaring the structure we can use this

```
structstu
{
char name[10],lastname[10];
int age;
float height;
}
```

in this structure will consume the 26 bytes of the Memory

10 for name

10 for Lastname

2 for Age

4 For Height

As you can see the Memory size of the Structure is equals to the Number of Bits of particular data types consume and the Total Memory Size is Judged by the Total Number of Variables with their Respective memory Size Like For

An Integer consume 2 Bytes in the Memory

An Character Consume 1 Byte In Memory

An Float Consume 4bytes in Memory.

### Assignment No.9

#### Structures

- Q 1. Define Structures. Give one example.
- Q 2. What are nested structures? Give one example.
- Q 3. What is the use of typedef. Explain with the help of an example.
- Q 4. What are the uses of #define pre processor directive? How is it different from #include pre processor directive. Give example to support your answer.
- Q 5. What are macros? Give an example to illustrate the use of a macro.
- Q 6. What is the mistake in the following structure declarations ?
- ```
Struct distance
{
    int feet;
    int inches;
}
distance d1;
```
- Q 7. Identify the error in the following structure declaration :
- ```
Struct
{
 int day;
 int month;
 int year;
} bdate;
bdatedoj, dob;
```
- Q 8. Give appropriate declarations for the following :
- An array temp to store 30 temperatures along with their corresponding dates {dd/mm/yy}.
  - A structure specification that includes two structure variables – distance and time. The distance include two variables- both of type float- called feet and inches. The time includes three variables- all of type int – called hrs, mins, and secs. Initialize such a structure with values 1350.05 feet, 8.5inches, 10 hrs, 51 mins and 17 secs.
  - An array of 5 structures that store rollno, name, marks in 3 subjects ( use array of three integers) for students. Also initialize the array with some values.
- Q 9. Consider the following :
- ```
Struct item
```



```

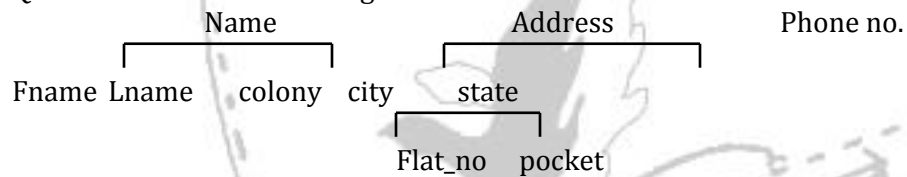
{
    int l_no;
    char l_name[20];
    float price[4];
};
itemAcat[3]= {
    { { 1001, "Bread", {7.50,8.00,8.00,9.25}},
      { { 1002, "Butter", {14.25,15.75,15.00,16.25}},
        { { 1003, "Cake", {21.50,21.00,28.05,119.25}},
          };

```

Using the above information answer the following :

- Acat[1].l_no = _____
- Acat[0].price[2] = _____
- Acat[1].l_name[3] = _____
- Acat[2].l_no = _____
- Acat[2].l_no[1] = _____

Q 10. Declare the following structure :



Q 11. Find out the errors, if any, in the following program :

```

void main()
{
    struct stud
    {
        char name[25];
        int age;
        int rollno;
    }
    stud s1;
    strcpy(stud.name, "Haris");
    age=17;
    cout<<name;
    cout<<stud.age;
}

```

Q 12. What will be the output of the following program :

```

a) structdef
{
    inti;
    int f;
};
void func(def d);
void main( )
{
    def d;
    d.i=100;
}

```

```

        d.f=1.5;
        func (d);
        cout<<d.i<<d.f;
    }
    void func(def d)
    {
        d.f=2.5;
        cout<<d.i<<d.f;
    }

```

b) `#include<iostream.h>`
`struct play`
`{`
 `int score, bonus;`
`};`
`void calculate(play &p, int n = 10)`
`{`
 `p.score ++;`
 `p.bonus+=n;`
`}`
`void main()`
`{`
 `play pl = {10,15};`
 `calculate(pl,5);`
 `cout<<pl.score<< " : "<<pl.bonus<<endl;`
 `calculate(pl);`
 `cout<<pl.score<< " : "<<pl.bonus<<endl;`
 `calculate(pl,15);`
 `cout<<pl.score<< " : "<<pl.bonus<<endl;`
`}`

Q 13. In the following C++ program if the value inputted by the user is 20 , what maximum and minimum value the program could possibly print? `# include<stdlib.h>`

```

# include<iostream.h>
void main( )
{
    int N, guessnum;
    cin>>N;
    randomize( );
    guessnum=random(N-10)+10
    cout<<guessnum ;
}

```

Q 14. In the following C++ program what is the expected value of MyMarks for the options (i) to (iv) given below:

```

# include<stdlib.h>
# include<iostream.h>
void main( )
{

```

```
randomize( );
int marks[ ]={99,92,94,96,93,95}, MyMarks;
MyMarks = Marks[1+random(2)];
Cout<<MyMarks<<endl;
}
```

(i) 99
(ii) 94
(iii) 96
(iv) None of the above

Program List No. 1

Getting Started With C++

Sample Program :

```
/* WAP to find the sum of two numbers
Name : XYZ
Date : 20/4/06 */
```

```
# include<iostream.h>
# include<conio.h>
```

```
void main( )
{
    clrscr( );
    int x, y, sum;
    cout<< "Enter Two numbers : ";
    cin>>x>>y;
    sum = x + y;
    cout<< "The sum of "<<x<< " and "<<y<< "is ="<<sum;
    getch( );
}
```

Now, write the following Programs :

- 1) WAP to input two sides of a rectangle and calculate its area.
- 2) WAP to input five numbers and calculate their average.
- 3) WAP to input the radius of a circle and find its area.
- 4) WAP to input two numbers and swap them.
- 5) WAP to input Principal, Rate and Time and calculate simple interest.
- 6) WAP to input roll number and marks in 5 subjects of a student and calculate its percentage.
- 7) WAP to input a number and calculate number of years, number of months and number of days in it.

(e.g. number = 475

it has, years =1, months = 3, days =20)

Program List No. 2
Selection Statements and Loops

- 1) WAP to input three numbers and print the greatest.
- 2) WAP to print the roots of a quadratic equation.
- 3) WAP to input a year and check whether it is a leap year or not
- 4) WAP to input marks of a student and print the grades.

<u>Marks Range</u>	<u>Grade</u>
90-100	A
80- 89	B
70 – 79	C
60 – 69	D
50 – 59	E
Below 50	F

- 5) WAP to simulate Arithmetic Calculator.
- 6) Write a menu driven Program to find the area of a circle, rectangle, square, triangle.
- 7) WAP to display first 10 natural numbers.
- 8) WAP to display first 20 even numbers.
- 9) WAP to display the sum of first 10 natural numbers.
- 10) WAP to input a number and print its factorial.
- 11) WAP to calculate a^b .
- 12) WAP to input a number and print its table up to 10 multiples.
- 13) WAP to input a number and print all its factors.
- 14) WAP to print first n terms of a Fibonacci series.
- 15) WAP to find the maximum and minimum of n inputted numbers.
- 16) WAP to input a number and display the number of digits in it.
- 17) WAP to input a number and print the sum of all its digits.
- 18) Write a menu driven program to input a number and
 - a) Reverse it.
 - b) Check if it is a palindrome or not.
 (Repeated execution)

Program List No. 3
Nested Loops

- 1) WAP to display the following format on the screen.

```

1
1   2
1   2   3
1   2   3   4
  
```

- 2) WAP to display the following format on the screen.

```

      1
1     1     1
  
```

- 1 1 1 1 1
1 1 1 1 1 1 1
- 3) WAP to print the sum of the following series

$$1^3 + 3^3 + 5^3 + 7^3 + \dots + (N)^3$$

- 4) WAP to print the sum of the following series

$$1 - \frac{x^1}{2!} + \frac{x^2}{3!} - \frac{x^3}{4!} + \dots + (-1)^n \frac{x^n}{(n+1)!}$$

- 5) WAP to display first n prime numbers.
- 6) Write a program to find the Armstrong number.
E.g., $153 = 1^3 + 5^3 + 3^3$
- 7) WAP to convert a binary number to its decimal equivalent.
- 8) WAP to convert a decimal number to its binary equivalent.
- 9) WAP to convert a binary number to its octal equivalent.
- 10) WAP to convert an octal number to its binary equivalent.
- 11) A computerized ticket counter of an underground metro rail station charges for each ride at the following rate

AGE	Amount/Head
18 or above	Rs. 5
5 or above but below 18	Rs. 3
Below 5	Rs. 1

Write a C++ program, which takes as input the number of people of various age groups and prints a ticket. At the end of the journey, the program, states the number of passengers of different age groups who traveled and total amount received as collection of fares.

- 12) WAP To input the length and breadth for a box and draw the box of that size.

Program List-4

(1-D Arrays and Functions)

- Q 1. Write a program to execute the following function :
- Reads an integer and an integer array and search for that integer in the integer array and returns its position, in case the number is not there in the list, function should return -1.(LINEAR SEARCH)
- Q 2. Write a program to execute the following function :
- Reads a string and return its length.
- Q 3. Write a program to execute the following function :
- Reads a string and calculate and return number of words in the string.
- Q 4. Write a program to execute the following function :
- Reads a string and check if it is a palindrome or not, in case the string is a palindrome function should return 1 otherwise function should return 0.
- Q 5. Write a menu driven program to execute the following functions :
- Reads an array of integers, an integer and number of integers in the array and delete that integer from the array of integers.

- Reads an array of integers, an integer and number of integers in the array and insert that integer into the array of integers.

Note : Both the functions should work on the same/array of integers, hence changes made by one function should be reflected to all other functions.

Hint : For number of integers use call by reference

Q 6. Write a program to execute the following function :

- To insert a substring into another string at a given position. The function should read two strings and the position where the string is to be inserted. The function should return 0 in case the position was not valid or there is not enough space in the main string for insertion otherwise function should return 1.

Q 7. Write a menu driven program to execute the following functions :

- Reads two strings and copy the contents of one string to another. (Do not use the pre defined function)
- Reads two strings concatenate them to form the third one and display it. (Do not use the pre defined function)

Q 8. Write a menu driven program to execute the following functions :

- Sort one dimensional integer array using selection sort.
- Read an integer, integer array (already sorted using the first function) and number of integers in the array and insert that integer in the array (without disturbing the order also change in size should be reflected back).

Q 9. Write a menu driven program to execute the following functions :

- Sort one dimensional integer array using Selection sort.
- Reads an integer and an integer array (already sorted using the first function) and search for that integer in the integer array and returns its position, in case the number is not there in the list, function should return -1. (USING BINARY SEARCH)

Q 10. Write a menu driven program to execute the following functions :

- Sort one dimensional integer array using Insertion sort.
- Reads two integer arrays (already sorted using the first function) and merge the two arrays to form a third array. The function should display the contents of the third array.

Note : The elements should be placed in the third array in an orderly manner.



Program List-5**(2-D Arrays and Structures)**

Q 1. Create a header File **matrix.h** with the following Specifications :

Define a structure Matrix with the following data members:

- A 5x 5 integer matrix
- Number of columns
- Number of rows

Now define the following functions

- Input a matrix
- Display a matrix

Q 2. Write program(s) to execute the following functions:

(Use header **matrix.h** to input and display the matrices)

- Print the sum of each row of a matrix
- Print the sum of each column of a matrix
- Print all the elements of the left diagonal of a matrix
- Print all the elements of the right diagonal of a matrix
- Transpose a matrix of and print the new rows and columns
- To display the Upper right and the Lower left triangle of an array of size m x n
- To check the equality of two matrices of size 3 x3 each
- Add two matrices
- Subtract two matrices
- Multiply two matrices

The above functions (wherever applicable) should return the status telling whether the task could be done or not.

Project Assignment

Q 1. WAP in C++ to computerize the **Billing System** of a Restaurant with help of the following functions :

- int password (char usr_name[], char pwd[]);** //checks for the validity of the user name and password and return the status; i.e., returns **1** in case password is valid and **0** in case password is invalid.
- void Menu (char list[5][20], float price[5]);** // displays the name of the Restaurant at the top and then the list of each item along with the price per unit of that item in the following format.

Hungry Kya ??

Menu Card

Item	Price per unit
1. Burger	50
2. Patty	25
3. Sandwich	20
4. Pepsi	15
5. Tea	10

Note : The above given Menu is a sample menu you can have your own menu.

Now, using the above functions write the program to generate the following screens:

Screen 1 :

1. Input the user name.
2. Input the password.
3. Check for the validity of the password; user should be given three chances to input the correct password.

Note : Correct user name and password should already be already stored in the program in the **password** function.

After the valid password has been entered :

Screen 2 :

Input the items to be there in the restaurant along with their prices and store them in 2-D character array and 1-D float array respectively.

Screen 3 :

1. Display the Menu Card using the **Menu** function.
2. Input the item to be ordered by the customer.
3. Input the quantity of that item.

Above three steps should be repeated till the customer wants to order more.

Screen 4 :

Display the Bill in the following format :

Hungry Kya ??

	Price per unit	Quantity	Value
1. Burger	50	2	100
2. Sandwich	20	1	20
3. Pepsi	15	3	45
TOTAL	Rs.	165	

Note : The above given Bill is a sample menu your bill would depend on customers choice.