

Energized Study Material of **Computer Science XI**

BY



Mr. Javeed Hassan Sofi (MCA/BEd)

Teacher GMS Brar Bandipora



A complete energized Study material of Computer
Science for Class XI



Prepared by DSEK and SIE Kashmir



Copyright © 2020 by DSEK, SIE Kashmir

As per Latest Syllabus of JKBOSE

Contents

Structure	Page No's
Unit 1: Computer Fundamentals	2
Basics	2
Functional Components of a Computer	6
Booting	10
Unit 2: Software Concepts	12
Software Concepts	12
Language Processor	14
Utility Software	15
Application Software	16
Unit 3: Number System	20
Number System	20
Internal Storage encoding of characters	27
Microprocessor	28
Unit 4: Programming Methodology	31
Programming Methodology	31
Problem solving process	34
Flowcharts	38
Unit 5: Introduction to C++	42
Basics(C++)	42
Data types, Variables and Constants	53
Operators and Expressions	55
Unit 6: Programming in C++	61
Statements	61
Iterative Statements: Loops	70
Unit 7: User Defined Functions	76
Functions	76
Function Prototypes	78
Calling a function	79
Unit 8: Arrays and Structures	85
Structured data types: Arrays	85
User Defined data types: Structures	92

Unit 1

Computer Fundamentals

Objectives:

In this Chapter you will learn about:

- Computer Basics
- Functional Components of a Computer
- Booting
- Software Concepts
- Language Processor
- Utility Software
- Application Software



What is 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:

1. Input
2. Processing
3. Output
4. Storage

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

2. Process:

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

3. 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.

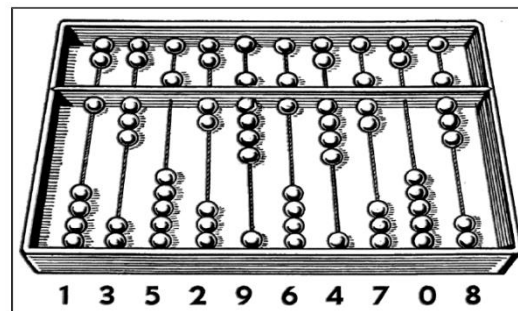
Evolution of Computers:

The growth of computer industry started with the need for performing fast calculations. The manual method of computing was slow and prone to errors. So

attempts were made to develop faster calculating devices. The journey that started from the first calculating device i.e. Abacus has led us today to extremely high speed calculating devices. Let us first have a look at some early calculating devices and then we will explore various generations of computer.

Abacus:

Abacus was discovered by the Mesopotamians in around 3000 BC. An abacus consisted of beads on movable rods divided into two parts. Addition and multiplication of numbers was done by using the place value of digits of the numbers and position of beads in an abacus.



Pascaline:

Blaise Pascal, a French mathematician invented an adding machine in 1642 that was made up of gears and was used for adding numbers quickly. This machine was also called Pascaline and was capable of addition and subtraction along with carry-transfer capability. It worked on clockwork mechanism principle. It consisted of various numbered toothed wheels having unique position values. The addition and subtraction operations was performed by controlled rotation of these wheels.



Leibnitz's Calculator:

In 1673 Gottfried Leibnitz, a German mathematician extended the capabilities of the adding machine invented by Pascal to perform multiplication and division as well. The multiplication was done through repeated addition of numbers using a stepped cylinder each with nine teeth of varying lengths.

Jacquard's Loom:

In order to make the cotton weaving process automatic, Joseph Jaquard devised punch cards and used them to control looms in 1801. The entire operation was under a program's control. Through this historic invention, the concept of storing and retrieving information started.

Difference engine and Analytical Engine:

Charles Babbage, an English mathematician developed a machine called Difference Engine in 1822 which could calculate various mathematical functions, do polynomial evaluation by finite difference and theoretically could also solve differential equations. Thereafter in 1833, he designed the Analytical Engine which later on proved to be the basis of modern computer. This machine could perform all the four arithmetic operations as well as comparison. It included the concept of central processor, memory storage and input-output devices. Even the stored information could be modified. Although the analytical engine was never built that time but Babbage established the basic principles on which today's modern computers work.

Both these great inventions earned him the title of “**Father of Modern Computers**”.

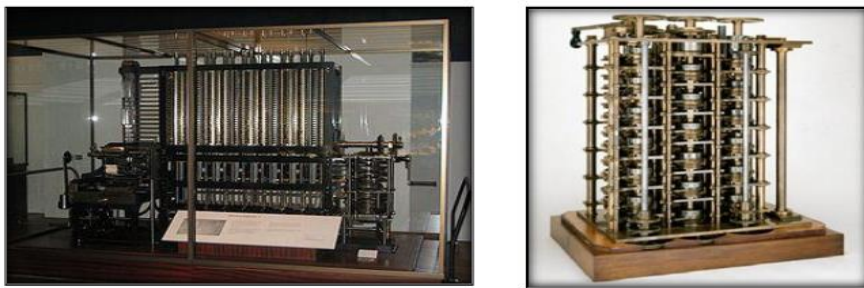


Fig: Difference Engine and Analytical engine

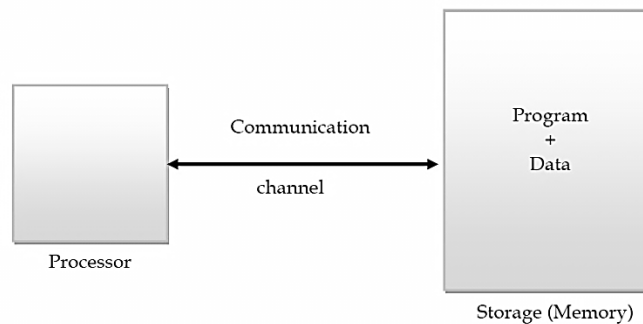
Mark 1:

In 1944 Prof Howard Aiken in collaboration with IBM constructed an electromechanical computer named Mark 1 which could multiply two 10 digit numbers in 5 seconds. This machine was based on the concept of Babbage's Analytical engine and was the first operational general purpose computer which could execute preprogrammed instructions automatically without any human intervention.

Basic Architecture Of The Von Neumann Computer:

In 1945, Dr. John Von Neumann proposed the concept of a stored program computer. As per this concept the program and data could be stored in the same memory unit. According to Von Neumann architecture, the processor executes instructions stored in the memory of the computer. Since there is only one communication channel, the processor at a time can either fetch data or an instruction. That means at one point of time either the data or an instruction can be picked (fetched) from the storage unit for execution by the processor. Hence execution takes place in sequential manner. This limitation of Von Neumann Computer is known as Von Neumann bottleneck. EDVAC (Electronic Discrete Variable Automatic Computer) was the first stored program computer developed in 1952. After the invention of first electronic computer ENIAC (Electronic

Numerical Integrator and Calculator) in 1946, the computer technology improved tremendously and at very fast pace.



Generation of Computers:

Growth in the computer industry is determined by the development in technology. Each phase/generation of computer development is characterized by one or more hardware/software developments that distinctly improved the performance of the computers of that generation. Based on various stages of development, computers can be divided into different generations.

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.

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. 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 known to be Micro Computer. High capacity hard disk 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.

Computer - Data and Information:

We all know what a computer is? It is an electronic device that processes the input according to the set of instructions provided to it and gives the desired output at a very fast rate. Computers are very versatile as they do lot of different tasks such as storing data, weather forecasting, booking airline, railway or movie tickets and even playing games.

Data:

It is the term used for raw facts and figures. For example, **102**, **-3**, **'Javeed'**, **'C'** are data. Definition of information should start from next line as given in the word file. In composed file it is starting from the same line immediately after the definition of data.

Information:

Data represented in useful and meaningful form is information. In simple words we can say that data is the raw material that is processed to give meaningful, ordered or structured information. For example **Javeed is 30 years old**. This is information about **Javeed** and conveys some meaning. This conversion of data to information is called data processing.

FUNCTIONAL COMPONENTS OF A COMPUTER:

Input Unit:

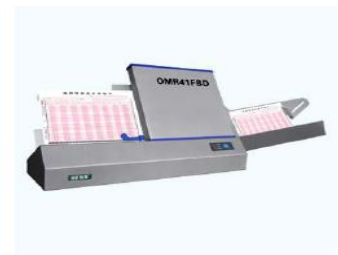
Those devices which help to enter data into computer system. e.g. Keyboard, Mouse, Touchscreen, Barcode Reader, Scanner, MICR, OMR etc.



Bar code Reader



MICR used in Bank



OMR

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.



Processor

Arithmetic and Logic Unit (ALU):

The ALU, as its name suggests performs mathematical calculations and takes logical decisions. Arithmetic calculations include addition, subtraction, multiplication and division. Logical decisions involve comparison of two data items to see which one is larger or smaller or equal.

Control Unit:

The Control unit coordinates and controls the data flow in and out of CPU and also controls all the operations of ALU, memory registers and also input/output units. It is also responsible for carrying out all the instructions stored in the program. It decodes the fetched instruction, interprets (understands) it and sends control signals to input/output devices until the required operation is done properly by ALU and memory.

CPU Registers:

A register is a temporary unit of memory in the CPU. These receive data/information and then this data/information is held in them as per the requirement. Registers can be of different sizes(16 bit , 32 bit , 64 bit and so on) and each register inside the CPU has a specific function like storing data, storing an instruction, storing address of a location in memory etc. The user registers can be used by an assembly language programmer for storing operands, intermediate results etc. Accumulator (ACC) is the main register in the ALU and contains one of the operands of an operation to be performed in the ALU.

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.

- Volatile memory(RAM):

RAM means Random Access Memory (Read/ Write Memory). It's known as Volatile Memory. 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.

- Non Volatile Memory (ROM):

ROM means Read Only Memory. 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:

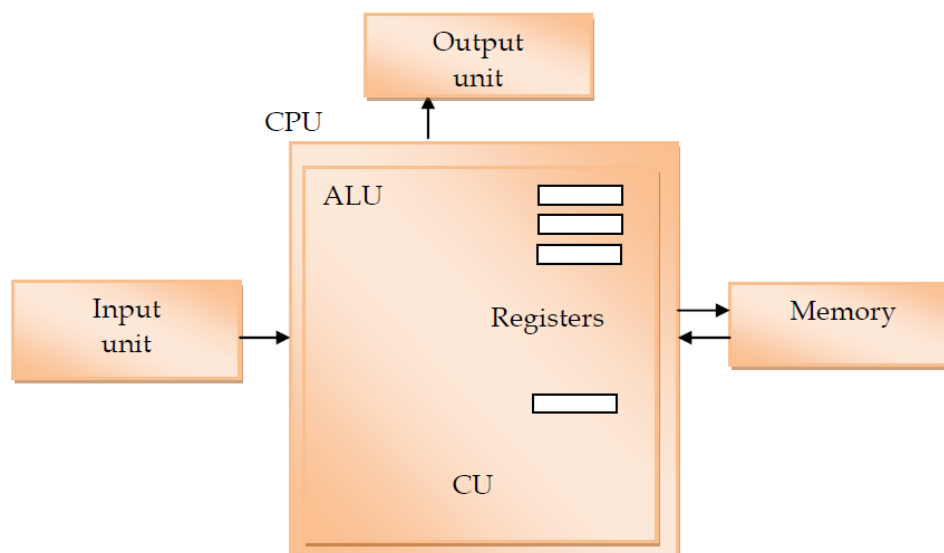
Cache memory is random access memory (RAM) that a computer microprocessor can access more quickly than it can access regular RAM. As the microprocessor processes data, it looks first in the cache memory and if it finds the data there (from a previous reading of data), it does not have to do the more 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
- E. Memory Cards
- F. External Hard Disk
- G. Blue Ray Disk

Output Unit:

Those devices which help to display the processed information. Eg. Monitor, Printer, Plotter, Projector



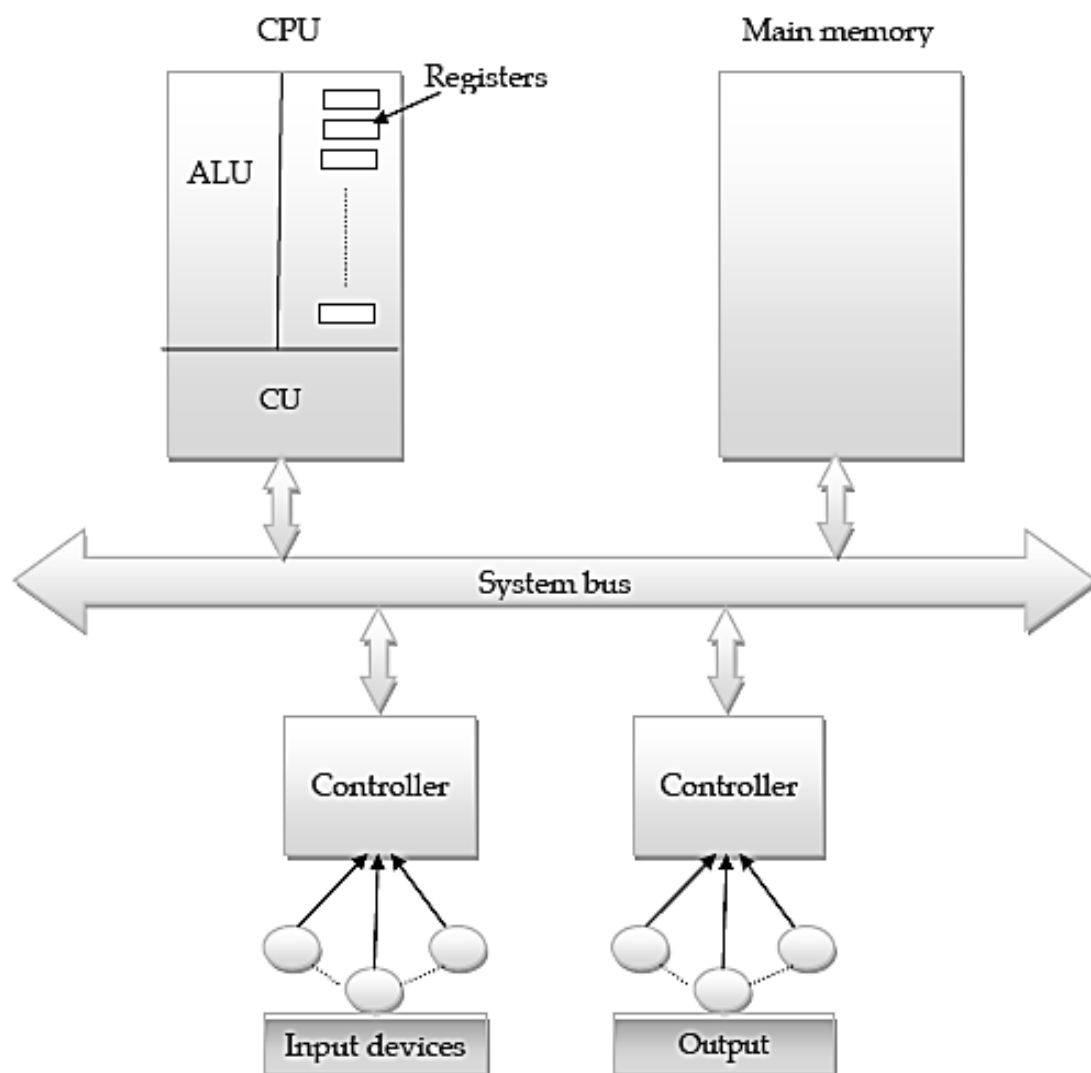
Functional units of Computer

Interconnection between Functional Components:

The interconnection between the functional components of a computer can be done in many ways. In microcomputers we generally see a Common Bus Architecture as shown in the figure below. As we have seen before that a computer consists of input unit that takes input, a CPU that processes the input and an output unit that produces output. All these devices communicate with each other through a common bus. A bus is a transmission path (set of conducting wires) over which data or information in the form of electric signals, is passed from one component to another in a computer. The bus can be of three types – **Address bus, Data bus and Control Bus.**

The address bus carries the address location of the data or instruction. The data bus

carries data from one component to another and the control bus carries the control signals. As shown in the figure below, the system bus is the common communication path that carries signals to/from CPU, main memory and input/output devices. The input/output devices communicate with the system bus through the controller circuit. This controller circuit helps to manage various input/output devices attached to the computer.



BOOTING:

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 booting.

Types of Booting:

There are two types of booting:

- **Cold Booting:**

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 booting.

- **Warm Booting:**

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 booting.

Unit 2

Software Concepts

Objectives:

In this Chapter you will learn about:

- Definition of Software
- Types of software
- Need for Operating System
- Functions of Operating System
- Concept of computer languages



SOFTWARE CONCEPTS

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:

- **Operating System Software:** DOS, Windows XP, Windows Vista, Unix/Linux, MAC/OS X etc.
- **Utility Software:** Windows Explorer (File/Folder Management), Compression Tool, Anti-Virus Utilities, Disk Defragmentation, Disk Clean, BackUp, WinZip, WinRAR etc...
- Language Processors Compiler, Interpreter and Assembler

2. Application software:

- **Package Software:** Ms. Office 2003, Ms. Office 2007, Macromedia (Dreamweaver, Flash, Freehand), Adobe (PageMaker, PhotoShop)
- **Tailored or Custom Software:** School Management system, Inventory Management System, Payroll system, financial system etc.

Operating System:

An Operating system is the most important system software. It is a set of programs that control and supervise the hardware of a computer and also provide services to application software, programmers and users. It manages all hardware and software, input, output and processing activities within the computer system, the flow of information to and from the processor, sets priorities for handling different tasks, and so on. Without operating system a computer cannot do anything useful. When a computer is switched on, the operating system is the first program that is loaded onto its memory. A user cannot communicate directly with the computer hardware, so the operating system acts as an interface between the user and the computer hardware. Some of the popular operating systems used in personal computers are DOS, Windows, Unix, Linux, Solaris, etc.

An operating system can be a Single User or a Multiuser operating system. A single user operating system allows only one user to work at any time but a multiuser operating system allows two or more users to use a powerful computer at the same time. For example Windows 7 is a single user operating system while Linux is a multiuser operating system.

Need for an Operating System:

Operating system provides a platform, on top of which, other programs, called application programs can run. As discussed before, it acts as an interface between the computer and the user. It is designed in such a manner that it operates, controls and executes various applications on the computer. It also allows the computer to manage its own resources such as memory, monitor, keyboard, printer etc.

Our choice of operating system, therefore, depends to a great extent on the CPU and the other attached devices and the applications we want to run. The operating system controls the various system hardware and software resources and allocates them to the users or programs as per their requirement.

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 run 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.

LANGUAGE PROCESSORS:

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

- **Compiler:**

It is translator which converts the HLL language into machine language in one go. A Source program in High Level Language get converted into Object Program in Machine Level Language.

- **Interpreter:**

It is a translator which converts the HLL language into machine language line by line. It takes one statement of HLL and converts it into machine code which is immediately executed. It eliminate the need of separate compilation/run. However, It is slow in processing as compare to compiler.

- **Assembler:**

The Assembler is used to translate the program written in Assembly language into machine code. The input of Assembler is a source program that contains assembly language instructions. The output generated by assembler is the object code or machine code understandable by the computer.

UTILITY SOFTWARES

A utility software is one which provides certain tasks that help in proper maintenance of the computer. The job of utility programs is to keep the computer system running smoothly. Nowadays many utility softwares are part of the operating system itself. Even if there is no utility software on your computer, the computer works but with the right kind of utility software loaded, the computer becomes more reliable and even its processing speed increases. Some of the commonly use utility softwares are antivirus, Disk defragmenter, backup, compression etc.

Antivirus:

An antivirus is utility software which detects and removes computer viruses. If the software is not able to remove the virus, it is neutralized. The antivirus keeps a watch on the functioning of the computer system. If a virus is found it may alert the user, flag the infected program or kill the virus. Some of the common types of viruses are:

- **Boot Sector Virus:**

A boot sector virus displaces the boot record and copies itself to the boot sector i.e. where the program to boot the machine is stored. So first the virus is loaded on to the main memory and then the operating system. Whenever a new disk is inserted the virus copies itself to the new disk. The antivirus overwrites the correct boot record on the infected boot sector and also cleans the bad sectors.

- **File Virus:**

A file virus generally attacks executable files. They can attach to various locations of the original file, replace code, fill in open spaces in the code, or create companion files to work with an executable file. Most of the file viruses are memory resident and wait in the memory until the user runs another program. While another program is running, the virus replicates.

- **Macro Virus:**

This virus infects an important file called normal.dot of MS Word. As soon as the application is opened the virus gets activated. It damages the formatting of documents and even may not allow editing or saving of documents.

- **Trojan Horse:**

It is a code generally hidden in games or spreadsheets. Since they are hidden, the program seems to function as the user wants but actually it is destroying the program. A Trojan horse does not require a host program to embed itself. It is a complete program. Its main objective is to cause harm to the data. They can create bad sectors on the disk, destroy file allocation tables and cause the system to hang.

- **Worm:**

Worm is a program capable of replicating itself on a computer network. A worm also does not require a host as it is a self-contained program. They generally travel from one computer to another across communication links on a network. They generally disrupt routine services.

Disk Defragmenter:

The memory is used in small chunks randomly. Sometimes when a memory chunk of appropriate size is not available, the operating system breaks or fragments the files resulting in slower access to files. A disk defragmenter scans the hard disk for fragmented files and brings all the fragments together.

Backup Utility:

This utility is used to create the copy of the complete or partial data stored in a disk or CD on any other disk. In case the hard disk crashes or some other system failure occurs, the files can be restored using backup software.

Compression Utility:

This utility is used to compress large files. Compression is useful because it helps reduce resources usage and the file transmission on the network becomes easier.

Disk Cleaner:

This utility scans for file that have not been accessed/used since long. Such files might be occupying huge amount of memory space. In that case the Disk Cleaner utility prompts the user to delete such files so as to create more space on the disk. If the files are important, the user might take a backup before deleting them.

File Management Tools:

This utility helps the user in storing, indexing, searching and sorting files and folders on the system. The most commonly used tool is the Windows Explorer and Google Desktop.

APPLICATION SOFTWARE

An application software is bought by the user to perform specific applications or tasks, say for example making a document or making a presentation or handling inventory or managing the employee database. An application software can be of two types – General Purpose Application Software and Customized Application software.

- **General Purpose Application Software**

Some of the application software is made for the common users for day to day applications and uses. These are also referred as Office Tools. The users may use them in the manner they want. Some of the popular types of general purpose application software are discussed below:

- ✓ **Word Processor:**

Word processor is a general purpose application software used to create documents. It allows us to create , edit and format documents. We can use different types of fonts of various sizes; underline or make bold a certain part of the text. We can add clipart and other graphics into the document. Popular examples of Word processing software are Writer (Open Office) and Microsoft Word.

We use word processing software for various uses like writing a simple document to designing special art effects. Since we can attach images and different shapes, can use different colors, even a poster can be designed using word processing software. Features like Mail Merge, Macro has further enhanced the word processing software and made it very useful.

- ✓ **Presentation Tools:**

Presentation tools is a general purpose application software that lets us create presentations on any topic. We can not only create a presentation and add slides into that but also can use different types of background, fonts, animations, audio, video, etc. We can add clipart and other graphics into our document. Even audio video files can be added on to the presentations. Popular examples of Presentation tools software are Impress (open office) and Microsoft Power Point.

- ✓ **Spreadsheet Packages:**

Spreadsheet is a general purpose application software that lets us create and store data in tabular form. Both text and numerical values can be entered in that tables known as a spreadsheet. We can not only create a document and add data into that but also can create different types of charts and graphs based upon the numerical data stored in that page. All common mathematical and statistical formulae can be used on the numeric data. Popular examples of Spreadsheet software are Calc (Open Office) and Microsoft Excel.

✓ **Database Management System:**

Database Management System is general purpose application software that lets us create computer programs that control the creation, maintenance, and the use of database for an organization and its end users. We can not only store data but can also manage data in a database. We can also import and export the data to many formats including Excel, Outlook, ASCII, dBase, FoxPro, Oracle, SQL Server, ODBC, etc. Popular examples of Database Management System are Base (Open Office) and Microsoft Access.

• **Customized Software**

Customized Software is one which is tailor made as per the user's requirement. Such type of software is customer specific. It is made keeping in mind the individual needs of the user and so are also referred as Domain Specific Tools. Such software cannot be installed and used by any other user/customer since the requirements may differ. Some examples of customized software are discussed below:

✓ **Inventory Management System:**

Inventory Management System is generally used in departmental stores or other organizations to keep the record of the stock of all the physical resources. For Example, in a Computer store, it keeps record of the number of computers, printers, printing sheet, printer cartridge available. It also helps to place purchase orders, bills, invoices etc. Various reports as to position of stock, sales made in a particular period, profit earned etc. can be generated.

✓ **School Management System:**

School Management System (sometimes called a School Information System or SIS) is a system that manages all of a school's data in a single, integrated application. Having all of the information in a single system allows schools to more easily connect data together. For example, when viewing a student's record, the user can follow a link to the student's class, and from there a link to the student's teacher, and from there a link to the teacher's other classes, and so on.

✓ **Payroll System:**

Payroll Management System software is used by all modern organizations to keep track of employees of the organization who receives wages or salary. All different payment amounts are calculated by the payroll software and the record is maintained. The software keeps track of personal records of employees viz. name, address, date of birth, qualification, date of joining etc. It also keeps track of professional record viz. allowances, perks, income tax, insurance etc. Different reports, pay slips etc can be generated through this software.

✓ **Financial System:**

Financial accounting System is used to prepare accounting information, maintain different accounts ledger, and account books. It also helps an organization to make budget.

✓ **Hotel Management:**

Hotel management software refers to management techniques used in the hotel sector. These can include hotel administration, accounts, billing, marketing, housekeeping, front office or front desk, food and beverage management, catering and maintenance. Even advance bookings can be made through this software. Customers can have a look at the hotel and the rooms before making bookings. At any point of time the room availability, tariff for each type of room and even booking status can be checked.

✓ **Reservation System:**

Reservation System is software used to book (reserve) air flights, railway seats, movie tickets, tables in a restaurant, etc. In the case of a booking system, the inputs are booking requests. The processing involves checking if bookings are possible, and if so making the bookings. The outputs are booking confirmations/rejections.

Unit 3

Number System

Objectives:

In this Chapter you will learn about:

- Number Systems
- Converting a number from one Base to another
- Internal storage encoding of characters
- Microprocessor



NUMBER SYSTEM

In order to work with data, the data must be represented inside the computer. Digital computers represent data by means of an easily identified symbol called a digit.

Each number system has a base also called a Radix. A decimal number system is a system of base 10; binary is a system of base 2; octal is a system of base 8; and hexadecimal is a system of base 16.

Number System	Base	Symbols used
Binary	2	0,1
Octal	8	0,1,2,3,4,5,6,7
Decimal	10	0,1,2,3,4,5,6,7,8,9
Hexadecimal	16	0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F where A = 10; B = 11; C = 12; D = 13; E = 14; F = 15

Converting a number from one Base to another:

Binary to Decimal:

Method to convert Binary to Decimal:

1. Start at the rightmost bit.
2. Take that bit and multiply by 2^n where n is the current position beginning at 0 and increasing by 1 each time. This represents a power of two.
3. Sum each terms of product until all bits have been used.

Example

Convert the Binary number 101011 to its Decimal equivalent.

$$1 * 2^5 + 0 * 2^4 + 1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 1 * 2^0$$

$$32 + 0 + 8 + 0 + 2 + 1 = (43)_{10}$$

Example

Convert the Binary number 1001 to its Decimal equivalent.

$$1 * 2^3 + 0 * 2^2 + 0 * 2^1 + 1 * 2^0$$

$$8 + 0 + 0 + 1 = (9)_{10}$$

Binary fraction to decimal:

Example

Convert $(11011.101)_2$ to decimal

$$2^4 \ 2^3 \ 2^2 \ 2^1 \ . \ 2^0 \ 2^{-1} \ 2^{-2} \ 2^{-3}$$

$$1 \ 1 \ 0 \ 1 \ 1 \ 1 \ 0 \ 1$$

$$= (1 \times 2^4) + (1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) + (1 \times 2^{-1}) + (0 \times 2^{-2}) + (1 \times 2^{-3})$$

$$= 16 + 8 + 0 + 2 + 1 + 0.5 + 0 + 0.125$$

$$= (27.625)_{10}$$

Decimal to Binary:

Method to convert a Decimal number into its Binary equivalent:

1. Divide the decimal number by 2.
2. Take the remainder and record it on the side.
3. Divide the quotient by 2.
4. REPEAT UNTIL the decimal number cannot be divided further.
5. Record the remainders in reverse order and you get the resultant binary number.

Example

Convert the Decimal number 125 into its Binary equivalent.

Quotient	Reminder
$125 / 2 = 62$	1
$62 / 2 = 31$	0
$31 / 2 = 15$	1
$15 / 2 = 7$	1
$7 / 2 = 3$	1
$3 / 2 = 1$	1
$1 / 2 = 0$	1

Answer: $(1111101)_2$

Converting Decimal fraction to Binary:

Convert $(0.75)_{10}$ to binary

Multiply the given fraction by 2. Keep the integer in the product as it is and multiply the new fraction in the product by 2. Continue the process till the required number of decimal places or till you get zero in the fraction part. Record the integers in the products from top to bottom.

Given fraction 0.75

Multiply 0.75 by 2 1.50

Multiply 0.50 by 2 1.00

Reading the integers from top to bottom 0.75 in decimal number system is 0.11 in binary number system.

Example

Convert $(105.15)_{10}$ to binary

Let us convert 105 first.

$(105)_{10} = (1101001)_2$

Let us convert $(0.15)_{10}$

Multiply 0.15 by 2 0.30

Multiply 0.30 by 2 0.60

Multiply 0.60 by 2 1.20

Multiply 0.20 by 2 0.40

Multiply 0.40 by 2 0.80

Multiply 0.80 by 2 1.60

Reading the integers from top to bottom $(0.15)_{10} = (0.001001)_2$

Final result $(105.15)_{10} = (1101001.001001)_2$

Decimal to Octal:

The method to convert a decimal number into its octal equivalent:

1. Divide the decimal number by 8.
2. Take the remainder and record it on the side.
3. Divide the quotient by 8.
4. REPEAT UNTIL the decimal number cannot be divided further.
5. Record the remainders in reverse order and you get the resultant binary

Example

Convert the Decimal number 125 into its Octal equivalent.

$125 / 8 = 15$ 5

$15 / 8 = 1$ 7

$$1/8 = 0 \quad 1$$

Answer: $(175)_8$

Converting Decimal fraction to Octal:

Convert $(0.75)_{10}$ to Octal

Multiply the given fraction by 8. Keep the integer in the product as it is and multiply the new fraction in the product by 8. Continue the process and read the integers in the products from top to bottom.

Given fraction 0.75

Multiply 0.75 by 8 6.00

Reading the integers from top to bottom 0.75 in decimal number system is 0.6 in octal number system.

Octal to Decimal:

Method to convert Octal to Decimal:

1. Start at the rightmost bit.
2. Take that bit and multiply by 8^n where n is the current position beginning at 0 and increasing by 1 each time. This represents the power of 8.
3. Sum each of the product terms until all bits have been used.

Example

Convert the Octal number 321 to its Decimal equivalent.

$$3 * 8^2 + 2 * 8^1 + 1 * 8^0$$

$$192 + 16 + 1 = (209)_{10}$$

Octal fraction to decimal:

Example

Convert $(23.25)_8$ to decimal

$$\begin{array}{cccc} 8^1 & 8^0 & . & 8^{-1} & 8^{-2} \\ 2 & 3 & & 2 & 5 \end{array}$$

$$= (2 \times 8^1) + (3 \times 8^0) + (2 \times 8^{-1}) + (5 \times 8^{-2})$$

$$= 16 + 3 + 0.25 + 0.07812$$

$$= 19.32812$$

$$= (19.32812)_{10}$$

Decimal to Hexadecimal:

Method to convert a Decimal number into its Hexadecimal equivalent:

1. Divide the decimal number by 16.
2. Take the remainder and record it on the side.
3. REPEAT UNTIL the decimal number cannot be divided further.
4. Record the remainders in reverse order and you get the equivalent hexadecimal number.

Example

Convert the Decimal number 300 into its hexadecimal equivalent.

$$300 / 16 = 18 \quad 12\text{-(C)}$$

$$18 / 16 = 1 \quad 2$$

$$1 / 16 = 0 \quad 1$$

Answer: $(12C)_{16}$

Converting Decimal fraction to Hexadecimal:

Example

Convert $(0.75)_{10}$ to hexadecimal

Multiply the given fraction by 16. Keep the integer in the product as it is and multiply the new fraction in the product by 16. Continue the process and read the integers in the products from top to bottom.

Given fraction 0.75

Multiply 0.75 by 16 $12.00 - C$

Reading the integers from top to bottom 0.75 in decimal number system is $0C$ in Hexadecimal number system.

Hexadecimal to Decimal:

Method to convert Hexadecimal to Decimal:

1. Start at the rightmost bit.
2. Take that bit and multiply by 16^n where n is the current position beginning at 0 and increasing by 1 each time. This represents a power of 16.
3. Sum each terms of product until all bits have been used.

Example

Convert the Hexadecimal number AB to its Decimal equivalent.

$$= A * 16^1 + B * 16^0$$

$$= 10 * 16^1 + 11 * 16^0$$

$$= 160 + 11 = (171)_{10}$$

Hexadecimal fraction to decimal:

Example

Convert $(1E.8C)_{16}$ to decimal

$$16^1 \quad 16^0 \quad . \quad 16^{-1} \quad 16^{-2}$$

$$1 \quad E \quad 8 \quad C$$

$$= (1 \times 16^1) + (14 \times 16^0) + (8 \times 16^{-1}) + (12 \times 16^{-2})$$

$$= 16 + 14 + 0.5 + 0.04688$$

$$= (30.54688)_{10}$$

Binary to Hexadecimal:

The hexadecimal number system uses the digits 0 to 9 and A, B, C, D, E, F.

Method to convert a Binary number to its Hexadecimal equivalent is:

We take a binary number in groups of 4 and use the appropriate hexadecimal digit in its place. We begin at the rightmost 4 bits. If we are not able to form a group of four, insert 0s to the left until we get all groups of 4 bits each. Write the hexadecimal equivalent of each group. Repeat the steps until all groups have been converted.

Example

Convert the binary number 1000101 to its Hexadecimal equivalent.

0100 0101 Note that we needed to insert a 0 to the left of 100.

Answer: (45)₁₆

In case of a fractional binary number form groups of four bits on each side of decimal point. Then replace each group by its corresponding hexadecimal number.

Example

Convert (11100.1010)₂ to hexadecimal equivalent.

0001 1100 . 1010

1 C . A

Answer : (1C.A)₁₆

Hexadecimal to Binary:

Method to convert a Hexadecimal number to its Binary equivalent is:

Convert each digit of Hexadecimal Number to its binary equivalent and write them in 4 bits. Then, combine each 4 bit binary number and that is the resulting answer.

Example

Convert the Hexadecimal number (10AF)₁₆ to its Binary equivalent.

1 0 A F

0001 | 0000 | 1010 | 1111

Answer: (0001000010101111)₂

Example

Convert the Hexadecimal number (A2F)₁₆ to its Binary equivalent.

A 2 F

1010 | 0010 | 1111

Answer: (1010 0010 1111)₂

Binary to Octal and Octal to Binary:

To convert Binary to Octal, as the octal system is a power of two (2^3), we can take the bits into groups of 3 and represent each group as an octal digit. The steps are the same for the binary to hexadecimal conversions except we are dealing with the octal base now.

To convert from octal to binary, we simply represent each octal digit in its three bit binary form.

Example

Convert the Octal number $(742)_8$ to its Binary equivalent.

7 | 4 | 2
111 | 100 | 010

Answer: $(111100010)_2$

Hexadecimal to Octal and Octal to Hexadecimal:

To convert Hexadecimal to Octal, Convert each digit of Hexadecimal Number to its binary equivalent and write them in 4 bits. Then, combine each 3 bit binary number and that is converted into octal.

Example

Convert the Hexadecimal number $(A42)_{16}$ to its Octal equivalent.

A | 4 | 2
1010 | 0100 | 0010
101 | 001 | 000 | 010

Answer: $(5102)_8$

To convert Octal to hexadecimal, convert each digit of Octal Number to its binary equivalent and write them in 3 bits. Then, combine each 4 bit binary number and that is converted into hexadecimal.

Example

Convert the Octal number $(762)_8$ to its hexadecimal equivalent.

7 | 6 | 2
101 | 110 | 010
0001 | 0111 | 0010

Answer: $(172)_{16}$

The following table summarizes the number representation in decimal, binary, octal and hexadecimal number system:

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

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.

MICROPROCESSOR

We studied that the Central processing unit processes data inside the computer. It interprets all the instructions given to it and carries out these instructions. A microprocessor is a Central Processing Unit (CPU) on a single chip. It is a multipurpose programmable device constructed using Metal Oxide Semiconductor (MOS) technology. In 1971 Intel Corporation fabricated the first microprocessor-4004. It could do only add and subtract operations and that too it could process only 4 bits at a time. But Intel 4004 powered one of the first portable electronic calculators and since then the microprocessor technology has come a long way. Today we have processors with which can process upto 128 bits at a time at the speed of billion instructions per second. Let us now have a look at the characteristics of microprocessors:

- **Instruction Set:**

It is the set of instructions that the microprocessor executes

- **Word Length:**

The number of bits processed in a single instruction is called word length or word size. The word size is directly proportional to the processing power of the CPU. During the processing, the internal general purpose registers hold data. So if internal registers can hold data upto 8 bits, the word length is 8 bits. If it can process 16 bits at a time, then the internal registers can hold upto 16 bits at a time and so on. Hence a 32 bit processor is about 4 times faster than an 8 bit processor. Examples of word lengths are 16 bit, 32 bit, 64 bit. The terms 16-bit CPU, 32-bit CPU, 64-bit CPU are used very often while talking about CPUs. Now we know that these terms mean the maximum number of bits a given CPU can handle at a time.

- **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: It indicates the width of the registers; a special high-speed storage area within the CPU. A 32-bit microprocessor can process data and memory addresses that are represented by 32 bits. Eg. Pentium dual core, core 2 duo.

Classification of Microprocessors:

Apart from the width of data (word length) that the microprocessors can process at a time, the classification is also based on the architecture i.e. Instruction Set of the microprocessor. While studying about CPUs, we come across two abbreviations CISC and RISC.

- **RISC:**

It stands for Reduced Instruction Set Computer. It is a type of microprocessor architecture that uses a small set of instructions of uniform length. These are simple but primitive instructions which execute in one clock cycle. For this reason, RISC chips are less complex and also less expensive to produce. The instructions are of uniform length which interface with about 32-36 registers. The program size in case of RISC architecture is more but more memory cycles are needed to access data. To reduce the number of memory cycles, RISC keeps the necessary data in the processor itself. The drawback of RISC design is that the computer must combine or repeat operations to complete a large program consisting of many processing operations. Since instructions are simple, RISC processors are relatively simple to design. Examples of RISC processor is SPARC, POWER PC etc.

- **CISC:**

It stands for Complex Instruction Set Computer. A CISC chip such as Intel Pentium provides programmers with hundreds of instructions of variable sizes, and the processing circuitry includes many special purpose circuits that carry out these instructions at high speeds. These instructions interface with memory in multiple mechanisms with complex addressing modes. In this case the program size is reduced and hence lesser number of memory cycles are required to execute the instruction. So fewer general purpose registers(8-12) are present in CISC processors. Also less number of memory cycles result in faster execution of the program.

- **EPIC:**

It stands for Explicitly Parallel Instruction Computing. It is a computer architecture that combines the best feature of both RISC and CISC. It does not use instructions of any fixed length but rather aims at parallel processing of instructions. It uses a bundle of complex instructions that in addition to basic instruction also contain information on how to run the instruction in parallel with other instructions. This greatly increases the efficiency of an EPIC processor. IA-64 (Intel Architecture-64) is Intel's first 64 bit processor based on EPIC.

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

Unit 4

Programing Methodology

Objectives:

In this Chapter you will learn about:

- Computer Program
- Algorithm/Flowchart
- Documentation/Program Maintenance



Introduction

Learning to write computer program is very much like learning any skill. First, we should understand the problems well and then try to solve it in a logical manner. For example: We have read many books available in the market for describing the car driving methods. However, we can learn driving once we actually get into the car and start driving it. The same logic is applied in computer programming also. Computer programming is the process of writing, testing, troubleshooting, debugging and maintaining of a computer program.

An effective program is that which gives result of all different inputs, including wrong input also. While creating program, we need to follow certain systematic approach. This systematic approach comprises two steps/things, viz., program structure and program representation. The program structure is implemented by using top-down or bottom-up approach and is known as „popular approach“, while the program representation plays an important role in making the program more readable and understandable.

What is a Good Program?

A Good Program means that it should produce correct and faster results, taking into account all the memory constraints. While making good program, we need to follow certain guidelines of programming language for creating a successful program. The following is the list of good programming habits that most people agree.

Clarity and Simplicity of Expression

Expressions are used to implement a particular task. It is a combination of Operators, Operands and Constants. Any expression used in the program should be understood by the user. The followings are some of the points to be kept in mind while using expressions in a program.

- (i) Use library functions to make programs more powerful

Example

To find output = x6

Output = $X * X * X * X * X * X$

We can use output = power (X, 6)

- (ii) Follow simplicity to maintain the clarity of expression

Example

$X = A + B - U + VY$

$A - B \quad X + Y$

Then, we can write

$X1 = (A + B) / (A - B)$

$X2 = (U + V * Y) / (X + Y)$

$X = X1 - X2$

- (iii) Avoid program tricks usage, whose meaning is difficult to understand by the user.

Use of proper names for identifiers

Identifiers are user defined names. They are used to name things. A name is associated with a function or data object (constants and variables) and used to refer to that function or data object. Identifiers are made up of letters (A-Z, a-z), digits (0-9), and the underscore character (_). They, however, must begin with a letter or underscore and not with a digit.

- (i) Give meaningful name for variable (data – object) and function.

Example

To calculate Area of a Square

We use the variable names are Area and Side

$\text{Area} = \text{Side} * \text{Side}$.

- (ii) Use proper names for constants.

Example

$\pi = 3.14$

Give Pi = 3.14

- (iii) Do not use same name like custom, customer or account, accountant.
- (iv) Do not use one letter identifiers.

Comments

A comment is a programming language construct, which is used to embed programmer-readable annotations in the source code of a computer program. Those annotations are potentially significant to programmers but typically ignorable to

compilers and interpreters. Comments are usually added with the purpose of making the source code easy to understand. Hence, add comments to your code in simple English language that describes the function of the code and the reason for your decision to do it in a particular way as well. Comments are implemented in C++ language by using “/*” or “//” before any comment statement.

Example

"Write a program to print all numbers from 1 to 100 using while loop in C++"

```
#include<iostream.h>
main()
{
a = 1
while (a<100):
cout<< a
a = a+1
}
```



// While statement

Comment line

Indentation

Leading white space (spaces and tabs) at the beginning of each statement, which is used to determine the group of statement, is known as “indentation”.

Example

```
If (A > B)
print “A is Big” // Block1
else
print “B is Big” // Block2
```

In the above example, if statements are a type of code block. If the “if” expression evaluates to true, then Block1 is executed, otherwise, it executes Block2. Obviously, blocks can have multiple lines. As long as they are all indented with the same amount of spaces, they constitute one block.

Characteristics of good programming

Every computer needs proper instruction set (programs) to perform the required/assigned task. The quality of the program depends upon the instructions given to it. However, it is required to feed/provide the proper and correct instructions to the computer in order to yield/provide a correct and desired output. Hence, a program should be developed to ensure proper functionality of the computer and also should be easy to understand. A computer program should have some important characteristics, which are as follows:

Flexibility

A program should be flexible enough to handle most of the changes without having to rewrite the entire program. A flexible program is used to serve many purposes. For example, CAD (Computer Aided Design) software is used for different purposes such as; engineering drafting, printing circuit board layout and design, architectural design, technical drawing, industrial art, etc. Most of the programs are being developed for certain period and they need updation during the course of time.

User Friendly

A program that can be easily understood by a beginner is called „user friendly“. It must interact with user through understandable messages. In addition, the proper message for the user to input data and to display the result, besides making the program easily understandable and modifiable.

Portability

Portability refers to the ability of an application to run on different platforms (operating systems) with or without minimal changes. Since the change of platform is a common phenomenon nowadays, due to the developments in hardware and the software, portability has to be taken care of it. In case, a program is developed for a particular platform, it would become obsolete after a certain period of time. At the same time, if a program that is developed does have the ability to work on different platforms, it makes software more useable. High language programs are often more portable than assembly language programs.

Reliability

It is the ability of a program to do its intended function accurately even if there are even small changes in the computer system. Moreover, the program must be able to handle unexpected situation like wrong input or no input. The programs, which save such ability are known as „reliable“. For example, if the user does/gives wrong information to input, it should display a proper error message.

Self-Documenting Code

The source code, which uses suitable name for the identifiers (variables and methods), is called self-documenting code. Also, giving proper name for variables and methods would tell the reader of your code clearly -what is it doing? Hence, a good program must have a self-documenting code.

Problem solving process

The problem solving process starts with the problem specifications and ends with a

concrete (and correct) program. Programming means a problem solving activity, which consists of four steps. They are;

- (i) Understanding the problem;
- (ii) Devising a plan;
- (iii) Executing the plan; and
- (iii) Evaluation

Understanding the problem

The first step is to understand the problem well. It may be very difficult to understand the problem but it is crucial. In general, one must find out the output from the given data (input data) and assess the relationship between input and output data. It is also important to verify whether the given information is sufficient to solve the problem or not.

Devising a plan

It means drawing an action plan to solve the problem, once understood. A plan is devised from data processing to the result according to the relationship that links both of them. If the problem is trivial, this step will not require much thinking.

Executing the plan

Once the plan is defined, it should follow the plan of action completely and each element of the plan should be checked as it is applied. In the course of execution, if any part of the plan is found to be unsatisfactory, the plan should be revised.

Evaluation

Finally, the result should be examined in order to make sure that it is valid and that the problem has been solved completely.

Problem solving methodology

As we all know, there are many methods/approaches available to solve a particular problem. However, the efficient way is to adopt a systematic method of problem solving. The use of systematic method of problem solving is crucial when we use a computer to solve a problem. We introduce here a seven steps problem solving method, which is closely related to the software life cycle (the various stages in the life of a program), that can be adapted by each person to solve the problem in their own style. They are given as under:

1. Problem Definition
2. Problem Analysis
3. Design the problem
4. Coding
5. Program Testing and Debugging

6. Documentation

7. Program Maintenance

Problem Definition/Specification (Theme)

Computer programs are written to solve problems posed by humankind. Prior to writing a program, one has to understand a description of the problem to solve. This description may be very precise or vague, but nevertheless, it is necessary/present. For instance, if you want to write a program to “Find the average of five numbers”, you should ask yourself:

“What does average mean exactly?”

“How to calculate average value?”

Posing such questions compels you to define the problem very precisely. Once you are sure of what the problem entails, you must write down a list of specifications. Specifications are precise definitions of what the program must do. It must include the following at least:

- **Input:** what data must be included as input and in which form?
- **Output:** what data must the program produce and in which form? (in order to solve the problem)

Note: At the end of the problem definition step, you should have a list of specifications.

Problem Analysis

In this step, the problem has to be fragmented into smaller and manageable parts. The original problem has to be analyzed and divided into a number of sub-problems as these sub-problems are easier to solve and their solutions would become the components of the final program. Each sub-problem is divided into further smaller ones and this fragmentation has to be continued to achieve simple solutions. The use of modular programming is to get proper solution.

Modular Programming: Modular Programming is the act of designing and writing programs as functions (a large program is divided into the small individual components) that each one performs, a single well-defined function, which has minimal interaction between the sub-programs. It means that the content of each function is cohesive and there is low coupling between them. There are two methods available for modular programming. They are: top-down design and bottom-up design.

Top-Down design: The principles of top-down design dictate that a program should be divided into a main module and its related module. Each module should

also be divided into sub modules according to software engineering and programming style. The division continues till the module consists only of an elementary process that is intrinsically understood and cannot be further sub-divided.

Bottom-up design: Bottom-up design is just the opposite of top-down design. It refers to a style of programming, in which, an application is constructed with existing primitives of the programming language and then gradually more and more complicated features are added till applications are written. In other words, initiating the design with simple modules and then build them into more complex structures ending at the top is bottom-up design.

Designing the problem

Designing the problem can be expressed in the form of

- Algorithm
- Flowchart

Algorithm: An algorithm is a set of instructions that describe a method for solving a problem. It is normally given in mix of computer code and English language. This is often called “pseudo-code”. These algorithms are simply written in English like statements or in a Pseudo-code or in a graphical form called Flow charts.

Example (Algorithm in Simple English Statements)

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.

Example (Algorithm Pseudo-code)

Write an algorithm to find area of a rectangle.

Step 1: Start

Step 2: $L \leftarrow 6$, $B \leftarrow 5$

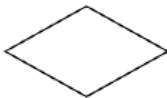
Step 3: $Area \leftarrow L \text{ multiplication } B$

Step 4: Print Area

Step 5: Stop

Flowchart: The algorithm is represented in the form of a diagram with action boxes linked by lines showing the order in which they are executed. This is known as “the flow of control”. It is the diagrammatic representation of an algorithm.

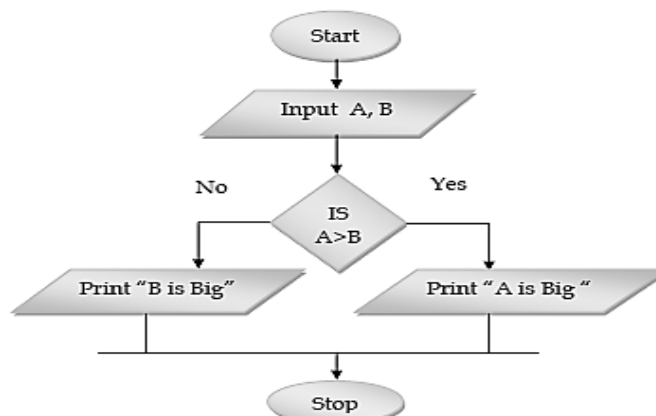
Some of the standard symbols along with respective function(s) that are used for making flowchart are as follows:

Symbols	Functions
1. 	Start/stop
2. 	Input/output
3. 	Processing
4. 	Decision Box
5. 	Flow of control
6. 	Connector

Example

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

Solution:



Coding

The process of translating the algorithm into syntax of a given language is known as “Coding”. Since algorithm cannot be executed directly by the computer, it has to be translated into a programming language.

Program Testing and Debugging

Program Testing means running the program, executing all its instructions/functions and testing the logic by entering sample data in order to check the output. Debugging is the process of finding and correcting the errors in the program code.

Type of errors: There are three types of errors generally occur during compilation and running a program. They are (i) Syntax error; (ii) Logical error; and (iii) Runtime error.

Syntax error: Every programming language has its own rules and regulations (syntax). If we overcome the particular language rules and regulations, the syntax error will appear (i.e. an error of language resulting from code that does not conform to the syntax of the programming language). It can be recognized during compilation time.

Example

```
include<iostream.h>
main
{
int a = 1;
while (a<100)           // While statement
cout<< a;
a = a+1
}
```

In the above statement, the 1st line is not correct, the correct form is “#include<iostream.h>”. in the second line “()” are missing, the correct form is “main()”. This will flash a syntax error.

Logical error: Programmer makes errors while writing program that is called “logical error”. It is an error in a program's source code that results in incorrect or unexpected result. It is a type of runtime error that may simply produce the wrong output or may cause a program to crash while running. The logical error might only be noticed during runtime, because it is often hidden in the source code and are typically harder to find and debug.

```
#include<iostream.h>
main()
{
int a = 10;
while (a<100)           // While statement
cout<< a;
a = a+1
}
```

In the above program “a” is initialized by 10, if you want to print values from 1 to 100, the above program will print values from 10 to 100, this is called logical error. To remove this logical error “a” must be initialized with 1. This type of error is very difficult to remove when a program is large.

Runtime error: A runtime error is an error that causes abnormal termination of program during running time. In general, the dividend is not a constant but might be a number typed by you at runtime. In this case, division by zero is illogical. Computers check for a "division by zero" error during program execution, so that you can get a "division by zero" error message at runtime, which will stop your program abnormally.

This type of error is called runtime error.

Example

(a) A=10

B=0

print A/B

(b) During running time, if we try to open a file that does not exist in the hard disk, then it will create runtime error.

Documentation

The documentation includes the problem definition, design documents, a description of the test perform, a history of the program development and its different versions and a user’s manual. Such a manual is designed for a naive user and illustrates the preparation of input data, running the program and obtaining & interpreting the results.

Program maintenance

It is not directly part of the original implementation process, but needs special emphasis. All activities that occur after a program operation are part of the program maintenance. Many large programs have long life span that often exceed

the lifetime of the hardware they run on. Usually, the expenditure for the program maintenance will be more than the developmental cost of the program. The program maintenance includes the following:

- Finding and eliminating previously undetected program errors;
- Modifying the current program, often to improve its performance, or to adapt to new laws or government regulations, or to adapt to a new hardware, or to a new operating system;
- Adding new features or a better user interface, or new capabilities to the program; and
- Updating the documentation.

Maintenance is an important part of the life cycle of a program. It is also important as far as documentation is concerned, since any change pertaining to a program will require updating of internal as well as external documentation. Maintenance documentation will include results of the program development steps, design documents, program code and test information.

Unit 5

Introduction to C++

Introduction:

This chapter describes the fundamental characteristics of the object oriented C++ programming language. In addition, you will be introduced to the steps necessary for creating a fully functional C++ program. The examples provided will help you retrace these steps and also demonstrate the basic structure of a C++ program.

Objectives:

In this Chapter you will learn about:

- Fundamental characteristics of the object oriented C++ programming language.
- Steps necessary for creating a fully functional C++ program.
- Basic structure of a C++ program.
- Object oriented Concept.



Historical Perspective:

The C++ programming language was created by **Bjarne Stroustrup** and his team at Bell Laboratories (AT&T, USA) to help implement simulation projects in an object-oriented and efficient way. The earliest versions, which were originally referred to as “C with classes,” date back to 1980. As the name C++ implies, C++ was derived from the C programming language: ++ is the increment operator in C. As early as 1989 an ANSI Committee (American National Standards Institute) was founded to standardize the C++ programming language. The aim was to have as many compiler vendors and software developers as possible agree on a unified description of the language in order to avoid the confusion caused by a variety of dialects. In 1998 the ISO (International Organization for Standardization) approved a standard for C++ (ISO/IEC 14882).

Characteristics of C++:

C++ is not a purely object-oriented language but a hybrid that contains the functionality of the C programming language. This means that you have all the features that are available in C:

- Universally usable modular programs
- Efficient, close to the machine programming
- Portable programs for various platforms.

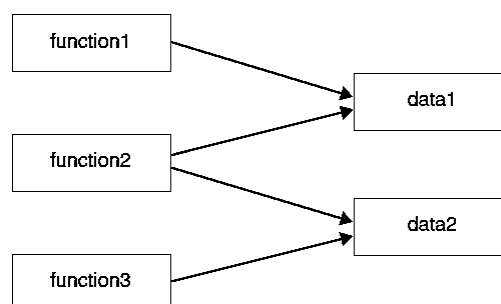
The large quantities of existing C source code can also be used in C++ programs. C++ supports the concepts of object-oriented programming (or OOP for short), which are:

- **data abstraction**, that is, the creation of classes to describe objects
- **data encapsulation** for controlled access to object data
- **inheritance** by creating derived classes (including multiple derived classes)
- **polymorphism** (Greek for multiform), that is, the implementation of instructions that can have varying effects during program execution.

Various language elements were added to C++, such as references, templates, and exception handling. Even though these elements of the language are not strictly object-oriented programming features, they are important for efficient program implementation.

Object-Oriented Programming:

Traditional Concept:



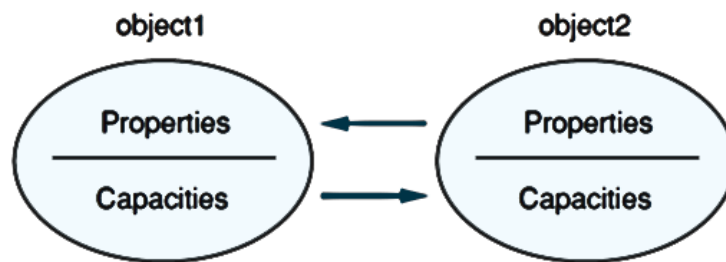
In traditional, procedural programming, data and functions (subroutines, procedures) are kept separate from the data they process. This has a significant effect on the way a program handles data:

- The programmer must ensure that data are initialized with suitable values before use and that suitable data are passed to a function when it is called.

- If the data representation is changed, e.g. if a record is extended, the corresponding functions must also be modified.

Both of these points can lead to errors and neither support low program maintenance requirements.

Object-oriented concept:



Object-oriented programming shifts the focus of attention to the objects, that is, to the

aspects on which the problem is centered. A program designed to maintain bank accounts would work with data such as balances, credit limits, transfers, interest calculations, and so on. An object representing an account in a program will have properties and capacities that are important for account management.

OOP objects combine data (properties) and functions (capacities). A class defines a certain object type by defining both the properties and the capacities of the objects of that type. Objects communicate by sending each other “messages,” which in turn activate another object’s capacities.

Advantages of OOP:

Object-oriented programming offers several major advantages to software development:

- **Reduced susceptibility to errors:** an object controls access to its own data. More specifically, an object can reject erroneous access attempts
- **Easy re-use:** objects maintain themselves and can therefore be used as building blocks for other programs.
- **Low maintenance requirement:** an object type can modify its own internal data representation without requiring changes to the application.

Developing A C++ Program:

Translating a C++ program:

The following three steps are required to create and translate a C++ program:

1. First, a text editor is used to save the C++ program in a text file. In other words, the **source code** is saved to a source file. In larger projects the programmer will normally use modular programming. This means that the source code will be stored in several source files that are edited and translated separately.

2. The source file is put through a compiler for translation. If everything works as planned, an **object file** made up of machine code is created. The object file is also referred to as a module.

3. Finally, the **linker** combines the object file with other modules to form an **executable file**. These further modules contain functions from standard libraries or parts of the program that have been compiled previously.

It is important to use the correct file extension for the source file's name. Although the file extension depends on the compiler you use, the most commonly found file extensions are .cpp and .cc.

Prior to compilation, **header files**, which are also referred to as **include files**, can be copied to the source file. Header files are text files containing information needed by various source files, for example, type definitions or declarations of variables and functions. Header files can have the file extension .h, but they may not have any file extension.

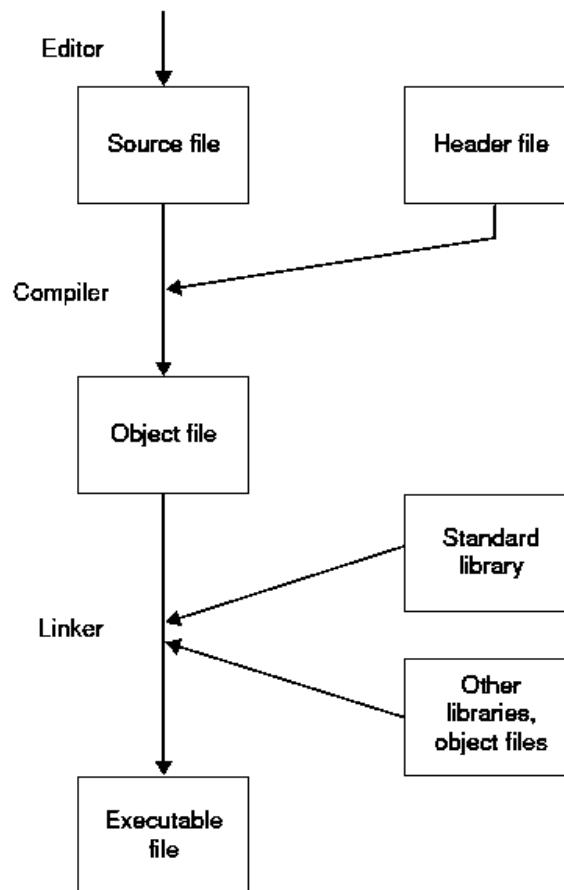
The C++ standard library contains predefined and standardized functions that are available for any compiler.

Modern compilers normally offer an integrated software development environment, which combines the steps mentioned previously into a single task. A graphical user interface is available for editing, compiling, linking, and running the application. Moreover, additional tools, such as a debugger, can be launched.

✓ NOTE

If the source file contains just one *syntax error*, the compiler will report an *error*. Additional error messages may be shown if the compiler attempts to continue despite having found an error. So when you are troubleshooting a program, be sure to start with the first error shown.

In addition to error messages, the compiler will also issue warnings. A warning does not indicate a syntax error but merely draws your attention to a possible error in the program's logic, such as the use of a non-initialized variable.



A Beginner's C++ Program:

Sample program

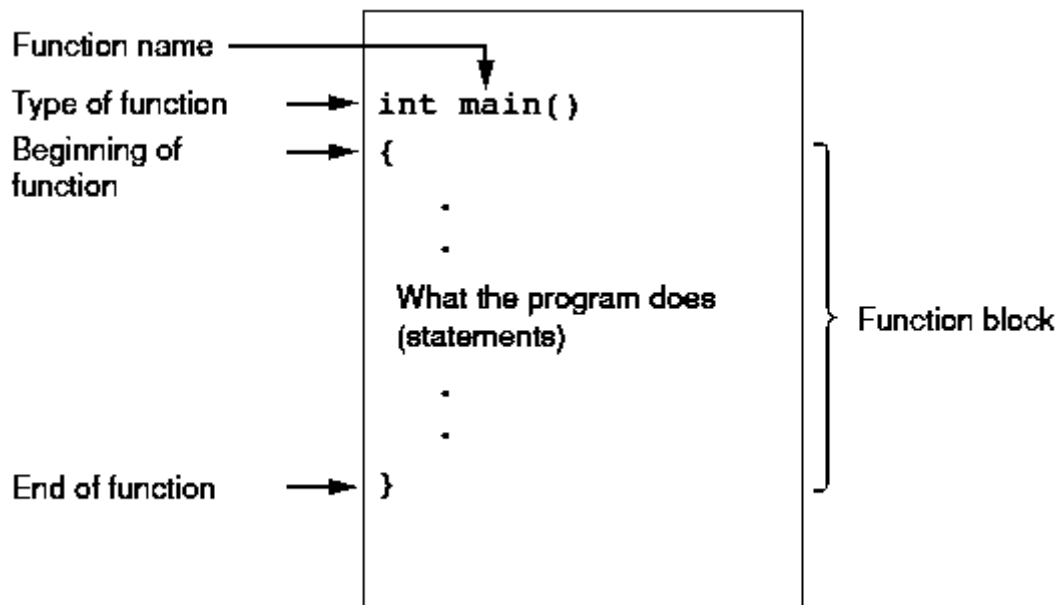
```
#include <iostream>
using namespace std;

int main()
{
    cout << "Enjoy yourself with C++!" << endl;
    return 0;
}
```

Screen output

Enjoy yourself with C++!

Structure of function main():



A Simple C++ program:

```
/******  
A simple C++ program with comments  
*****/  
#include <iostream>                                //Header file  
using namespace std;  
  
int main()                                          // main function  
{  
    cout << "Hello! The program starts in main()." << endl;    //output function  
    cout << "At the end of main()." << endl;  
}
```

Screen output

```
Hello! The program starts in main().  
At the end of main().
```

Start with the `main()` function as this function controls the program flow. In other words, `main()` calls functions that have yet to be defined. This is made possible by supplying the compiler with a function prototype that includes all the information the compiler needs. This example also introduces comments. Strings enclosed in `/* ... */` or starting with `//` are interpreted as comments.

Check your Progress:

Q1: Write a C++ program that outputs the following text on screen:

```
Oh what  
a happy day!  
Oh yes,  
what a happy day!
```

Q2: What is the output of following C++ program?

```
#include <stream>  
int main()  
{  
cout << "If this text";  
cout >> "appears on your display";  
cout << "you can pat yourself on";  
return 0;  
}
```

1.6 SOLUTIONS:

Q1:

```
#include <iostream>  
using namespace std;  
int main()  
{  
cout << " Oh what " << endl;  
cout << " a happy day! " << endl;  
cout << " Oh yes, " << endl;  
cout << " what a happy day! " << endl;  
return 0;  
}
```

Q2:

```
If this text  
appears on your display  
you can pat yourself on
```

Getting Started with C++:

C++ Character Sets:

Letters	A-Z , a-z
Digits	0-9
Special Symbols	Space + - * / ^ \ () [] { } = != < > . , , “ \$, ; : % ! & ? _ # <= >= @
White Spaces	Blank spaces, horizontal tab, carriage return
Other Characters	Any of the 256 ASCII character

C++ Tokens:

The smallest individual unit in a program is known as token. Tokens used in C++ are:

KEYWORDS :

Keywords are the certain reserved words that convey a special meaning to the compiler. These are reserve for special purpose and must not be used as identifier name.eg for , if, else , this , do, etc.

IDENTIFIERS:

Identifiers are programmer defined names given to the various program elements such as variables, functions, arrays, objects, classes, etc.. It may contain digits, letters and underscore, and must begin with a letter or underscore. C++ is case sensitive as it treats upper and lower case letters differently. The following are some valid identifiers: Pen time580 s2e2r3 _dos _HJI3_JK

LITERALS:

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

- Integer literals
- Character literals
- Floating literals
- String literals

INTEGER LITERALS:

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

- (i) **Decimal Integer Literals:-** An integer literal without leading 0 (zero) is called decimal integer literals e.g., 123, 786 , +97 , etc.
- (ii) **Octal Integer Literals:-** A sequence of octal digit starting with 0 (zero) is taken to be an octal integer literal (zero followed by octal digits). e.g., 0345, 0123 , etc.
- (iii) **Hexadecimal Integer Literals :-** Hexadecimal Integer Literals starts with 0x or 0X followed by any hexa digits. e.g., 0x9A45, 0X1234, etc.

CHARACTER LITERALS:

Any single character enclosed within single quotes is a character literal. e.g ‘ A’ , ‘3’

FLOATING LITERALS:

Numbers which are having the fractional part are referred as floating literals or real literals. 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 character surrounded by double quotes. e.g., “abc” , “23”.

PUNCTUATORS:

The following characters are used as punctuators which are also known as separators in C++ [] { } () , ; : * = #

Punctuator	Name	Function
[]	Brackets	These indicates 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.
,	Comma	It is used as a separator.
:	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:

An operator is a symbol or character or word which trigger some operation (computation) on its operands.

- (i) **Unary operators:** Those which require only one operand to operate upon. e.g. unary `-`, unary `+`, `++`, `--` !.
- (ii) **Binary operators:** Binary operators require two operands to operate upon. e.g. `+`, `*`, `/`, `-`, etc.
- (iii) **Ternary Operator:** Ternary operator require three operands to operate upon. Conditional operator (`? :`) is a ternary operator in C++.

Structure of a C++ program:

Structure of a C++ program	A C++ program to prints a string on the screen
<pre>#include<iostream.h> void main () { Statement_1; Statement_2; : : }</pre>	<pre>#include<iostream.h> void main () { cout<< "Kendriya Vidyalaya"; } The program produces following output: Kendriya Vidyalaya</pre>

COMMENTS IN A C++ PROGRAM.:

Comments are the pieces of code that compiler ignores to compile. There are two types of comments in C++.

1. **Single line comment:** The comments that begin with `//` are single line comments. The Compiler simply ignores everything following `//` in the same line.
2. **Multiline Comment:** The multiline comment begin with `/*` and end with `*/` . This means everything that falls between `/*` and `*/` is consider a comment even though it is spread across many lines.

INPUT OUTPUT (I/O) OPERATIONS IN C++:

Input & Output operations are supported by the `istream` (input stream) and `ostream` (output stream) classes. The predefined stream objects for input, output are:

(i) **The `cout` Object:** The identifier **`cout`** is a predefined object of `ostream` class that represents the standard output stream in C++ and tied to standard output. **`cout`** stands for console output. **`cout`** sends all output to the standard output device i.e. monitor. The syntax of **`cout`** is as follows:

`cout<< data;`

Where data may be a variable or constant or string etc. e.g.

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

Output Operator (<<): The output operator (<<) is also known as ‘stream insertion’ or ‘put to’ operator. It directs the contents of the variable (or value) on its right to the object on its left (i.e. `cout`).

(ii) **The `cin` Object:** The **`cin`** object is an `istream` class object tied to standard input. **`cin`** stands for console input. `cin` object 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)

Input Operator (>>): The input operator (>>) is also known as extraction or ‘get from’ operator. It extracts (or takes) the value from the keyboard and assigns it to the variable on its right.

CASCADING OF OPERATOR:

When input or output (>> or <<) are used more than one time in a single statement then it is called as cascading of operators.

e.g `cout<< roll<< age<< endl;`

DATA TYPES, VARIABLES & CONSTANTS:-

DATA TYPES IN C++:

Data types are means to identify the types of data and associated operations of handling it. Data types in C++ are of two types:

1. Fundamental or Built-in data types.
2. Derived data types.

1. Fundamental or Built-in data types: These data types are already known to compiler. These are the data types those are not composed of other data types. There are following fundamental data types in C++:

- (i) int data type (for integer) :- int data type is used for integer value. An identifiers declare as int cannot have fractional part.
- (ii) char data type (for characters):- An identifiers declare as char can store a character.
- (iv) float data type (for floating point numbers):- An identifier declare as float can hold a floating point number.
- (v) double data type (for double precision floating point numbers):- The double data type is also used for handling floating point numbers but it occupies twice as much memory as float and store numbers with much larger range and precision.

Data Type Modifiers:-There are following four data type modifiers in C++ , which may be used to modify the fundamental data types to fit various situations more precisely:

- (i) signed
- (ii) unsigned
- (iii) long
- (iv) short

signed, unsigned, long, short data type modifiers may be apply to char & int data types. However you may also apply long to double

Data Type	Size (in Bytes)	Range
char	1	-128 to 127
unsigned char	1	0 to 255
Signed char	1	Same as char
Int	2	-32768 to 32767
unsigned int	2	0 to 65535
signed int	2	Same as int
short (or short int)	2	-32768 to 32767
long (or long int)	4	-2147483648 to 2147483647
float	4	3.4×10^{-38} to $3.4 \times 10^{38} - 1$ (upto 7 digits of precision)
double	8	1.7×10^{-308} to $1.7 \times 10^{308} - 1$ (upto 15 digits of precision)
long double	10	3.4×10^{-4932} to $1.1 \times 10^{4932} - 1$ (upto 19 digits of precision)

2.Derived Data Types:- These are the data types that are composed of fundamental data types. e.g., array, class, structure, etc.

Variables:- A named memory location, whose contains can be changed with in program execution is known as variable.

OR

A variable is an identifier that denotes a storage location, which contains can be varied during program execution.

Declaration of Variables:- All variables must be declared before they are used in executable statements. Variable declaration reserves memory required for data storage and associates it with a name. Syntax for variable declaration is:

datatypes variable_name1, variable_name2, variable_name3,..... ;

e.g.,

int num;

int num, sum, avg;

We can also initialize a variable at the time of declaration by using following syntax:

datatypes variable_name = value;

e.g.,

int num = 0;

Constant:- A named memory location, whose contains cannot be changed with in program execution is known as constant.

OR

A constant is an identifier that denotes a storage location, which contains cannot be varied during program execution. Syntax for constant declaration is:

const **datatypes** **constant_name** = **value** ;

e.g.,

const float **pi** = 3,14f ;

FORMATTED OUTPUT (MANIPULATORS):

Manipulators are the operators used with the insertion operator << to format the data

display. The most commonly used manipulators are endl and setw.

1. **The endl manipulator:** The endl manipulator, when used in a output statement , causes a line feed to be inserted. It has same effect as using new line character “\n”.

e.g.,

```
cout<< “ Director School Education Kashmir”<<endl;
```

```
cout<< “ Computer Science Book”;
```

The output of the above code will be

```
Director School Education Kashmir
Computer Science Book
```

2. **The setw() Manipulator:** The setw() manipulator sets the width of the field assign for the output. It takes the size of the field (in number of character) as a parameter. The

output will be right justified e.g., the code:

```
cout<<setw(6)<<”R” ;
```

Generates the following output on the screen (each underscore represent a blank space)

```
_ _ _ _ _ R
```

In order to use these manipulator , it is must to include header file iomanip.h

OPERATORS AND EXPRESSIONS:-

OPERATORS:-

Arithmetic operators:- Those operators are operates only on numeric data types operands are known as arithmetic operators.

Operator	Operation	Example
Unary -	Result is the negation of operand's value (Reverse the sign of operand's value)	If a=5 then - a means -5. If a = - 4 then - a means 4.
Unary +	The result is the value of its operand	If a=5 then +a means 5. If a = - 4 then +a means -4.
+ (Addition Operator)	Addition (it adds two numbers)	4+20 results is 24. If a = 5 then a + 5 results 10.
- (Subtraction Operator)	Subtraction (Subtract the second operand from first)	14 - 3 evaluates to 12. If a = 2 , b = 3 then b - a evaluates 1.
* (Multiplication Operator)	Multiplies the values of its operands	3*4 evaluates to 12. If a=2, b=3 then a*b evaluates to 6.
/ (Division Operator)	Divides its first operand by the second	100/5 evaluates 20. If a =10 , b = 5 then a/b evaluates 2
% (Modulus Operator)	It produce the remainder of dividing the first operand by second	19%6 evaluates to 1. If a = 14 , b = 3 then a%b evaluates to 2. Modulus operator requires that both operands be integer and second operand be non-zero.

Increment and Decrement Operators (++ , - -) :

The increment operator (++) adds 1 to its operand and decrement operator (--) subtract one from its operand. In other word

a = a + 1; is same as ++a; or a++;
& a = a - 1 ; is same as --a; or a--;

Both the increment & decrement operators come in two versions:

(i) Prefix increment/decrement:- When an increment or decrement operator precedes its operand, it is called prefix increment or decrement (or pre-increment / decrement). In prefix increment/decrement , C++ perform the increment or decrement operation before using the value of the operand. e.g.,

If sum = 10 and count =10 then
Sum = sum +(++count);
First count incremented and then evaluate sum = 21.

(ii) Postfix increment/decrement:- When an increment or decrement operator follows its operand, it is called postfix increment or decrement (or post-increment / decrement). In postfix increment/decrement , C++ first uses the value of the

operand in evaluating the expression before incrementing or decrementing the operand's value. e.g.,

If sum = 10 and count = 10 then

Sum = sum + (count++);

First evaluate sum = 20, and then increment count to 11.

Relational Operator: These operators are used to compare two values. If comparison is true, the relational expression results into the value 1 and if the comparison is false its result will be 0.

The six relational operators are:

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

Logical Operators: In addition to the relational operator, C++ contains three logical

operators. Relational operators often are used with logical operators to construct more complex decision making expressions.

Operators	Use	Return True if
&& (Logical AND)	op1 && op2	op1 and op2 are both true
(Logical OR)	op1 op2	Either op1 or op2 is true
! (Logical NOT)	!op1	op1 is false (it is unary operator)

Assignment Operator: C++ offers an assignment operator (=) to assign a value to an

identifier. The assignment statement that make use of this operator are written in the form :

var = expression ;

where var generally represents a variable and expression may be a constant or a variable or an expression.

C++ offers special shorthand operators that simplify the coding of a certain type of assignment statement . e.g.,

`a = a + 10 ;` can be written as `a+=10 ;`

This shorthand works for all binary arithmetic operators. The general form of this shorthand is

`Var = var operator expression ;` is same as
`var operator = expression ;`

Following are some examples of C++ shorthands:

`x -=10 ;` equivalent to `x = x -10 ;`

`x*=3 ;` equivalent to `x = x * 3 ;`

`x/=2 ;` equivalent to `x = x/2 ;`

`x%=z` equivalent to `x = x % z ;`

Conditional operator (? :):

The conditional operator (? :) is a ternary operator i.e., it require three operands. The general form of conditional operator is:

expression1? expression2: expression3;

Where expression1 is a logical expression, which is either true or false. If expression1 evaluates to true i.e., 1, then the value of whole expression is the value of expression2, otherwise, the value of the whole expression is the value of expression3. For example

`min = a<b? a : b;`

Here if expression (`a<b`) is true then the value of a will be assigned to min otherwise value of b will be assigned to min.

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.

sizeof():

This operator returns the size of its operand in bytes. The operand may be an expression or identifier or it may be a data type.

a = sizeof (char);

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

EXPRESSIONS:-

An expression in C++ is any valid combination of operators, constants, and variables. **Pure Expressions:-** If an expression have all operand of same data types then it is called a pure expression.

Mixed Expressions:- If an expression have operands of two or more different data types then it is called a mixed expression.

Arithmetic Expressions:- Arithmetic expression can either be integer expressions or real expressions. Sometimes a mixed expression can also be formed which is a mixture of real and integer expressions.

Integer Expressions:- Integer expressions are formed by connecting all integer operands using integer arithmetic operators.

Real Expressions:- Real expressions are formed by connecting real operands by using real arithmetic operators.

Logical Expressions:- The expressions which results evaluates either 0 (false) or 1 (true) are called logical expressions. The logical expressions use relational or Logical operators.

TYPE CONVERSION:- The process of converting one predefined data type into another is called type conversion. C++ facilitates the type conversion in two forms:

(i) Implicit type conversion:- An implicit type conversion is a conversion performed

by the compiler without programmer's intervention. An implicit conversion is applied

generally whenever different data types are intermixed in an expression. The C++ compiler converts all operands upto the data type of the largest data type's operand,

which is called type promotion.

(ii) Explicit type conversion :- An explicit type conversion is user-defined that forces

an expression to be of specific data type.

TYPE CASTING:-

The explicit conversion of an operand to a specific type is called type casting.

Type Casting Operator - (type) :-Type casting operators allow you to convert a data item of a given type to another data type. To do so , the expression or identifier must be preceded by the name of the desired data type , enclosed in parentheses . i. e.,

(data type) expression

Where data type is a valid C++ data type to which the conversion is to be done. For example , to make sure that the expression $(x+y/2)$ evaluates to type float , write it as:

(float) (x+y/2)

Precedence of Operators:- Operator precedence determines which operator will be performed first in a group of operators with different precedence. For instance

5 + 3 * 2 is calculated as 5 + (3 * 2), giving 11

Unit 6

Programing in C++

Objectives:

In this Chapter you will learn about:

- Statements
- Types of Statements
- Loops.
- Types of Loops



Statements:

STATEMENTS:-

Statements are the instructions given to the Computer to perform any kind of action.

Null Statement:- A null statement is useful in those case where syntax of the language requires the presence of a statement but logic of program does not give permission to do anything then we can use null statement. A null statement is nothing only a ;. A null (or empty statement have the following form:

; // only a semicolon (;)

Compound Statement :- A compound statement is a group of statements enclosed in the braces { }. A Compound statement is useful in those case where syntax of the language requires the presence of only one statement but logic of program have to do more thing i.e., we want to give more than one statement in place of one statement then we can use compound statement.

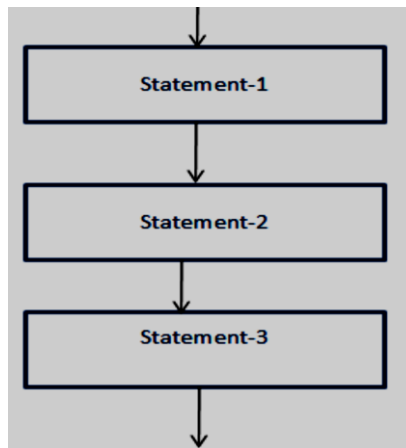
```
{  
St-1;  
St-2;  
:  
:  
}
```

Statement Flow Control:- In a program , statements may be executed sequentially, selectively, or iteratively.

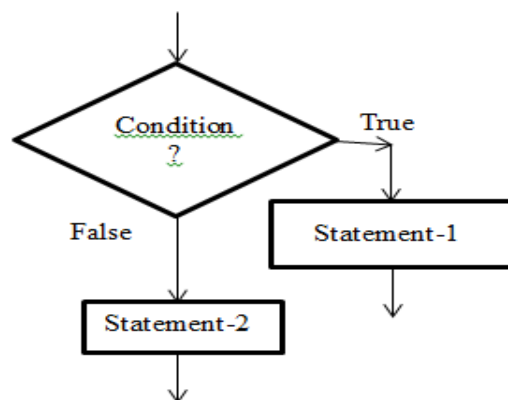
Every programming language provides three constructs:

1. Sequence Constructs
2. Selection Constructs
3. Iteration Constructs

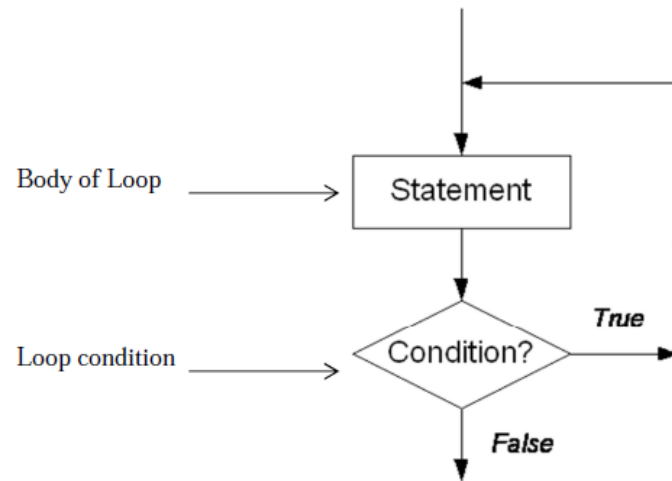
Sequence Construct:- The sequence construct means the statements are being executed sequentially. It represents the default flow of statements.



Selection Construct:- The selection construct means the execution of statement(s) depending on a condition. If a condition is true, a group of statements will be execute otherwise another group of statements will be execute.

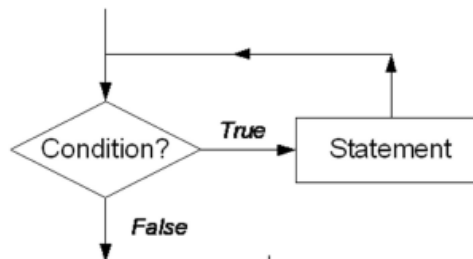


Looping or Iteration Statements:- Looping the iteration construct means repetition of set of statements depending upon a condition test. The iteration statements allow a set of instructions to be performed repeatedly until a certain condition is true.

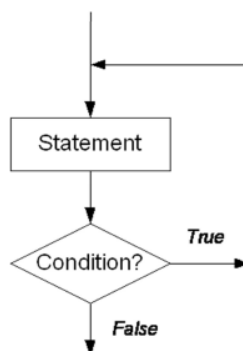


There are two types of loops:-

1. **Entry-controlled loop:-** In entry-controlled loop first of all loop condition is checked and then body of loop is executed if condition is true. If loop condition is false in the starting the body of loop is not executed even once.



2. **Exit-controlled loop :-** In exit-controlled loop first body of loop is executed once and then loop condition is checked. If condition is true then the body of loop will be executed again. It means in this type of loop, loop body will be executed once without checking loop condition.



SELECTION STATEMENTS :-

There are two types of selection statements in C++:

1. **if statement**
2. **switch statement**

1. if Statement: If statement have three forms

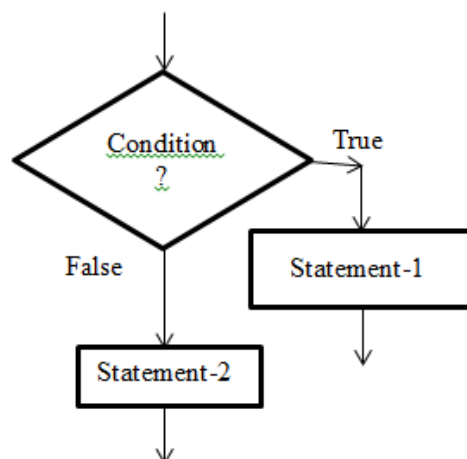
(a) if ... else Statement :- It is useable, when we have to performs an action if a condition is True and we have to perform a different action if the condition is false. The syntax of if...else statement is:

```
if ( < conditional expression > )  
{  
< statement-1 or block-1>;  
// statements to be executed when conditional expression is true.  
}  
else  
{  
< statement-2 or block-2>;  
// statements to be executed when conditional expression is false.  
}
```

If the <conditional expression> is evaluated to true then the < statement-1 or block-1>

(statement under if () block) will be executed otherwise the <statement-2 or block-2>

(statements under else block) would be executed. if there exists only one program statement under if() block then we may omit curly braces { }.



Sample program

```
// if_else.cpp
// Demonstrates the use of if-else statements

#include <iostream>
using namespace std;
int main()
{
    float x, y, min;

    cout << "Enter two different numbers:\n";
    if( cin >> x && cin >> y)    // If both inputs are
    {                             // valid, compute
        if( x < y )               // the lesser.
            min = x;
        else
            min = y;
        cout << "\nThe smaller number is: " << min << endl;
    }
    else
        cout << "\nInvalid Input!" << endl;

    return 0;
}
```

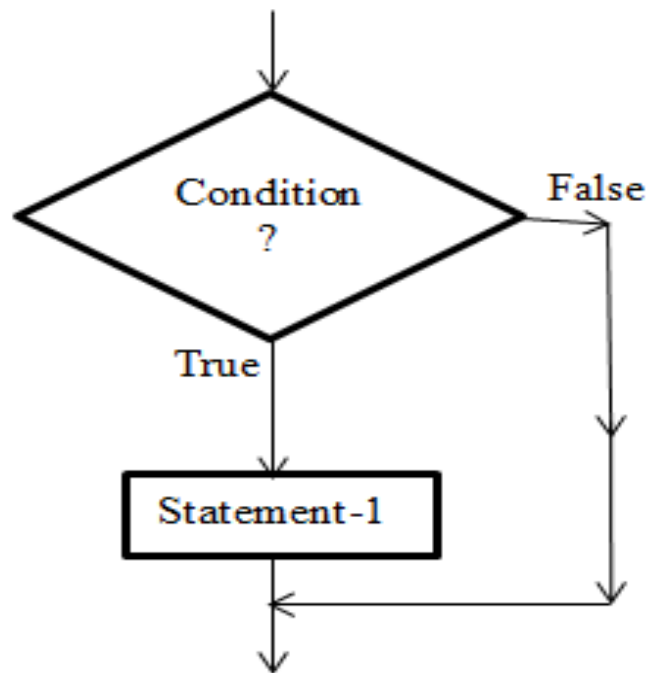
Sample output for this program

```
Enter two different numbers:
7.5 5.7
The smaller number is: 5.7
```

(b) Simple if statement:- The else part in if ... else statement is optional, if we omit the else part then it becomes simple if statement. This statement is usable, when we have to either perform an action if a condition is True or skips the action if the condition is false. The syntax of simple if statement is:

```
if ( < conditional expression > )
{
    < statement-1 or block-1>;
    // statements to be executed when conditional expression is true.
}
```

Here <statement-1 or block-1> will be executed only if <conditional expression > evaluates true. if there exists only one program statement under if() block then we may omit curly braces { } .



Sample program

```
// greater.cpp
#include <iostream>
using namespace std;

int main()
{
    float x, y;

    cout << "Type two different numbers:\n";
    if( !(cin >> x && cin >> y) ) // If the input was
    {                               // invalid.
        cout << "\nInvalid input!" << endl;
    }
    else
    {
        cout << "\nThe greater value is: "
              << (x > y ? x : y) << endl;
    }

    return 0;
}
```

Sample output for this program

```
Type two different numbers:
173.2
216.7
The greater value is: 216.7
```

(c) The if-else-if ladder :- This statement allows you to test a number of mutually exclusive cases and only execute one set of statements for which condition evaluates true first. The syntax is:

if (<condition -1>)
statement-1; // do something if condition-1 is satisfied (True)

```
else if ( <condition – 2 >)  
statement-3 ; // do something if condition -2 is satisfied (True)  
else if (<condition – 3 >)  
statement-3 ; // do something if condition- 3 is satisfied (True)  
:  
: // many more n-1 else - if ladder may come  
:  
else if( < condition – n >)  
statement-n ; // do something if condition – n is satisfied (True)  
else  
statement-m ; // at last do here something when none of the  
// above conditions gets satisfied (True)  
}
```

<>	in syntax is known as a place holder, it is not a part of syntax, do not type it while writing program. It only signifies that anything being kept there varies from program to program.
[]	is also not a part of syntax , it is used to mark optional part of syntax i.e. all part of syntax between [] is optional.

In the above syntax there are ladder of multiple conditions presented by each if() , all of these conditions are mutually exclusive. If one of them would evaluates true then the statement followed that condition will be executed and all the conditions below it would not be evaluated (checked).

Say suppose if condition-3 gets satisfy (i.e. evaluates true value for the condition), then

statement-3 gets executed and all other conditions below it would be discarded.

If none of the n if () conditions gets satisfied then the last else part always gets executed. It is not compulsory to add an else at the last of the ladder.

We can also write a group of statement enclosed in curly braces { } (as a compound statement) in place of any statement (statement-1. Statement-2,....., statement-n) if required in above syntax.

Nested if Statement:- If an if statement is written in the if or else clause of another if statement then it is known as nested if. Some possible syntax of nested if statements given below:

Syntax 1:-

```
if ( <outer- condition > )
```

```
{
    if ( <inner-condition> )
    {
        //some statements to be executed
        // on satisfaction of inner if ( ) condition.
    } // end of scope of inner if( )
    //some statements to be executed
    // on satisfaction of outer if ( ) condition.
} // end of the scope of outer if( )
```

Syntax 2:-

```
if ( <outer- condition > )
{
    if ( <inner-condition> )
    {
        //some statements to be executed
        // on satisfaction of inner if ( ) condition.
    }
else
{
    // statements on failure of inner if( )
}
```

<> in syntax is known as a place holder, it is not a part of syntax, do not type it while writing program. It only signifies that anything being kept there varies from program to program.

[] is also not a part of syntax , it is used to mark optional part of syntax i.e. all part of syntax between [] is optional.

```
        //some statements to be executed
        // on satisfaction of outer if ( ) condition.
    }
else
{
    // statements on failure of outer if( )
}
```

Example:

```
if( n > 0 )
{
    if( n%2 == 1 )
    {
        cout << " Positive odd number ";
    }
}
```

```
    }  
}  
else  
cout << "Positive even number";
```

2. Switch Statement :- This is multi-branching statement. Syntax of this statement is as follows:

```
switch (expression/variable)  
{  
case value_1: statement -1;  
break;  
case value_2: statement -2;  
break;  
:  
:  
case value_n: statement -n;  
break;  
[ default: statement -m ]  
}
```

Note: expression/variable should be integer or character type only.

When the switch statement is executed, the expression/variable is evaluated and control is transferred directly to the statement whose case label value matches the value of expression/ variable. If none of the case label value matches the value of expression/variable then only the statement following the default will be executed. If no default statement is there and no match is found then no action take place. In this case control is transferred to the statement that follows the switch statement.

Example:

```
// Evaluates given input.  
int command = menu();           // The function menu() reads  
                                // a command.  
  
switch( command )             // Evaluate command.  
{  
  case 'a':  
  case 'A':  
  action1();                   // Carry out 1st action.  
  break;  
  case 'b':  
  case 'B':  
  action2();                   // Carry out 2nd action.  
  break;  
  default:  
  cout << '\a' << flush;      // Beep on  
  }                             // invalid input
```

ITERATIVE STATEMENTS:

Loops in C++:- There are three loops or iteration statements are available in C++

1. for loop
2. while loop
3. do.... while loop



1. The for Loop: For loop is a entry control loop the syntax of for loop is :

```
for(initialization_expression(s); loop_Condition; update_expression)
{
    Body of loop
}
```

Working of the for Loop:-

1. The initialization_expression is executed once, before anything else in the for loop.
2. The loop condition is executed before the body of the loop.
3. If loop condition is true then body of loop will be executed.
4. The update expression is executed after the body of the loop
5. After the update expression is executed, we go back and test the loop condition again, if loop_condition is true then body of loop will be executed again, and it will be continue until loop_condition becomes false.

Example:

```
for (int i = 0; i < 7; i++)
    cout<< i * i << endl;
```

Interpretation:

An int i is declared for the duration of the loop and its value initialized to 0. i2 is output in the body of the loop and then i is incremented. This continues until i is 7.

Sample Programe:

```
// Euro1.cpp
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
    double rate = 1.15;                // Exchange rate:
```

```
                                // one Euro to one
                                Dollar
cout << fixed << setprecision(2);
cout << "\tEuro \tDollar\n";
for( int euro = 1; euro <= 5; ++euro)
cout << "\t " << euro << "\t " << euro*rate << endl;
return 0;
}
```

Screen output

```
Euro Dollar
1 0.95
2 1.90
3 2.85
4 3.80
5 4.75
```

2. While Loop:- while loop is also an entry controlled loop. The syntax of while loop is:

```
while (loop_condition)
{
Loop_body
}
```

Where the Loop_body may contain a single statement, a compound statement or an empty statement.

The loop iterates (Repeatedly execute) while the loop_condition evaluates to true. When the loop_condition becomes false, the program control passes to the statement after the loop_body.

In while loop , a loop control variable should be initialized before the loops begins. The loop variable should be updated inside the loop body.

// average.cpp

Sample Programme for While loop:

```
// Computing the average of numbers
#include <iostream>
using namespace std;
int main()
{
```

```
int x, count = 0;
float sum = 0.0;
cout << "Please enter some integers:\n"
      "(Break with any letter)"
      << endl;
while( cin >> x )
{
    sum += x;
    ++count;
}
cout << "The average of the numbers: "
      << sum / count << endl;
return 0;
}
```

Sample output from the above program

```
Please enter some integers:
(Break with any letter)
9 10 12q
The average of the numbers: 10.3333
```

3. do-while loop:- do-while loop is an exit-controlled loop i.e. it evaluates its loop_condition at the bottom of the loop after executing its loop_body statements. It means that a do-while loop always executes at least once. The syntax of do-while loop is:

```
do
{
    Loop_body
}while (loop_condition);
```

In do-while loop first of all loop_body will be executed and then loop_condition will be evaluates if loop_condition is true then loop_body will be executed again, When the loop_condition becomes false, the program control passes to the statement after the loop_body.

Sample Programe:

```
// tone.cpp
#include <iostream>
using namespace std;
const long delay = 10000000L;
int main()
{
    int tic;
    cout << "\nHow often should the tone be output? ";
    cin >> tic;
do
{
    for( long i = 0; i < delay; ++i );
    cout << "Now the tone!\a" << endl;
}
while( --tic > 0 );
cout << "End of the acoustic interlude!\n";
return 0;
}
```

Nested Loops:- Any looping construct can also be nested within any other looping construct . Let us look at the following example showing the nesting of a for() loop within the scope of another for() loop :

```
for(int i = 1 ; i<=2 ; i++) Outer for( ) loop
{
    for( int j = 1 ; j<=3 ; j++) Inner for( ) loop
    {
        cout<< i * j <<endl ;
    }
}
```

For each iteration of the outer for loop the inner for loop will iterate fully up to the last value of inner loop iterator. The situation can be understood more clearly as :

1st Outer Iteration

i= 1

1st Inner Iteration

j = 1 , output : 1 * 1 = 1

2nd Inner Iteration

$j = 2$, output : $1 * 2 = 2$

3rd Inner Iteration

$j = 3$, output : $1 * 3 = 3$

2nd Outer Iteration

$i = 2$

1st Inner Iteration

$j = 1$, output : $2 * 1 = 2$

2nd Inner Iteration

$j = 2$, output : $2 * 2 = 4$

3rd Inner Iteration

$j = 3$, output : $2 * 3 = 6$

You can observe that j is iterated from 1 to 2 every time i is iterated once.

JUMP STATEMENTS:-

These statements unconditionally transfer control within function . In C++ four statements perform an unconditional branch :

1. **return**
2. **goto**
3. **break**
4. **continue**

1. Return Statement:- The return statement is used to return from a function. It is useful in two ways:

- (i) An immediate exit from the function and the control passes back to the operating system which is main's caller.

(ii) It is used to return a value to the calling code.

2. Goto statement :- A goto Statement can transfer the program control anywhere in the program. The target destination of a goto statement is marked by a label. The target label and goto must appear in the same function. The syntax of goto statement is:

```
goto label;  
:  
label :
```

Example :

```
a= 0;  
start :  
    cout<<"\n" <<++a;  
if(a<50) goto start;
```

3. Break Statement :- The break statement enables a program to skip over part of the code. A break statement terminates the smallest enclosing while, do-while, for or switch statement. Execution resumes at the statement immediately following the body of the terminated statement.

4. Continue Statement:- The continue is another jump statement like the break statement as both the statements skip over a part of the code. But the continue statement is somewhat different from break. Instead of forcing termination, it forces the next iteration of the loop to take place, skipping any code in between.

Unit 7

User Defined Functions

Objectives:

In this Chapter you will learn about:

- Functions
- Function prototype
- Call by value
- Call by reference



Functions in C++

FUNCTION:

Function is a named group of programming statements which perform a specific task and return a value. There are two types of functions:-

1. Built-in (Library) functions
2. User defined functions

Built-in Functions (Library Functions) :- The functions, which are already defined in C++ Library (in any header files) and a user can directly use these function without giving their definition is known as built-in or library functions. e.g., `sqrt( )`, `toupper( )`, `isdigit( )` etc.

Following are some important Header files and useful functions within them:

Header Files	Built-in functions
stdio.h (standard I/O function)	<code>gets()</code> , <code>puts()</code>
ctype.h (character type function)	<code>isalnum()</code> , <code>isalpha()</code> , <code>isdigit ()</code> , <code>islower ()</code> , <code>isupper ()</code> , <code>tolower ()</code> , <code>toupper()</code>
string.h (string related function)	<code>strcpy ()</code> , <code>strcat ()</code> , <code>strlen()</code> , <code>strcmp()</code> , <code>strcmpi()</code> , <code>strrev()</code> , <code>strupr()</code> , <code>strlwr()</code>
math.h (mathematical function)	<code>fabs ()</code> , <code>pow ()</code> , <code>sqrt ()</code> , <code>sin ()</code> , <code>cos ()</code> , <code>abs ()</code>
stdlib.h	<code>randomize ()</code> , <code>random ()</code>

The above list is just few of the header files and functions available under them , but actually there are many more. The calling of library function is just like User defined function , with just few differences as follows:

- i) We don't have to declare and define library function.
- ii) We must include the appropriate header files , which the function belongs to, in global area so as these functions could be linked with the program and called.

Library functions also may or may not return values. If it is returning some values then the value should be assigned to appropriate variable with valid datatype.

gets() and puts() : these functions are used to input and output strings on the console during program run-time.

gets() accept a string input from user to be stored in a character array.

puts() displays a string output to user stored in a character array.

isalnum() , isalpha() , isdigit() : checks whether the character which is passed as parameter to them are alphanumeric or alphabetic or a digit ('0' to '9') . If checking is true functions returns 1.

islower (), isupper (), tolower (), toupper() : islower() checks whether a character is lower case , isupper() check whether a character is upper case . tolower() converts any character passed to it in its lower case and the toupper() convert into upper case.

fabs (), pow (), sqrt (), sin (), cos (), abs () :

randomize (), random () : The above functions belongs to header file stdlib.h . Let us observe the use of these functions:

randomize() : This function provides the seed value and an algorithm to help random() function in generating random numbers. The seed value may be taken from current system's time.

random(<int>) : This function accepts an integer parameter say x and then generates a random value between 0 to x-1

For example : random(7) will generate numbers between 0 to 6.

To generate random numbers between a lower and upper limit we can use following

Formula

random(U – L +1) + L

where U and L are the Upper limit and Lower limit values between which we want to find out random values.

For example : If we want to find random numbers between 10 to 100 then we have to write code as :

random(100 -10 +1) + 10 ; // generates random number between 10 to 100

User-defined function:- The functions which are defined by user for a specific purpose is known as user-defined function. For using a user-defined function it is required, first define it and then using.

Function Prototype:- Each user define function needs to be declared before its usage in the program. This declaration is called as function prototype or function declaration. Function prototype is a declaration statement in the program and is of the following form :

Return_type function_name(List of formal parameters) ;

Declaration of user-defined Function: In C++ , a function must be defined, the general form of a function definition is :

```
Return_type function_name(List of formal parameters)  
{  
    Body of the function  
}
```

Where Return_type is the data type of value return by the function. If the function does not return any value then void keyword is used as return_type.

List of formal parameters is a list of arguments to be passed to the function. Arguments have data type followed by identifier. Commas are used to separate different arguments in this list. A function may be without any parameters, in which case , the parameter list is empty. statements is the function's body. It is a block of statements surrounded by braces { }.

Function_name is the identifier by which it will be possible to call the function.
e.g.,

```
int addition (int a, int b)  
{
```

```
    int r ;  
    r=a+b ;  
    return (r) ;  
}
```

Calling a Function:- When a function is called then a list of actual parameters is supplied that should match with formal parameter list in number, type and order of arguments. Syntax for calling a function is:

function_name (list of actual parameters);
e.g.,

```
#include <iostream>  
int addition (int a, int b)  
{  
    int r;  
    r=a+b;  
    return (r);  
}  
void main ( )  
{  
    int z ;  
    z = addition (5,3);  
    cout<< "The result is " << z;  
}
```

The result is 8

Call by Value (Passing by value) :- The call by value method of passing arguments to a function copies the value of actual parameters into the formal parameters , that is, the function creates its own copy of argument values and then use them, hence any change made in the parameters in function will not reflect on actual parameters . The above given program is an example of call by value.

Call by Reference (Passing by Reference) :- The call by reference method uses a different mechanism. In place of passing value to the function being called , a reference to the original variable is passed . This means that in call by reference method, the called function does not create its own copy of original values , rather, it refers to the original values only by different names i.e., reference . thus the called function works the original data and any changes are reflected to the original values.

```
// passing parameters by reference
#include <iostream.h>
void duplicate (int& a, int& b, int& c)
{
    a*=2;
    b*=2;
    c*=2;
}

void main ()
{
    int x=1, y=3, z=7;
    duplicate (x, y, z);
    cout <<"x="<< x <<" , y="<< y <<" , z="<< z;
}
```

output :x=2, y=6, z=14

The ampersand (&) (address of) is specifies that their corresponding arguments are to be passed by reference instead of by value.

Constant Arguments:- In C++ the value of constant argument cannot be changed by the function. To make an argument constant to a function , we can use the keyword const as shown below:

```
int myFunction( const int x , const int b );
```

The qualifier const tell the compiler that the function should not modify the argument. The compiler will generate an error when this condition is violated.

Default Arguments:- C++ allows us to assign default value(s) to a function's parameter(s) which is useful in case a matching argument is not passed in the function call statement. The default values are specified at the time of function definition. e.g.,

```
float interest ( float principal, int time, float rate = 0.70f)
```

Here if we call this function as:

```
si_int= interest(5600,4);
```

then rate =0.7 will be used in function.

Formal Parameters:- The parameters that appear in function definition are formal parameters.

Actual Parameters:- The parameters that appears in a function call statement are actual parameters.

Functions with no return type (The use of void):- If you remember the syntax of a function declaration:

Return_type function_name(List of formal parameters)

you will see that the declaration begins with a type, that is the type of the function itself (i.e., the data type of value that will be returned by the function with the return statement). But what if we want to return no value?

Imagine that we want to make a function just to show a message on the screen. We do not need it to return any value. In this case we should use the void type specifier for the function. This is a special specifier that indicates absence of type.

The return Statement :- The execution of return statement, it immediately exit from the function and control passes back to the calling function (or, in case of the main(), transfer control back to the operating system). The return statement also returns a value to the calling function. The syntax of return statement is:

return (value);

Scope of Identifier:- The part of program in which an identifier can be accessed is known as scope of that identifier. There are four kinds of scopes in C++

- (i) **Local Scope :-** An identifier declare in a block ({ }) is local to that block and can be used only in it.
- (ii) **Function Scope :-** The identifier declare in the outermost block of a function have function scope.
- (iii) **File Scope (Global Scope):-** An identifier has file scope or global scope if it is declared outside all blocks i.e., it can be used in all blocks and functions.
- (iv) **Class Scope :-** A name of the class member has class scope and is local to its class.

Lifetime :- The time interval for which a particular identifier or data value lives in the memory is called Lifetime of the identifier or data value.

Examples 1:

```
// function example
#include <iostream>
using namespace std;
int addition (int a, int b)
{
    int r;
    r=a+b;
    return (r);
}
int main ()
{
    int z;
    z = addition (5,3);
    cout << "The result is " << z;
    return 0;
}    The result is 8
```

Examples 2:

```
// function example
#include <iostream>
using namespace std;
int subtraction (int a, int b)
{
    int r;
    r=a-b;
    return (r);
}
int main ()
{
    int x=5, y=3, z;
    z = subtraction (7,2);
    cout << "The first result is " << z << '\n';
    cout << "The second result is " << subtraction (7,2) << '\n';
    cout << "The third result is " << subtraction (x,y) << '\n';
    z= 4 + subtraction (x,y);
    cout << "The fourth result is " << z << '\n';
    return 0;
}
```

```
}
```

Output on Screen:

The first result is 5
The second result is 5
The third result is 2
The fourth result is 6

Examples 3: function without arguments

```
// void function example
#include <iostream>
using namespace std;
void printmessage ()
{
    cout << "I'm a function!";
}
int main ()
{
    printmessage ();
    return 0;
}
```

Output on Screen:

I'm a function!

Examples 4: function with call by reference

```
// passing parameters by reference
#include <iostream>
using namespace std;
void duplicate (int& a, int& b, int& c)
{
    a*=2;
    b*=2;
    c*=2;
}
int main ()
{
    int x=1, y=3, z=7;
    duplicate (x, y, z);
    cout << "x=" << x << ", y=" << y << ", z=" << z;
```

```
    return 0;
}
```

Output on Screen:

x=2, y=6, z=14

Examples 4: function with call by value

```
// default values in functions
#include <iostream>
using namespace std;
int divide (int a, int b=2)
{
    int r;
    r=a/b;
    return (r);
}
int main ()
{
    cout << divide (12);
    cout << endl;
    cout << divide (20, 4);
    return 0;
}
```

Output on Screen:

6
5

Unit 8

Arrays and Structures

Objectives:

In this Chapter you will learn about:

- Concept of Array
- Types of Arrays
- User defined data types: Structures



Structured Data type: Arrays

ARRAYS:-

Array is a group of same data type variables which are referred by a common name.

An array is a series of elements of the same type placed in contiguous memory locations that can be individually referenced by adding an index to a unique identifier.

Need for Arrays:- Suppose we have to store roll number & marks of 50 students in a program, then in all we need 100 variable each having a different name. Now remembering and managing these 100 variables is not easy task and it will make the program a complex and not understandable program.

But if we declare two array each having fifty elements one for roll numbers and another for marks. Now, we have only two names to remember. Elements of these array will be referred to as Arrayname[n], where n is the element number in the array. Therefore arrays are very much useful in a case where quit many elements of the same data types need to be stored and processed.

The element number s in parenthesis are called subscripts or index. The subscript or index of an element designates its position in the array's ordering.

TYPES OF ARRAY :-

Arrays are of two types:

1. One-dimensional Arrays

2. Multi-dimensional Arrays

One –dimensional Arrays:- The simplest form of an array is one - dimensional. The array itself is given a name and its elements are referred to by their subscripts. In C++, an array must be defined before it can be used to store information. The general form of an array declaration is:

Data_type Array_name[size];

For example the following statement declares an array marks of int data type , which have 50 elements marks[0] to marks[49].

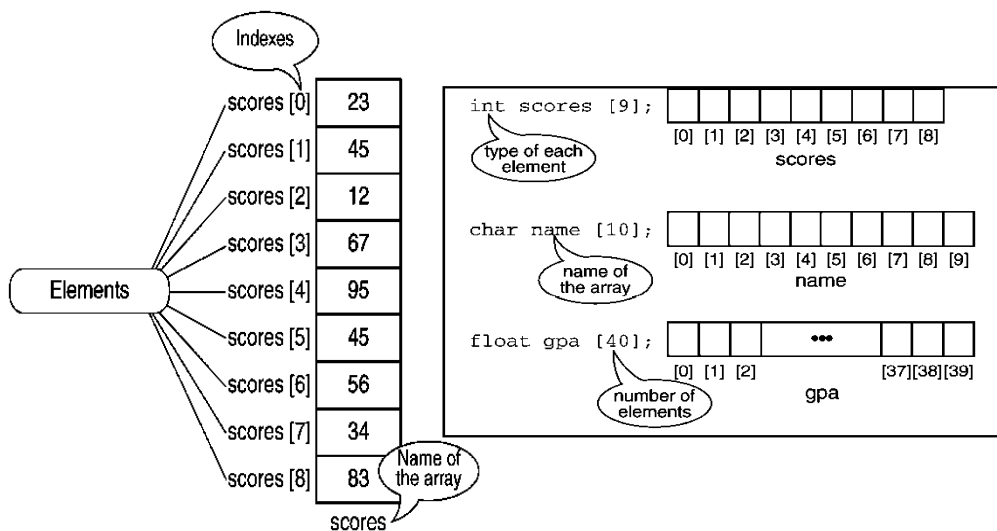
int marks[50] ;

Referencing Array Elements:- We can reference array elements by using the array's name & subscript (or index). The first element has a subscript of 0. The last element in an array of length n has a subscript of n-1. If we declare an array:

int scores[9];

then its 10 element will be referred as:

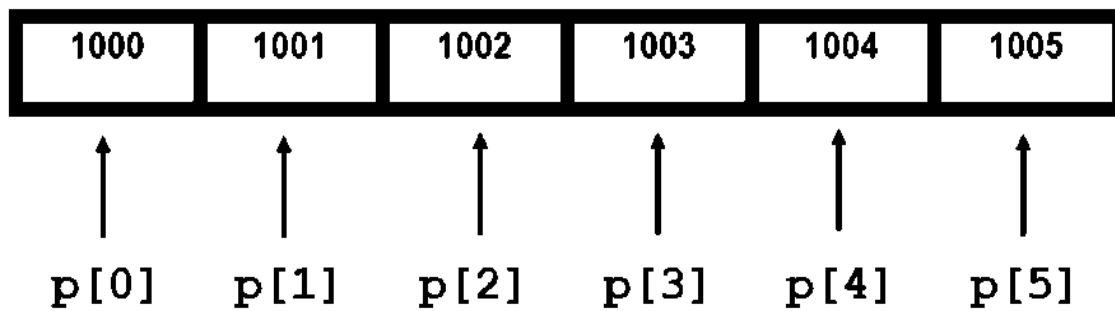
scores[0], scores[1],, scores[8]



Memory Representation of Single Dimensional Arrays:- Elements of single dimensional array are stored in contiguous memory location in their index order. For example , an array p of type char with 6 elements declared as:

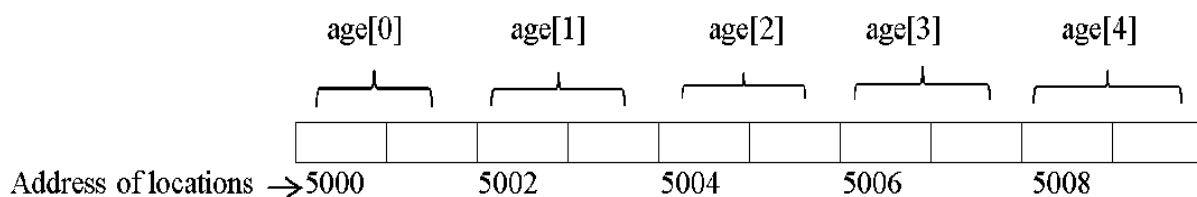
char p[6];

will be represented in memory as shown below:

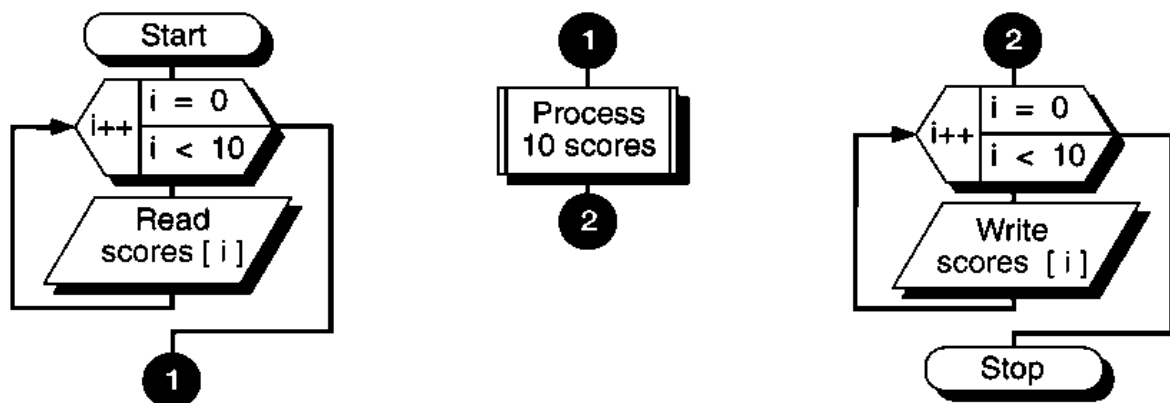


If we declare an int array age with 5 elements as:

int age[5];



Accepting Data in Array from User (Inputting Array elements) and Printing Array elements:- Since we can refer to individual array elements using numbered indexes, it is very common for programmers to use for (or any) loops when processing arrays.



General form of for loop for Reading elements of array (1-D)	Generally processing part may be include with in the loop of reading or printing, otherwise a same type separate loop may be used for processing	General form of for loop for printing elements of array (1-D)
<pre>for (int i=0; i< size; i++) { cout<<"Enter Array Element "<<i+1; cin>>Array_Name[i]; }</pre>		<pre>for (int i=0; i< size; i++) { cout<<Array_Name[i]<< " , "; }</pre>

1. Initializing arrays.

When we declare an array, we have the possibility to assign initial values to each one of its elements by enclosing the values in braces { }. Syntax for initialization of one-dimensional array is:

Data_type Array_name[size] = { value list } ;

For example:

int num [5] = { 16, 2, 77, 40, 12071 };

This declaration would have created an array like this:

	0	1	2	3	4
num	16	2	77	40	12071

The amount of values between braces { } must not be larger than the number of elements that we declare for the array between square brackets [].

C++ allows us to skip the size of the array in an array initialization statement (by leaving the square brackets empty []). In this case, the compiler will assume a size for the array equal to the number of values given between braces { }:

int num [] = { 16, 2, 77, 40, 12071 };

After this declaration, the size of array num will be 5, since we have provided 5 initialization values.

Example: Programme for one-dimentional array:

```
// arrays example
#include <iostream>
using namespace std;
```

```
int billy [] = {16, 2, 77, 40, 12071};
int n, result=0;
int main ()
{
    for ( n=0 ; n<5 ; n++ )
    {
        result += billy[n];
    }
    cout << result;
    return 0;
}
```

Screen output:

12206

String as an array:- C++ does not have a string data type rather it implements string as single dimensional character array. A string is defined as a character array terminated by a null character '\0'. Hence to declare an array strg that holds a 10 character string, you would write :

```
char strg[11];
```

Example program:

```
#include <iostream>
using namespace std;
int main ()
{
    char question[] = "Please, enter your first
name: ";
    char greeting[] = "Hello, ";
    char yourname [80];
    cout << question;
    cin >> yourname;
    cout << greeting << yourname << "!";
    return 0;
}
```

Screen output:

Please, enter your first name: John

Hello, John!

Two dimensional array : A two dimensional array is a continuous memory location holding similar type of data arranged in row and column format (like a matrix structure).

Declaration of 2-D array:- The general syntax used for 2-D array declaration is:

Data_type Array_name [R][C] ;

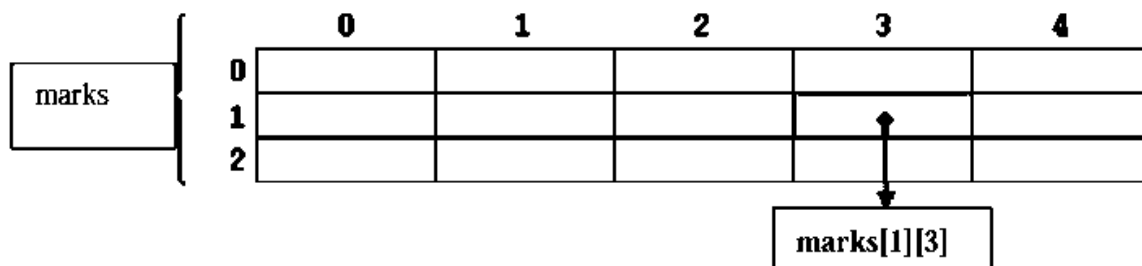
Where R represent number of rows and C represent number of columns in array.

For example,

int marks [3][5];

and, for example, the way to reference the second element vertically and fourth horizontally in an expression would be:

marks[1][3]



Initialization of 2-D Array:- Two dimensional arrays are also initialize in the same way as single dimensional array . e. g.,

int cube[5][2] = {1,1, 2,8,3,27,4,64,4,125 };

will initialize cube[0][0]=1, cube[0][1]=1, cube[1][0]=2, cube[1][1]=8 , cube[2][0]=3 and so on. 2-D array can also be initialized in unsized manner. However the first index can be skiped , the second index must be given. e.g.,

int cube[][2] = {1,1, 2,8,3,27,4,64,4,125 };

is also valid.

Accepting Data in 2-D Array from User (Inputting 2-D Array elements) and Printing 2-D Array elements:- Since We must reference both, a row number and a

column number to input or output values in a two-dimensional array. So, we use a nested loop (generally nested for loop) , when processing 2-D arrays.

General form of for loop for Reading elements of 2-D array	Generally processing part may be include within the loop of reading or printing, otherwise a same type separate nested loop may be used for processing	General form of for loop for printing elements of 2-D array
<pre>for (int i=0; i< R; i++) { cout<<"Enter Row "<<i+1; for (int j=0; j<C ; j++) cin>>Array_Name[i][j]; }</pre>		<pre>for (int i=0; i< R; i++) { for (int j=0; j<C ; j++) cout<<Array_Name[i][j] <<'t'; cout<<'n'; }</pre>

Where R represent number of rows and C represent number of columns in array.

Array of Strings:- An array of string is a two-dimensional character array. The size of first index determines the number of strings and size of second index determines maximum length of each string. The following statement declares an array of 10 strings , each of which can hold maximum 50 valid characters.

```
char string[10][51] ;
```

Notice that the second index has been given 51, because 1 extra size is given for store null character ‘\0’.

Example Program:

```
#define WIDTH 5
#define HEIGHT 3
int jimmy [HEIGHT][WIDTH];
int n,m;
int main ()
{
    for (n=0;n<HEIGHT;n++)
    for (m=0;m<WIDTH;m++)
    {
        jimmy[n][m]=(n+1)*(m+1);
    }
    return 0;
}
```

USER DEFINED DATA TYPES:-

The C++ language allows you to create and use data types other than the fundamental data types. These types are called user-defined data types.

STRUCTURES:-

In C++, a structure is a collection of variables that are referenced by a single name. The variable can be of different data types. They provide a convenient means of keeping related information together.

Defining Structure :-

A Structure is defined by using the Keyword struct. The General Syntax for defining a Structure is :

Syntax:

```
struct< Name of Structure >
{
    <datatype>< data-member 1>;
    <datatype>< data-member 2>;
    <datatype>< data-member 3>;
    ...
    ...
    <datatype>< data-member n>;
};
```

Example:

A Proper example following the previous syntax could be:

```
struct WORKER
{
    char name[45];
    char gender;
    int age;
    float rate;
};
```

Structure name

data-member

Declaring Structure Variable :-

This is similar to variable declaration. We can declare the variable of structure type using two different ways either with structure definition or after structure definition.

The following syntax shows the declaration of structure type variables with structure definition:

```
struct< Name of Structure >
```

```
{
    <datatype>< data-member 1>;
    <datatype>< data-member 2>;
    <datatype>< data-member 3>;
    ...
    ...
    <datatype>< data-member n>;
} var1, var2,....., varn ;
```

We can declare the structure type variables separately (after defining of structure) using following syntax:

Structure_name var1, var2,, var_n;

Accessing Structure Elements :- To access structure element , dot operator is used. It is denoted by (.). The general form of accessing structure element is:

Structure_Variable_Name.element_name

Initializing of Structure elements:- You can initialize members of a structure in two ways. You can initialize members when you declare a structure or you can initialize a structure with in the body of program. For Example:

First Method:-

```
#include <iostream.h>
#include <conio.h>
void main()
{
    // Declaring structure at here
    struct Employee
    {
        int empcode;
        float empsalary;
    };
    Employee emp1 = {100, 8980.00} ;    // emp1 is the structure variable ,
                                        // which is also initialize with declaration

    clrscr();
    int i;                             // declares a temporary variable for print a line
                                        // Printing the structure variable emp1 information to the
                                        screen
    cout<< "Here is the employee information : \n";
```

```
for (i = 1; i <= 32; i++)
    cout<< "=";
cout<< "\nEmployee code : " << emp1.empcode;
cout<< "\nEmployee salary : " << emp1.empsalary;
}
```

Second Method:-

```
#include <iostream.h>
#include <conio.h>
void main()
{
    // Declaring structure here

    struct Employee
    {
        int empcode;
        float empsalary;
    } emp1; // emp1 is the structure variable
    clrscr();
    int i; // declares a temporary variable for print a line
           // Initialize members here

    emp1.empcode = 100;
    emp1.empsalary = 8980.00;
           // Printing the structure variable emp1 information to the
           screen
    cout<< "Here is the employee information : \n";
    for (i = 1; i <= 32; i++)
        cout<< "=";
    cout<< "\nEmployee code : " << emp1.empcode;
    cout<< "\nEmployee salary : " << emp1.empsalary;
}
```

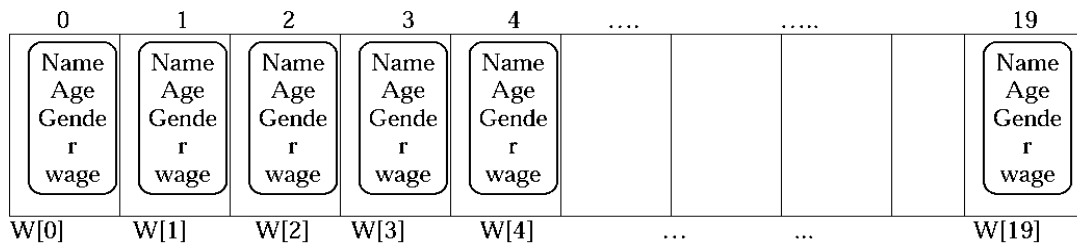
Structure variable assignments :-

We know that every variable in C++ can be assigned any other variable of same data type i.e :

if int a = 7 ; b = 3;

we can write :

The above structure could be visualized as:



Each of the elements of the array is itself a structure hence each of them have all the four components.

Function with Structure variable:-

We can also pass a structure variable as function parameter / arguments. The benefit is that the structure carries a bundled information to the structure. The prototype of such a function would be

<return_type> function_name(<structure_name><var> , ... , ...);

Let us understand the concept with following function, which increases the wage of a female worker by passed percent

```
void increaseWage( Worker & w , float incr )  
{  
    if( w.gender == 'F' || w.gender == 'f' )  
    {  
        w.wage += w.wage* (incr /100) ;  
    }  
}
```

Look at the highlighted parameter, we have passed formal structure variable as reference, so that the increment is reflected back in actual parameter.

Similarly, if we don't want to pass our structure variable as reference we can do so, but then we have to return the modified structure variable back. This can be achieved in above function, if we take return type as structure name. Let us modify the function so that it returns a structure :

```
Worker increaseWage( Worker w , float incr)  
{  
    if( w.gender == 'F' || w.gender == 'f' )  
    {  
        w.wage += w.wage* (incr /100) ;  
    }  
}
```

```
    }  
    return w ;  
}
```

Nested structure :- A Structure within another structure is known as nested structure. A structure containing an another structure as valid element is known as nested structure. e.g.

```
struct address  
{  
    int houseno;  
    char city[20];  
    char area[20];  
    long int pincode;  
};
```

```
struct employee  
{  
    int empno;  
    char name[30];  
    char design[20];  
    char department[20];  
    address ad;                // nested structure  
}
```

Declaration:

employee e;

Input /Output :

```
cin>>e.ad.houseno;    // members are accessed using dot(.) operator.  
cout<<e.ad.houseno;
```

typedef :- The typedef keyword allows to create a new names for data types. The syntax is:

typedef existing_data_type new_name ; e.g.,

```
typedef int num;
```

#define Preprocessor Directive:-The # define directive create symbolic constant, constants that are represent as symbols and macros (operation define as symbols). The syntax is:

#define identifier replacement_text ;

When this line appears in a file, all subsequent occurrences of identifier in that file will be replaced by replacement_text automatically before the program is compiled. e.g.,

Program: Program to calculate area of a circle by using #define preprocessor directive.

```
#include <iostream.h>
#define PI 3.14159
#define CIRCLE_AREA(x) (PI * (x) * (x))
void main()
{
    float area;
    int r;
    cout<< "Enter the radius : ";
    cin>> r;
    area = CIRCLE_AREA(r);
    cout<< "Area is : " << area;
}
```

Book References

1. C++ By E Balagurusamy
2. IGNOU Study Material
3. MCGRAW Abraham Silberschatz, Henry F. Korth, S. Sudarshan
4. Ronald J. Tocci Neal S. Widmer

QR Code References

1. E- Content developed by Sajad Akbar Rather (I/C ICT DIET SOPORE)
2. E- Content developed by Javeed Hassan Sofi (Tr. GMS Brar Bandipora)
3. www.easytuts4you.com
4. SuccessCDs (<http://youtu.be/Kelm63Q-LO8>)
5. Knowledge Gate (<http://youtu.be/C-CzmtqMVfo/>)
6. <http://youtu.be/SJXTjSqjcLg>
