

COURSE SYLLABUS

Problem Solving using Computers (CMPC101)			
Unit No. & Name	Details	Counseling Sessions	Weightage
Unit 1 Introduction to Computer	<ul style="list-style-type: none"> • Computer Fundamentals: Introduction • Characteristics of Computers, • Uses of computers, • Types and generations of Computers. • Basic Computer Organization: Units of a computer, CPU, ALU, memory hierarchy, registers, I/O devices. 	4	
Unit 2 Techniques of Problem Solving:	<ul style="list-style-type: none"> • Concept of problem solving, Problem definition, Program design • Flowcharting, decision table, algorithms, Structured programming concepts 	4	
Unit 3 Planning the Computer Program	<ul style="list-style-type: none"> • Programming methodologies viz. top-down and bottom-up programming • Debugging, Types of errors in programming, Documentation 	2	
Unit 4 Introduction to C	<ul style="list-style-type: none"> • History of C • C Basics • C character set, tokens, constants, variables, keywords, identifiers • C operators- arithmetic, logical, assignment, relational, increment and decrement, conditional, bit wise, special, operator precedence, C expressions datatypes. • Problem solving techniques: flowchart and algorithm • 	5	
Unit 5 Managing Input and Output Operations	<ul style="list-style-type: none"> • Reading a Character ,Writing a Character , Formatted input , Formatted Output 	5	
Unit 6 Decision Making and looping	<ul style="list-style-type: none"> • Decision making and branching if-statement–if, if- else, else-if ladder, nested if else, switch case statement, break statement • Decision making and looping-while, do, do-while statement, for loop, continue statement 	5	

Unit 7 Arrays	<ul style="list-style-type: none"> Arrays Declaration and initialization of one dimensional, two Dimensional and character arrays, accessing array elements. 	5	
Unit 8 Strings	<ul style="list-style-type: none"> Declaration and initialization of string variables, string handling functions from standard library – strlen(), strcpy(), strcat(), strcmp() 	5	
Unit 9 Functions	<ul style="list-style-type: none"> Need of functions, scope and lifetime of variables, defining functions, function call, call by value, call by reference, return values, storage classes. category of function-No argument No return value, No argument with return value, argument with return value, recursion, command line Arguments 	5	
Unit 10 Pointers	<ul style="list-style-type: none"> Understanding pointers, declaring pointer variable, initialization of pointer variable, accessing address of a variable, pointer expressions, Pointers arithmetic 	4	
Unit 11 Structures and Unions	<ul style="list-style-type: none"> Structures: - Defining structure, declaring and accessing structure members, initialization of structure, arrays of structure, Difference between Structure and Union 	4	5
Unit 12 File Handling	<ul style="list-style-type: none"> Introduction to Streams Types of Files Operation on text files Standard Library Input Output Function 	4	
Unit 13 Storage Classes	<ul style="list-style-type: none"> Scope and Extent Storage Classes in a Single source file Auto, extern, static and register 	4	
Unit 14 The C Preprocessor	<ul style="list-style-type: none"> Features of C Preprocessor Macro Expansion Macros with Arguments Macros versus Functions File Inclusion Conditional Compilation 	4	

UNIT 1: INTRODUCTION TO COMPUTERS

UNIT OBJECTIVES

After successful completion of this unit, you will be able

- ❖ To know what is a computer.
- ❖ To list the characteristics of a computer.
- ❖ To list the uses of computers in various fields.
- ❖ To classify computers on the basis of their use.
- ❖ To classify computers on the basis of size.
- ❖ To explain the generations of computers.
- ❖ To explain the basic organization of a computer.

INTRODUCTION TO COMPUTERS:

A computer is an electronic device that can be instructed to carry out sequence of arithmetic or logical operations automatically.

The ability of computers to follow generalized sets of operations, called programs, enables them to perform an extremely wide range of tasks.

Such computers are used as control systems for a very wide variety of industrial and home devices. This includes simple special purpose devices like microwave ovens and remote controls, factory devices such as industrial robots and computer assisted design, but also in general purpose devices like personal computers and mobile devices such as smartphones.

Thus, **computer** is an electronic device that receives input, stores or processes the input as per user instructions and provides output in a desired format.

CHARACTERISTICS OF COMPUTERS:

Following are the characteristics of a computer:

1. **SPEED:** In general, no human being can compete to solving the complex computation, faster than computer. It works at a high speed.
2. **ACCURACY:** Since a computer is programmed, so whatever input we give to it, it gives us the result with accuracy. It is 100 % accurate.
3. **STORAGE:** Computer can store huge data in appropriate format.
4. **DILIGENCE:** Computer can work for hours without any break and without any errors. It does not get tired.
5. **VERSATILITY:** We can use computer to perform various types of jobs. It can be used in all fields. No field today is untouched by computers.
6. **INTELLIGENCE:** Computer does not have its own intelligence. Computer does not work on its own. It requires a set of instructions for performing a task. Advance technology now enables us to add artificial intelligence to a computer.
7. **EMOTIONS & FEELING:** Computer does not have emotions and feelings of its own, but advance technology now enables us to add artificial emotions and feelings to a computer



Figure 1.1 Characteristics of Computers

USES OF COMPUTERS:

Computers have become an essential part of modern human life. Since the invention of computer they have evolved in terms of increased computing power and decreased size. Owing to the widespread use of computers in every sphere, life in today's world would be unimaginable without computers. They have made human lives better and happier. There are many computer uses in different fields of work. Engineers, architects, jewelers, and filmmakers all use computers to design things. Teachers, writers, and most office workers use computers for research, word processing and emailing. Small businesses can use computers as a point of sale and for general record keeping.

1. Education:

Computers have its dominant use in the education field which can significantly enhance performance in learning. Even distance learning is made productive and effective through internet and video-based classes. Researchers have massive usage of these computers in their work from the start to the end of their scholarly work.

2. Health and Medicine:

Most of the medical information can now be digitized from the prescription to reports. Computation in the field of medicine allows us to offer varied miraculous therapies to the patients. ECG's, radiotherapy wasn't possible without computers.

3. Financial Institutions:

We know well that computers are being used by the financial institutions like banks for different purposes. The foremost important thing is to store information about different account holders in a database to be available at any time. Keeping the records of the cash flow, giving the information regarding your account.

4. Entertainment:

Computers are now the major entertainers and the primary pass time machines. We can use computers for playing games, watching movies, listening to music, drawing pictures.

5. Transport System:

With internet on computers we can know the details of the buses or trains or the flight available to our desired destination. The timings and even the updates on the delay can also be known through these computers. We can book our tickets online. Staff of the transport system will keep a track of the passengers, trains or flight details, departure and arrival timings by using computers.

6. Businesses:

Shared information can be recorded by using computer. Electronic mail is the revolutionary service offered by the computer. Video Conferencing is also another major advantage. With the help of electronic communication and video conferencing official meetings, deals and the issues raised (if any) can be handled online. We use email system to exchange the information. It has wide uses in marketing, stock exchanges and bank. Even the departmental stores can't run effectively without computer.

7. E-Commerce:

Electronic shopping through online shopping added favor to purchaser and merchants. Electronic banking is now at your hand where every bank has online support for transaction of monetary issues. You can easily transfer your money anywhere even from your home.

8. Security and Defense:

Computers are the main tools which help in developing missiles and other equipment in the defense system. Designing and the maintenance are possible only through computers. Computer builds the links between the soldiers and commanders through the satellite. Construction of weapons and controlling their function is not possible without the aid of computers. The list of the criminals and the records of the cops are maintained regularly in the system. For air defense tracking, positioning and targeting the system through global Positioning system and drones is very efficient. This is due to the use of computer hardware and software.

9. Computer as Designer:

As per the title, computers aid in designing buildings, magazines, prints, newspapers, books and many others. The construction layouts are designed beautifully on system using different tools and software. Computers have aided in the design of safety equipment in sports such as football helmets, shoes, mouth guards etc. Today AI is becoming part and parcel for the designing work.



Figure 1.2 Uses of Computers

TYPES OF COMPUTERS:

Historically computers were classified according to processor types because development in processor and processing speeds were the developmental benchmarks. Earliest computers used vacuum tubes for processing, were huge and broke down frequently. However, as vacuum tubes were replaced by transistors and then chips, their size decreased and processing speeds increased manifold.

All modern computers and computing devices use microprocessors whose speeds and storage capacities are skyrocketing day by day. The developmental benchmark for computers is now their size. Computers are now classified on the basis of their use or size –

- Desktop
- Laptop
- Tablet
- Server
- Mainframe
- Supercomputer

GENERATIONS OF COMPUTERS

First Generation of Computers (1942-1955)

The beginning of commercial computer age is from UNIVAC (Universal Automatic Computer). It was developed by two scientists Mauchly and Eckert at the Census Department of United States in 1947. *The first generation computers were used during 1942-1955.* They were based on **vacuum tubes**. Examples of first generation computers are ENIVAC and UNIVAC-1.

Advantages

- Vacuum tubes were the only electronic component available during those days.
- Vacuum tube technology made possible to make electronic digital computers.
- These computers could calculate data in millisecond.

Disadvantages

- The computers were very large in size.
- They consumed a large amount of energy.
- They heated very soon due to thousands of vacuum tubes.
- They were not very reliable.
- Air conditioning was required.
- Constant maintenance was required.
- Non-portable.

Second Generation Computers (1955-1964)

The second generation computers used **transistors**. The scientists at Bell laboratories developed transistor in 1947. These scientists include John Barden, William Brattain and William Shockley. The size of the computers was decreased by replacing vacuum tubes with transistors. The examples of second generation computers are IBM 7094 series, IBM 1400 series and CDC 164 etc.

Advantages

- Smaller in size as compared to the first generation computers.

- The 2nd generation computers were more reliable.
- Used less energy and were not heated.
- Wider commercial use.
- Better portability as compared to the first generation computers.
- Better speed and could calculate data in microseconds.
- Used faster peripherals like tape drives, magnetic disks, printer etc.
- Used Assembly language instead of Machine language.
- Accuracy improved.

Disadvantages

- Cooling system was required.
- Constant maintenance was required.
- Commercial production was difficult.
- Only used for specific purposes.
- Costly and not versatile.
- Punch cards were used for input.

Third Generation Computers (1964-1975)

The Third generation computers used the **integrated circuits (IC)**. Jack Kelby developed the concept of integrated circuit in 1958. It was an important invention in the computer field. The first IC was invented and used in 1961. The size of an IC is about ¼ square inch. A single IC chip may contain thousands of transistors. The computer became smaller in size, faster, more reliable and less expensive. The examples of third generation computers are IBM 370, IBM System/360, UNIVAC 1108 and UNIVAC AC 9000 etc.

Advantages

- Smaller in size as compared to previous generations.
- More reliable.
- Used less energy.
- Produced less heat as compared to the previous two generations of computers.
- Better speed and could calculate data in nanoseconds.
- Used fan for heat discharge to prevent damage.
- Maintenance cost was low because hardware failure is rare.
- Totally general purpose.
- Could be used for high-level languages.
- Good storage.
- Versatile to an extent.
- Less expensive.
- Better accuracy.
- Commercial production increased.
- Used mouse and keyboard for input.

Disadvantages

- Air conditioning was required.
- Highly sophisticated technology required for the manufacturing of IC chips.

Fourth Generation Computers (1975-Present)

The fourth generation computers started with the invention of Microprocessor. The Microprocessor contains thousands of ICs. Ted Hoff produced the first microprocessor in 1971 for Intel. It was known as Intel 4004. The technology of integrated circuits improved rapidly. The LSI (Large Scale Integration) circuit and **VLSI (Very Large Scale Integration)** circuit was designed. It greatly reduced the size of computer. The size of modern Microprocessors is usually one square inch. It can contain millions of electronic circuits. The examples of fourth generation computers are Apple Macintosh & IBM PC.

Advantages

- More powerful and reliable than previous generations.
- Small in size.
- Fast processing power with less power consumption.
- Fan for heat discharging and thus to keep cold.
- No air conditioning required.
- Totally general purpose.
- Commercial production.
- Less need for repair.
- Cheapest among all generations.
- All types of High level languages can be used in this type of computers.

Disadvantages

The latest technology is required for manufacturing of Microprocessors.

Fifth Generation Computers (Present & Beyond)

Scientists are working hard on the **5th generation computers** with quite a few breakthroughs. It is based on the technique of **Artificial Intelligence (AI)**. Computers can understand spoken words & imitate human reasoning. They can respond to its surroundings using different types of sensors. Scientists are constantly working to increase the processing power of computers. They are trying to create a computer with real IQ with the help of advanced programming and technologies. **IBM Watson** computer is one example that outsmarts **Harvard University Students**. The advancement in modern technologies will revolutionize the computer in future.

Generation of Computers	From - To	Major Characteristics
First	1942-1955	Vacuum tubes.
Second	1955-1964	Transistors.
Third	1964-1975	Integrated Circuits.
Fourth	1975-Present	VLSI (Very Large Scale Integration) circuit
Fifth	Present & Beyond	Artificial Intelligence (AI)

Figure1.3: Generations of computers

BASIC COMPUTER ORGANIZATION:

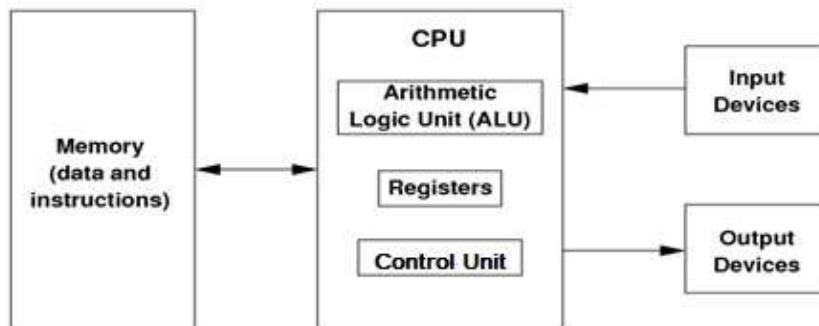


Figure 1.4 : Basic Computer Organization

The given below explanation refers to the above figure 1.4.

- **Input:** This is the process of entering data and programs in to the computer system. You should know that computer is an electronic machine like any other machine which takes raw data as input and performs some processing giving out processed data. Therefore, the input unit takes data from us to the computer in an organized manner for processing.
- **Storage:** The process of saving data and instructions permanently is known as storage. Data has to be fed into the system before the actual processing starts. It is because the processing speed of Central Processing Unit (CPU) is so fast that the data has to be provided to CPU with the same speed. Therefore the data is first stored in the storage unit for faster access and processing. This storage unit or the primary storage of the computer system is designed to do the above functionality. It provides space for storing data and instructions.

The storage unit performs the following major functions:

- All data and instructions are stored here before and after processing.
- Intermediate results of processing are also stored here.
- **Processing:** The task of performing operations like arithmetic and logical operations is called processing. The Central Processing Unit (CPU) takes data and instructions from the storage unit and makes all sorts of calculations based on the instructions given and the type of data provided. It is then sent back to the storage unit.
- **Output:** This is the process of producing results from the data for getting useful information. Similarly the output produced by the computer after processing must also be kept somewhere inside the computer before being given to you in human readable form. Again the output is also stored inside the computer for further processing.
- **Control:** The manner how instructions are executed and the above operations are performed. Controlling of all operations like input, processing and output are performed by control unit. It takes care of step by step processing of all operations inside the computer.

FUNCTIONAL UNITS:

In order to carry out the operations mentioned in the previous section the computer allocates the task between its various functional units. The computer system is divided into three separate units for its operation. They are

1) Arithmetic logical unit

2) Control unit.

3) Central processing unit.

Arithmetic Logical Unit (ALU) : After you enter data through the **input device** it is stored in the primary storage unit. The actual processing of the data and instruction are performed by Arithmetic Logical Unit. The major operations performed by the ALU are addition, subtraction, multiplication, division, logic and comparison. Data is transferred to ALU from storage unit when required. After processing the output is returned back to storage unit for further processing or getting stored.

Control Unit (CU): The next component of computer is the Control Unit, which acts like the supervisor seeing that things are done in proper fashion. Control Unit is responsible for coordinating various operations using time signal. The control unit determines the sequence in which computer programs and instructions are executed. Things like processing of programs stored in the main **memory**, interpretation of the instructions and issuing of signals for other units of the computer to execute them. It also acts as a switch board operator when several users access the computer simultaneously. Thereby it coordinates the activities of computer's peripheral equipment as they perform the input and output.

Central Processing Unit (CPU): The ALU and the CU of a computer system are jointly known as the central processing unit. You may call CPU as the brain of any computer system. It is just like brain that takes all major decisions, makes all sorts of calculations and directs different parts of the computer functions by activating and controlling the operations.

MEMORY HIERARCHY:

The figure 1.5 displays the Memory Hierachy in a diagrammatic way.

1. Registers Memory

- CPU registers are at the top most level of this hierarchy, they hold the most frequently used data. They are very limited in number and are the **fastest**.
- They are often used by the CPU and the ALU for performing arithmetic and logical operations, for temporary storage of data.

2. Cache memory

- The very next level consists of small, fast **cache memories** near the CPU. They act as Staging areas for a subset of the data and instructions stored in the relatively slow **main memory**.
- There are often two or more levels of cache as well. The cache at the top most level after the registers is the primary cache. Others are secondary caches.
- Many a times there is cache present on board with the CPU along with other levels that are outside the chip.

3. Main Memory:

- The next level is the main memory; it stores data on large, slow disks often called hard disks. These hard disks are also called secondary memory, which are the last level in the hierarchy. The main memory is also called primary memory.
- The secondary memory often serves as staging areas for data stored on the disks or tapes of other machines connected by networks.

4. Magnetic disk:

An information storage device for a computer in the shape of a round flat plate which can be rotated to give access to all parts of the surface. The data may be stored either magnetically (in a *magnetic disk*) or optically (in an *optical disk* such as a CD-ROM).

5. Magnetic tape:

Magnetic tape is used in recording sound, pictures, or computer data.

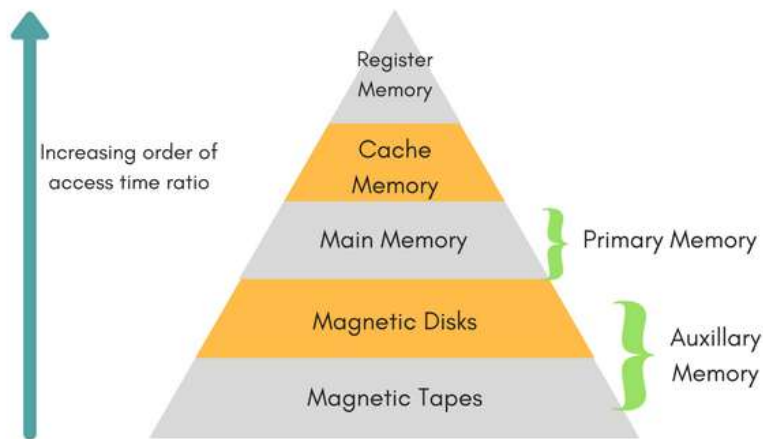


Figure 1.5: Memory Hierarchy

REGISTERS

Registers are a type of computer memory used to quickly accept, store, and transfer data and instructions that are being used immediately by the **CPU**. Register is a group of flip flops with each flip flop capable of storing one bit of information. An 'n' bit register has a group of n flip flops which is capable of storing n bit information. There are various types of **Registers** those are used for various purpose. Among of the some mostly used Registers are AC or Accumulator, Data Register or DR, the AR or Address Register, program counter (**PC**), Memory Data Register (MDR), Index **register**, *Memory Buffer Register*.

Types of Registers are as follows:

MAR stand for Memory Address Register

This register holds the **memory** addresses of data and instructions. This register is used to access data and instructions from memory during the execution phase of an instruction. Suppose CPU wants to store some data in the memory or to read the data from the memory. It places the address of the-required memory location in the MAR.

Program Counter

The program counter (PC), commonly called the instruction pointer (IP) in Intel x86 microprocessors, and sometimes called the instruction address register, or just part of the instruction sequencer in some **computers**, is a processor register.

It is a 16 bit special function register in the 8085 microprocessor. It keeps track of the next memory address of the instruction that is to be executed once the execution of the current instruction is completed. In other words, it holds the address of the memory location of the next instruction when the current instruction is executed by the microprocessor.

Accumulator Register

This register is used for storing the results, those which are produced by the system..

Memory Data Register (MDR)

MDR is the register of a **computer's** control unit that contains the data to be stored in the computer storage (e.g. **RAM**), or the data after a fetch from the computer storage. It acts like a buffer and holds anything that is copied from the memory ready for the processor to use it. MDR holds the **information** before it goes to the decoder.

MDR which contains the data to be written into or readout of the addressed location. For example, to retrieve the contents of cell 123, we would load the value 123 into the MAR and perform a fetch operation. When the operation is done, a copy of the contents of cell 123 would be in the MDR. To store the value 98 into cell 4, we load a 4 into

the MAR and a 98 into the MDR and perform a store. When the operation is completed the contents of cell 4 will have been set to 98, by discarding whatever was there previously.

Index Register

A hardware element which holds a number that can be added to (or, in some cases, subtracted from) the address portion of a computer instruction to form an effective address. It is also known as base register. An index register in a computer's CPU is a processor register used for modifying operand addresses during the run of a program.

Memory Buffer Register

MBR stands for *Memory Buffer Register*. This register holds the contents of data or instruction read from, or written in memory. It means that this register is used to store data/instruction coming from the memory or going to the memory.

Data Register

A register used in microcomputers to temporarily store data being transmitted to or from a peripheral device.

INPUT/OUTPUT DEVICES:

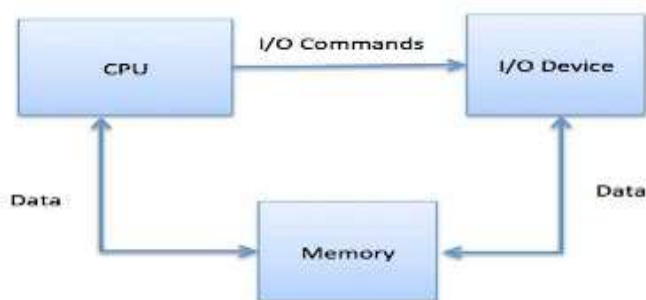


Figure 1.6 : Working of I/O Devices

The devices which are used to input the data and the programs in the **computer** are known as "**Input Devices**". Or Input device can read data and convert them to a form that a computer can use. **Output Device** can produce the final product of machine processing into a form usable by humans. It provides man to machine communication. Some of the I/O devices are explained below.

(1) Keyboard: Keyboard is used in the input phase of a computer-based **information** system. Keyboard is most common **input device** is used today. The data and instructions are input by typing on the keyboard. The message typed on the keyboard reaches the **memory** unit of a computer. It is connected to a computer via a cable. Apart from alphabet and numeral keys, it has other function keys for performing different functions.



(2) Mouse: It is a pointing device. The mouse is rolled over the mouse pad, which in turn controls the movement of the cursor in the screen. We can click, double click or drag the mouse. Most of the mice have a ball beneath them, which rotates when the mouse is moved. The ball has 2 wheels at the sides, which in turn sense the movement of the ball. The sensor notifies the speed of its movements to the computer, which in turn moves the cursor/pointer on the screen.



(3) **Scanner:** Scanners are used to enter information directly in to the **computers** memory. This device works like a photocopier machine. The scanner converts any type of printed or written information including photographs into digital pulses, which can be manipulated by the computer.



(4) **Track Ball:** Track ball is similar to the upside- down design of the mouse. The user moves the ball directly, while the device itself remains stationary. The user spins the ball in various directions to effect the screen movements.



(5) **Light Pen:** This is an input device which is used to draw lines or figures on a computer screen. It's touched to the CRT screen where it can detect raster on the screen as it passes.



(6) **Optical Character Reader:** It is' a device which detects alpha numeric characters printed or written on a paper. The text which is to be scanned is illuminated by a low frequency light source. The light is absorbed by the dark areas but reflected from the bright areas. The reflected light is received by the photocells.



(7) **Bar Code Reader:** This device reads bar codes and converts them into electric pulses to be processed by a computer. A bar code is nothing but data coded in form of light and dark bars.



(8) **Voice Input Systems:** These devices convert spoken words to machine language form. A microphone is used to convert human speech into electric signals. The signal pattern is then transmitted to a computer where it is compared with a dictionary of patterns that have been previously placed in a storage unit of computer. When a close match is found, the word is recognized.



(9) **Plotter:** Plotter is an O/P device that is used to produce graphical O/P on papers. It uses single color or multi-color pens to draw pictures as blue print etc.



(10) **Digital Camera:** It converts graphics directly into digital form. It looks like an ordinary camera, but no film is used therein, instead a CCD (Charge Coupled Device) Electronic chip is used. When light falls on the chip through the lens, it converts light waves into electrical waves.



SOLVED PROBLEMS AND SELF CHECK QUESTIONS

SOLVED PROBLEMS

- 1) Computer is a/an _____ device which accepts data, processes it and gives us information.
 - a) **electronic**
 - b) electrical
 - c) mechanical
 - d) optical

2) Second generation of computers used _____.

- a) **transistors**
- b) vacuum tubes
- c) integrated circuits
- d) all of the above

SELF-TEST 01

01 The first generation of computers is considered during the period ____

- a) 1942-1955
- b) 1955-1964
- c) 1964-1975
- d) 1947-1950

02 Which of the following are used to quickly accept, store and transfer data and instructions that are being used immediately by the CPU.

- a) Registers
- b) Capacitors
- c) Integrated circuit
- d) Registers and Capacitors

03 Which of the following device reads those zebra stripes and converts them into electric pulses to be processed by a computer.

- a) Bar Code reader
- b) Voice Input System
- c) Optical Character Reader
- d) Light Pen

04 Which of the following is at highest level of the Memory hierarchy?

- a) Register memory
- b) Main memory
- c) Cache Memory
- d) Magnetic Tape

05 Choose the correct option which means computer never gets tired.

- a) Diligence
- b) Versatility
- c) Accuracy
- d) Automaticity

SUMMARY :

In this chapter we studied the definition of a computer i.e. A Computer is an electronic device which accepts data , processes it and gives us information. Different characteristics of computers such as Speed, Accuracy, Storage Capacity, Reliability, Versatility etc were discussed. Computer application areas such as Education, Health and Medicine, Financial institutions, Transport systems and many others were discussed..

Five generations of computers were studied. First generation used vacuum tubes, Second generation used transistors, Third generation used Integrated Circuits , Fourth generation used VLSI technology. AI is being used in Fifth Generation.

Block diagram of Computer was elaborated with various types of registers and their uses. Memory Hierarchy showed that Register Memory is at the top most level. Finally different types of Input/Output devices were discussed.

EXERCISE

- Q1. Define a Computer and state its characteristics.
- Q2. Write the uses of Computers in various fields.
- Q3. Discuss the different generations of Computers.
- Q4. Draw a block diagram of a computer and explain it in details.
- Q5. Write a short note on Memory Hierarchy.
- Q6. Explain different types of registers.
- Q7. Explain any five input / output devices.

REFERENCE MATERIAL

REFERENCES

MOOCs

YOUTUBE VIDEOS

WIKIPEDIA

BLOGS

OER (SLIDES, PDFs, TUTORIALS,ETC)

REFERENCE BOOKS

<https://www.stanfordcomputeroptics.com/>

<http://www.javatpoint.com>

<https://www.studytonight.com>

<https://www.toppr.com>

ANSWERS TO SELF-CHECK QUESTIONS

1. (a) 2. (a) 3. (a) 4. (a) 5. (a)

UNIT 2: TECHNIQUES OF PROBLEM SOLVING

UNIT OBJECTIVES

After successful completion of this unit, you will be able

- ❖ To explain the concept of Problem Solving,
- ❖ To explain the steps involved in problem solving.
- ❖ To give the problem definition.
- ❖ To explain what is meant by program design.
- ❖ To explain the term flowchart and its symbols.
- ❖ To explain the term algorithm and the characteristics of an algorithm.
- ❖ To explain what is meant by structured programming. To explain the term decision table, its advantages and its disadvantages.
- ❖ To write algorithms, draw flowcharts and prepare decision tables
- ❖ To use structured programming concepts while programming.

CONCEPT OF PROBLEM SOLVING:

Problem-solving consists of using some methods, in an orderly manner, for finding solutions to problems. Problem solving is an art and there are no universal approaches to solving problems. However, we can master this art by learning the tools used for solving problems or by studying the problems solved by others to understand the approach, used by them and apply this to solve new problems.

Problem solving is not new; it is an integral part of learning. The famous Hungarian mathematician George Polya (1887-1985), published the book “How to Solve it?” in 1945 which describes methods of problem solving.

Problem Solving Steps:

- I. Understand the problem
- II. Devise a plan
- III. Carry out the plan
- IV. Look back

Let us understand each step in detail.

- 1. Understand the problem:** It is impossible to solve a problem, if you do not know what the problem is, what is the input? Is there enough information to solve the problem? etc.
- 2. Devise a plan:** How are you going to attack the problem? Find a strategy to solve the problem. The possible strategies or “trick” can be:

- I. Trial and error
- II. Draw a picture
- III. Look for a pattern
- IV. Make a list
- V. Solve a simpler problem

VI. Use a variable, write an equation, etc.

3. **Carry out the plan:** Execute or implement the plan. Most problems are not solved quickly and may require many attempts. If one strategy does not work, try a different one.
4. **Look back:** Examine the solution. Is the problem solved completely? Is there another way of solving the problem which may be easier? Identify what worked and what did not work. This will help in solving future problems.

PROBLEM DEFINITION:

Before a program is written for solving a problem, it is important to define the problem clearly. For most software projects, systems analysts approach system users to collect user requirements and define the problem that a system aims to solve. They typically look at the following issues:

- What is the problem?
- What is the input required for supporting the solution process?
- What is the expected output of the solution?
- How do people solve the problem currently?
- Can the problem or part of the problem be more effectively solved by a software solution?

For most programmers, they work in accordance with the design specification produced by the system analysts/designers. Design specification describes the program input, output and functionality of the programs of the system. Note that the program input may be entered by the user through any input devices or retrieved from data stores (in terms files or database) whereas program output may be stored, printed or displayed on screen, etc. In addition to verifying the correctness of program input, output and functionality, it is important to ensure that any planned user-computer interactions be approved by the users.

PROGRAM DESIGN:

Program design is the activity of progressing from a specification of some required program to a description of the program itself. Most phase models of the **software life cycle** recognize program design as one of the phases. The input to this phase is a specification of what the program is required to do. During this phase the design decisions are made as to how the program will meet these the project requirements, and the output of the phase is a description of the program in some form that provides a suitable basis for subsequent implementation.

Frequently the design phase is divided into two sub phases 1. *architectural design* 2. *Detailed design*.

The **architectural design** produces a description of the program at a gross level, it is normally given in terms of the major components of the program and their interrelationships, the main algorithms that these components employ, and the major data structures. The **detailed design** then refines the architectural design to the stage where actual implementation can begin.

FLOWCHART SYMBOLS:

A flowchart is a diagrammatic or pictorial representation of an algorithm.. It makes use of symbols which are connected among them to indicate the flow of information and processing. It will show the general outline of how to solve a problem or perform a task. It is prepared for better understanding of the algorithm. Basic symbols used for flowcharting are as follows:

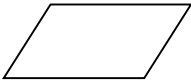
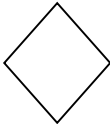
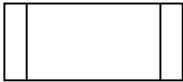
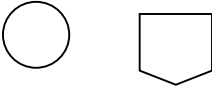
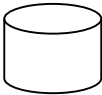
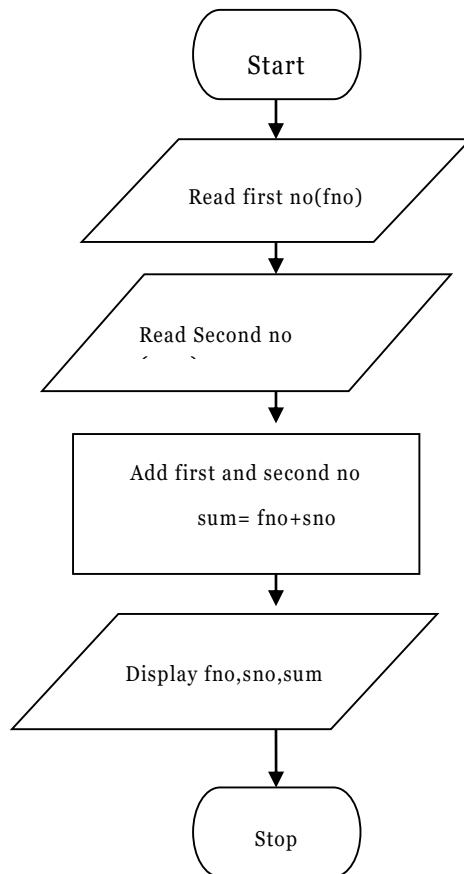
No	Symbol	Use
1		An Oval used to indicate beginning and end of an algorithm.
2		A parallelogram indicates input and output of an algorithm.
3		A rectangle indicates the processing and assignment operation.
4		A rhombus indicates a point where a decision is to be made.
5		A hexagon indicates the beginning of the repetition structure.
6		A double lined rectangle is used to point where a subprogram is used.
7		A arrow indicates the direction of flow of an algorithm
8		Connectors are used to connect the various parts of a flow chart. One is connector on the same page and another is used to connect on another page.
9		Represents document such as printout or report
10		To indicate database store.

Figure 2.1 : Flowchart Symbols

Draw a flowchart to add two numbers.



ALGORITHM:

An **algorithm** is a finite set of steps defining the solution of a particular problem. An algorithm is expressed in pseudo code-something resembling C language or Pascal, but with some statements in English rather than within the programming language. Developing an efficient algorithm requires a lot of practice and skill. It must be noted that an efficient algorithm is one which is capable of giving the solution to the problem by using minimum resources of the system such as memory and processor's time. Algorithm is language independent. It is thus a sequence of steps required to solve a problem written in simple English considering all alternatives. The word algorithm was derived from the name of the Persian mathematician al-Khwarizmi, Abu Ja'far Muhammad ibn Musa

Characteristics of Algorithm:

1. **Definiteness** – the steps must be precisely stated, they should be clear and unambiguous.
2. **Effectiveness**: every instruction must be such that it can be carried out by a person with pen and paper
3. **Finiteness** – the algorithm stops after a finite number of instructions are executed.
4. **Input** – the algorithm receives zero or more input.
5. **Output** – the algorithm produces at least one or more output.

STRUCTURED PROGRAMMING:

C is called a structured programming language because to solve a large problem, C programming language divides the problem into smaller modules called functions or procedures each of which handles a particular responsibility. The program which solves the entire problem is a collection of such functions. In structured programming, we divide the whole program into small modules, so that program becomes easy to understand.

WHY SHOULD WE FIRST WRITE AN ALGORITHM, THEN DRAW A FLOWCHART AND THEN WRITE A PROGRAM?

In computing, algorithms are essential because they serve as the systematic procedures that computers require. A good algorithm is like using the right tool in the workshop. It does the job with the right amount of effort. Using the wrong algorithm or one that is not clearly defined is like trying to cut a piece of plywood with a pair of scissors: although the job may get done, you have to wonder how effective you were in completing it.

A flowchart describes what operations (and in what sequence) are required to solve a given problem. A flowchart can be linked to the blueprint of a building. As we know a designer builder gets a blueprint drawn from an architect before starting the construction of a building. Similarly, a programmer prefers to draw a flowchart prior to writing a computer program.

Programming involves activities such as analysis, developing, understanding, generating **algorithms**, verification of requirements of algorithms including their correctness and consumption of resources. It is this of resources. It is this implementation of algorithms in a target **programming language**.

DECISION TABLES:

A decision table is a graphical method for explaining the logic of making decision in tabular format. It is a set of conditions and set of actions and different combinations of decisions.

"It is a matrix representation of logic of decisions which specify the possible conditions for decision and resulting actions."

The decision table is divided into two parts:

1. Condition

2. Action

1. Condition:

The condition part specifies all conditions that are applied to the inputted data. It is also divided into two parts-

i. Condition Stub:-

It describes the conditions that exist in the program logic.

ii. Condition Entry:-

It provides answers in the condition term.

2. Action:

The action part is subdivided into two parts ->

i. Action Stub:-

It describes the action to be taken to meet each condition.

ii. Action Entry:-

It describes the appropriate action resulting from the answer to the condition in condition entry.

Format:

Table Heading	Decision Rule
Condition Stub	Condition Entry
Action Stub	Action Entry

Advantages of Decision Table

1. It provides compact representation of decision making process.
2. It is easier to understand a particular path.
3. It can be changed according to situation.
4. These are best suited for calculating discounts, commissions or inventory control procedures.
5. The structure of decision table promotes a logically complete and consistent problem definition.

Disadvantages of Decision Table

1. It cannot express the complete sequence of operations to solve a problem therefore it may be difficult for the programmer to translate decision table into program.
2. If there are too many alternatives, it is difficult to list in decision table.
3. It does not show the flow of logic for the solution to a given problem.

An insurance company uses the following rule to determine the eligibility of a driver for insurance.

The driver will be insured if:-

1. The driver lives in the city with population less than 5000 and he is married man.
2. The driver lives in the city with population less than 5000 and he is married and age is over 30 years old.
3. The driver lives in the city with population is 5000 or more and is a married female.
4. The driver is male over 30.
5. The driver is married and under 30.

Condition	Rule1	Rule2	Rule3	Rule4	Rule5	Else
Lives in city population < 5000	Y	Y	N	-	-	-
Male	Y	-	N	Y	-	-
Married	Y	Y	Y	-	Y	-
Age < 30	-	Y	-	Y	N	-
Action	Rule1	Rule2	Rule3	Rule4	Rule5	Else
Insured	Y	Y	Y	Y	Y	
Uninsured						Y

SUMMARY :

In this Chapter we discussed the concept of Problem Solving which consists of using methods in an orderly manner for finding solutions to problems. Before a program is written for solving a problem it is important to define a problem clearly i.e. its definition must be crystal clear. Various program design techniques such as flowcharts were discussed.

Algorithm, Flowcharts, Decision tables were studied. It is always good to follow the above sequence as it becomes easy for development and finding the errors.

SOLVED PROBLEMS AND SELF CHECK QUESTIONS

SOLVED PROBLEMS

01: A sequence of steps required to solve a problem written in simple English considering all alternatives is ----
----.

- a) **algorithm**
- b) flowchart
- c) decision table
- d) Structured Chart

02: Graphical method for explaining the logic of making decision in tabular format is known as -----.

- a) **Decision table**
- b) Flowchart
- c) Algorithm
- d) Program

SELF-TEST 01

01: The characteristics of an algorithm which ensures that it will stop the execution after some specific timings i.e. it stops after a finite number of instructions are executed.

- a) Finiteness
- b) Input
- c) Output
- d) Generality

02: Which of the following shape is used to indicate the beginning and end of a flowchart?

- a) Oval
- b) Rhombus
- c) Rectangle
- d) Parallelogram

03: Which of the following shape is used to make decisions in a flowchart?

- a) Rhombus
- b) Oval
- c) Connectors
- d) Parallelogram

04: Which of the following option includes examining the solution and deciding whether the problem is solved completely or not?

- a) Look back
- b) Devise Plan
- c) Carrying out the plan
- d) None of the above

05: The decision table is divided into ____ part(s).

- a) two

- b) three
- c) four
- d) one

EXERCISE

- Q1. What is meant by Problem Solving in Computers?
- Q2. What is meant by Problem Definition?
- Q3. What is Program Design?
- Q4. Discuss basic symbols used for drawing a flowchart. flowcharting.
- Q5. Draw a Flowchart to find area of a circle.
- Q6. Define an Algorithm and discuss its characteristics?
- Q7. Write an algorithm to send email.
- Q8. Explain Decision table with an example.

REFERENCE MATERIAL

REFERENCES

MOOCs

YOUTUBE VIDEOS

WIKIPEDIA

BLOGS

OER (SLIDES, PDFs, TUTORIALS, ETC)

REFERENCE BOOKS

<http://bcastuff.blogspot.com>

<https://www.geeksforgeeks.org>

<https://www.tutorialspoint.com>

ANSWERS TO SELF-CHECK QUESTIONS

- 1) (a) 2. (a) 3. (a) 4. (a) 5. (a)

UNIT 3: PLANNING THE COMPUTER PROGRAM

UNIT OBJECTIVES

After successful completion of this unit, you will be able to

- ❖ Name the different Programming Methodologies (Top-down and Bottom-Up).
- ❖ Explain the term Debugging.
- ❖ Explain the types of debugging.
- ❖ Explain the types of errors in programming.
- ❖ Write a short on Documentation.
- ❖ Tell the advantages of documentation.
- ❖ Tell what is meant by internal and external documentation.

PROGRAMMING METHODOLOGIES:

Top-down Approach:

Once we have defined the problem and have an idea of how to solve it, we can then use the powerful techniques for designing algorithms. Most of the problems are complex or large problems and to solve them we have to focus on to comprehend at one time, a very limited span of logic or instructions. A technique for algorithm design that tries to accommodate this human limitation is known as top-down design or stepwise refinement.

Top down design provides the way of handling the logical complexity of algorithm. It allows building solutions to problems in a step by step manner. In this way, specific and complex details of the implementation are encountered only at the stage when sufficient groundwork on the overall structure and the relationship among the various parts of the problem is understood.

Bottom-up Approach:

In **bottom up** approach, you identify lower-level tools that you can compose to develop a bigger program. Basically you focus on parts of the problem to solve it. In this approach, bottom level modules are developed first (Lower level module developed, tested and debugged). Then the next module is developed, tested and debugged. This process is continued until all modules have been completed.

This approach is exactly opposite to the top-down approach. This approach is good for reusability of code.

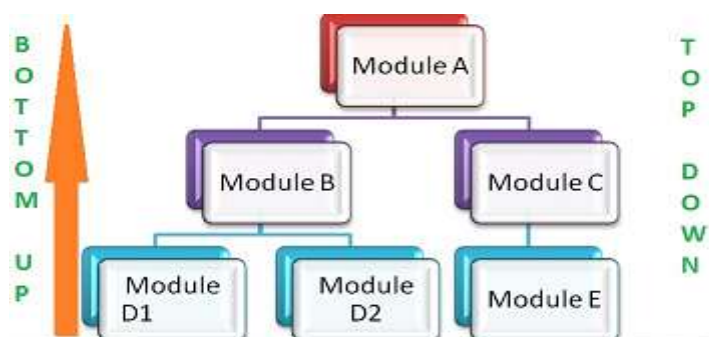


Figure 2.1 : Top Down and Bottom Up Approach

Differences between Top-down Approach and Bottom-up Approach

- Top-down approach analyzes risk by aggregating the impact of internal operational failures while bottom-up approach analyzes the risks in individual process using models.
- Top-down approach doesn't differentiate between high frequency low severity and low frequency high severity events while bottom-up approach does.
- Top-down approach is simple and not data intensive whereas bottom-up approach is complex as well as very data intensive.
- Top-down approaches are backward-looking while bottom-up approaches are forward-looking.

DEBUGGING:

Bug means an error and debugging means correcting the error. Debugging is the process of finding and resolving of defects or problem within the program that prevent correct operation of **computer software** or a **system**. Debugging tactics can involve **interactive** debugging, **control flow** analysis, **unit testing**, **integration testing**, **log file analysis**, monitoring at the **application** or **system** level, **memory dumps**, and profiling. Debugging checks, detects and corrects errors or bugs to allow proper program operation according to set specifications.

Types of Debugging:

1. **Print debugging (or tracing)/Brute Force Method**

It is the act of watching (live or recorded) trace statements, or print statements, that indicate the flow of execution of a process. This is sometimes called *printf debugging*, due to the use of the **printf** function in C. This kind of debugging is turned on by the command TRON in the original versions of the novice-oriented **BASIC** programming language. TRON stood for, "Trace On." TRON caused the line numbers of each BASIC command line to print as the program ran.

2. Backtracking:

This is a common approach., during this approach, starting from the statement at which an error symptom has been discovered, the source code is derived backward till the error is discovered. This method is little tedious because if the number of lines to be derived back are more , the quantity of potential backward work will increase

3. Cause Elimination Method:

In this approach, a listing of causes that may have contributed to the error symptom is developed and tests are conducted to eliminate every error.

TYPES OF ERRORS:

There are three types of error: **syntax errors**, **logical errors** and **run-time errors**. (Logical errors are also called **semantic** errors).

1. **Syntax error:** Every language has an associated grammar, and the program run in that language has to follow the rules of that grammar. For example in English a sentence such as "Shyam is, playing, with a ball" . This sentence is syntactically incorrect because commas should not come the way they are in the sentence.

Likewise, C also follows certain syntax rules. When a program is compiled, the compiler will check that the program is syntactically correct. If there are any syntax errors in the program, those will be displayed on the screen with the corresponding line numbers. Suppose we don't give a semicolon after the statement such as `printf(" YCMOU")` , the C compiler will display a syntax error.

2. **Logical errors or semantic errors:** After the program is compiled and linked successfully we execute the program. Now there are three possibilities:

- 1) The program executes and we get correct results,
- 2) The program executes and we get wrong results, and
- 3) The program does not execute completely and aborts in between.

The first case simply means that the program is correct. In the second case, we get wrong results; it means that there are some logical mistakes in our program. This kind of error is known as logical error. This error is the most difficult to correct. This error is corrected by debugging. Suppose if we write the statement in 'C' language as :

`if(per>=75) printf("Fail");` This is considered as a logical error. Logical errors are checked by human beings and not by a Compiler.

3. **Run-time errors:** A **runtime error** is a program error that occurs while the program is running. The term is often used in contrast to other types of program errors, such as syntax errors and compile time errors. There are many different types of runtime errors. An example of runtime error is a memory leak. This type of error causes a program to continually use up more RAM while the program is running. Another run time error is entering the divisor as zero.

DOCUMENTATION:

Any written text, illustrations or video that describe a software or program to its users is called **program or software document**. User can be anyone from a programmer, system analyst and administrator to end user. At various stages of development multiple documents may be created for different users. In fact, **software documentation** is a critical process in the overall software development process.

In modular programming, documentation becomes even more important because different modules of the software are developed by different teams. If anyone other than the development team wants to or needs to understand a module, good and detailed documentation will make the task easier.

These are some guidelines for creating the documents –

- Documentation should be from the point of view of the reader.
- Document should be unambiguous.
- There should be no repetition
- Industry standards should be used.
- Documents should always be updated.
- Any outdated document should be phased out after due recording of the phase out.

Advantages of Documentation

These are some of the advantages of providing program documentation –

- Keeps track of all parts of a software or program.
- Maintenance is easier.
- Programmers other than the developer can understand all aspects of software.
- Improves overall quality of the software.
- Assists in user training.
- Ensures knowledge de-centralization, cutting costs and effort if people leave the system abruptly.

Example Documents

Software can have many types of documents associated with it. Some of the important ones include –

- **User manual** – It describes instructions and procedures for end users to use the different features of the software.

- **Operational manual** – It lists and describes all the operations being carried out and their inter-dependencies.
- **Design Document** – It gives an overview of the software and describes design elements in detail. It documents details like **data flow diagrams, entity relationship diagrams**, etc.
- **Requirements Document** – It has a list of all the requirements of the system as well as an analysis of viability of the requirements. It can have user cases, real life scenarios, etc.
- **Technical Documentation** – It is a documentation of actual programming components like algorithms, flowcharts, program codes, functional modules, etc.
- **Testing Document** – It records test plan, test cases, validation plan, verification plan, test results, etc. Testing is one phase of software development that needs intensive documentation.
- **List of Known Bugs** – Every software has bugs or errors that cannot be removed because either they were discovered very late or are harmless or will take more effort and time than necessary to rectify. These bugs are listed with program documentation so that they may be removed at a later date. Also they help the users, implementers and maintenance people if the bug is activated.

Internal and External Documentation:

- Internal documentation is written in a program as comments.
- External documentation is written in a place where people who need to use the software can read about how to use the software. External documentation can be broken down into library documentation, which describes tools that a programmer can use, and user documentation, which is intended for users of an application.
- The line between internal and library documentation is not clear-cut because the trend is to write library documentation inside a program as comments, relying on software that extracts the documentation and puts it into a form suitable for people who only want to use the library. For example, the extensive Java library documentation is created by software called javadoc that reads Java programs, including comments, and writes documentation.

SUMMARY :

In this chapter we studied two programming methodologies i.e. Top Down and Bottom Up approach. Concept of Debugging and various types of errors were discussed. Finally concept of documentation which include Internal and External documentation was discussed. Documentation plays an important role for the developer as well as for the end user.

SOLVED PROBLEMS AND SELF CHECK QUESTIONS

SOLVED PROBLEMS

SELF-TEST 01

Author should provide here about 5-7 MCQs, on this Unit

MCQ 01: In ____ approach, you identify lower-level tools that you can compose to become a bigger program.

- a) **Bottom-up**
- b) Top Down
- c) Iterative
- d) Recursive

MCQ 02: It is the process of debugging a program running on a system different from the debugger.

- a) **Remote Debugging**
- b) Print Debugging
- c) Post Mortem debugging
- d) Wolf fencing

SELF-TEST 01

Q1. _____ is the process of finding and resolving of defects or problem within the program that prevent correct operation of computer software or a system.

- a) **Debugging**
- b) Flowchart
- c) Algorithm
- d) Decision Table

Q2. _____ approach analyzes risk by aggregating the impact of internal operational failures while bottom-up approach analyzes the risks in individual process using models.

- a) Top Down
- b) Bottom up
- c) Vertical
- d) Horizontal

Q3. _____ errors are related to the grammar of the language.

- a) **Syntax**
- b) Logical
- c) Semantic
- d) Run Time

Q4. _____ errors occur at the time of execution.

- a) **Syntax**
- b) Logical
- c) Semantic

d) Run Time

Q5_____ documentation is written in a program as comments.

- a) Internal
- b) External
- c) Both a & b
- d) Manual

EXERCISE

Q1. Discuss in short various programming methodologies i.e. top-down and bottom-up.

Q2. Differences between Top-down Approach and Bottom-up Approach.

Q3. What is meant by debugging?

Q4. Name some debugging types?

Q5. Discuss the main three types of Errors.

Q6. What is meant by Documentation? Discuss its types.

Q7. Discuss examples of different types of Software Documents.

REFERENCES

MOOCs

YOUTUBE VIDEOS

WIKIPEDIA

BLOGS

OER (SLIDES, PDFs, TUTORIALS, ETC)

REFERENCE BOOKS

<https://techterms.com/definition/program>

<https://en.wikipedia.org/wiki/Debugging>

ANSWERS TO SELF-CHECK QUESTIONS

1. (a) 2. (a) 3.(a) 4.(d) 5.(a)

STUDYING UNIT 4 : INTRODUCTION TO 'C' LANGUAGE.

UNIT OBJECTIVES

After successful completion of this unit, you will be able to

- ❖ Generations of Computer Languages.
- ❖ History of 'C' Language.
- ❖ Features of 'C' Language
- ❖ Basic Constructs of 'C' Language.

HOW TO STUDY FROM EXISTING TEXT BOOK AND SUPPLEMENTARY TEXT

4.1 Generations of Computer Languages:

1. First Generation Language :(Machine Language)
 - a. Punched Cards were used as input device.
 - b. To Punch on the Punched Card was very tedious
 - c. Once punched one cannot make can changes, i.e. no rectification.
 - d. It was a bulky process to type a program, as one can punch only 80 characters on a single punched card.
 - e. In this no translator was required as it was directly given to the computer.
2. Second Generation Language : (Assembly Language)
 - a. It used Mnemonics i.e. codes such as ADD, DAD, SUB, DAA.
 - b. These codes were to be converted to binary using Translators known as Assembler. Assemblers such as MASM, TASM are available. Its work is to convert the program written in assembly language to machine language.
 - c. This language was meant for only expert people. Layman found the mnemonics difficult to remember.
3. Third Generation Language : (High Level Language)
 - a. It uses simple English words such as PRINT, STOP, END, REM etc. These were easily understood by Layman.
 - b. Program written in high level language is to be converted to machine language using translators known as Compilers and Interpreters.
 - c. Compilers convert the program to machine language, they give the errors if any at the end of the program, whereas an interpreter converts one line at a time, if any error in that line, it will give us error and tell us to rectify it. Thus an interpreter is Line by Liner converter. Compilers are fast and Interpreters are slow. Compilers are meant for Expert people whereas Interpreters are for Novice people.
 - d. Some High Level Languages are BASIC: Beginners All Purpose Symbolic Instruction Code. It was developed by John G. Kemeny and Thomas E Kurtz during 1964. It is a general purpose language. FORTRAN: Formula Translation: It is a mathematical

language. COBOL: Common Business Oriented Language: It is a banking language.
PASCAL: A scientific language.

4. Fourth Generation Language : (4GL)

- a. To reduce programming effort and the time to develop software, visual environment was provided which resulted in reduction of time and cost of software development. Visual basic, Visual age, CSS, SQL, Query Languages, Report Generators, Form designers were provided readily by these languages. Drag and Drop techniques were used.

5. Fifth Generation Language : (AI)

- a. The Fifth generation programming languages are also known as natural languages.
- b. Provides a visual or graphical interface for creating source codes.
- c. It allows people to interact with computers without needing any specialized knowledge.
- d. People can talk with computers and the voice recognition systems can convert the spoken sounds into written words.
- e. Prolog and Mercury are the best fifth generation languages.
- f. Used mainly for Artificial Intelligence.

4.2 History of 'C' Language:

- 1. 'C' is a general purpose, structured programming language. It is a base for programming.
- 2. It was developed in 1972 by Dennis MacAlistair Ritchie at Bell Labs , USA (AT &T: American Telephone and Telegraph)
- 3. It is an Offspring of two earlier languages Basic Combined Programming Language (BCPL) and 'B' Language which were used to develop an Operating System later known as UNIX.
- 4. Till 1978 it was confined to Bell labs only. It was only when Dennis M Ritchie and Brian Wilson Kernighan published an article, then 'C' became popular
- 5. It is categorized as a Middle Level Language. Machine language and Assembly language are considered as Low Level Language and the other three languages are considered as High Level Language. 'C' lies between Low Level and High Level Language.

4.3 Features of 'C' Language:

- 1. It is a very powerful and flexible language: It offers tools for data structures such as stacks, queues, trees, linked list. It is very close to the machine.
- 2. It is highly structured. It has inbuilt control structured such as case, if else for, while, do-while.
- 3. 'C' is portable i.e. .program written on one machine can be compiled and run on another machine by making few or no changes. It is thus machine independent.
- 4. 'C' is case sensitive.
- 5. It has only 32 keywords.
- 6. It is a language of functions.
- 7. It has a very powerful set of operators.
- 8. Another important feature of C program, is its ability to extend itself.

9. Programs written in C are efficient and fast. This is due to its variety of data type and powerful operators.
10. It is used to write applications, Operating Systems like Windows and many other programs like Oracle database, Git, Python interpreter, Network drivers, Print spoolers, utilities etc.
- 11.

Basic Constructs of 'C' Language:

4.4 'C' Character Set:

Any language contains a collection of characters which are used to represent data and information. The collection of characters used in 'C' programming language is called as a character set. 'C' character set consists of:

1. Alphabets : A to Z, a to z
2. Digits : 0 to 9
3. White spaces : Blank Space, Horizontal Tab(\t), Carriage return(\r), New Line(\n), Form feed(\f)
4. Special characters :

,	.	;	:	?	'	"	!		/	\	~	_	\$	%
&	^	*	-	+	<	>	(	)	[	]	{	}	#	

4.5 'C' Tokens: The smallest individual units of a program are known as tokens just as in a passage of text individual words and punctuation marks are called tokens. There are **six** types of tokens in 'C' language.

1. Keywords : e.g. float, case
2. Identifiers : e.g. cost, amount;
3. Constants : e.g. 25, 4.5
4. Strings : e.g. "YCMOU", "C language"
5. Operators : e.g. +, -, *, /
6. Special Symbols : e.g. [], ()

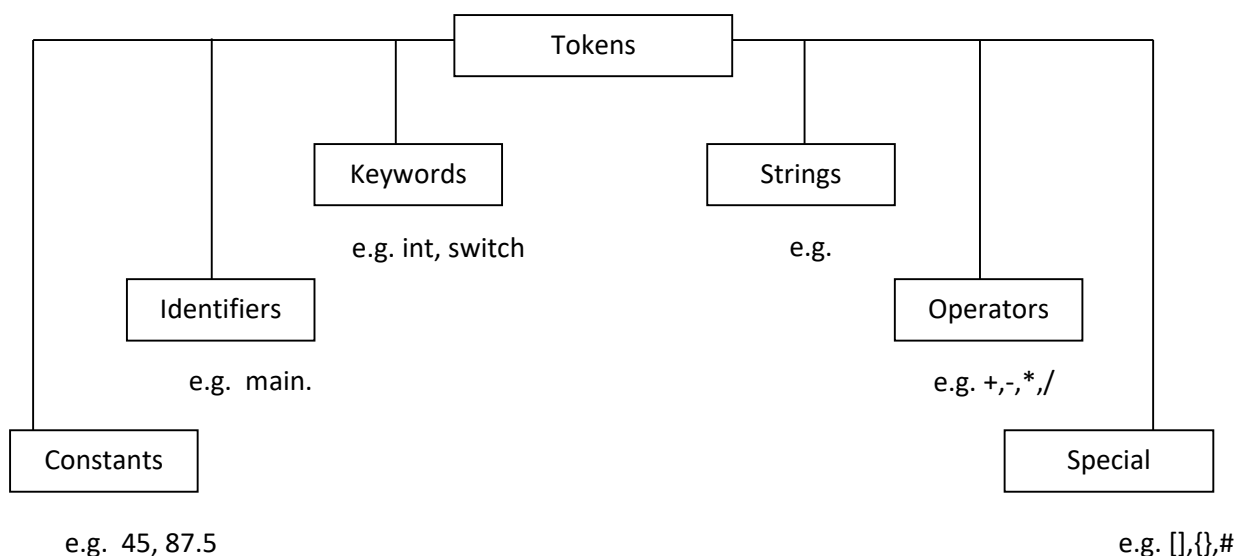


Figure 4.1

Keywords have predefined meaning. They cannot be used for any other purpose. 'C' has 32 keywords. They are always written in lowercase.

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

4.7 Identifiers /Variables: Each program element in a 'C' program is given a name called as identifier. Names given to identify variables, functions and arrays are identifiers. A **variable** is a data name that may be used to store a data value.

Rules:

1. The first character should be an alphabet or underscore and the following characters can be any combination of letters, digits and underscores.
2. Except underscore no special character is allowed.
3. Maximum length of an identifier can be upto 31 characters.
4. Blank spaces are not allowed.
5. 'C' reserved words cannot be used as identifiers.

e.g. valid identifiers : fno, _sno, get_num;

Invalid identifiers : Gross Pay, 1no

Variables can be local or global: The area where a variable can be accessed is known as scope of the variable. It can be local or global. A variable declared outside any function is known as global variable, these global variables can be used by all functions in the program. Local variable is declared inside the function. Such variables are accessed only within the function.

4.8 Constants: The figure 4.2 shows different types of constants. A constant is an entity that does not change but a variable is an entity that may change. Constant is usually referred as a value and variable is a name which stores value at a time. A constant is a fixed value which does not change during the execution of a program.

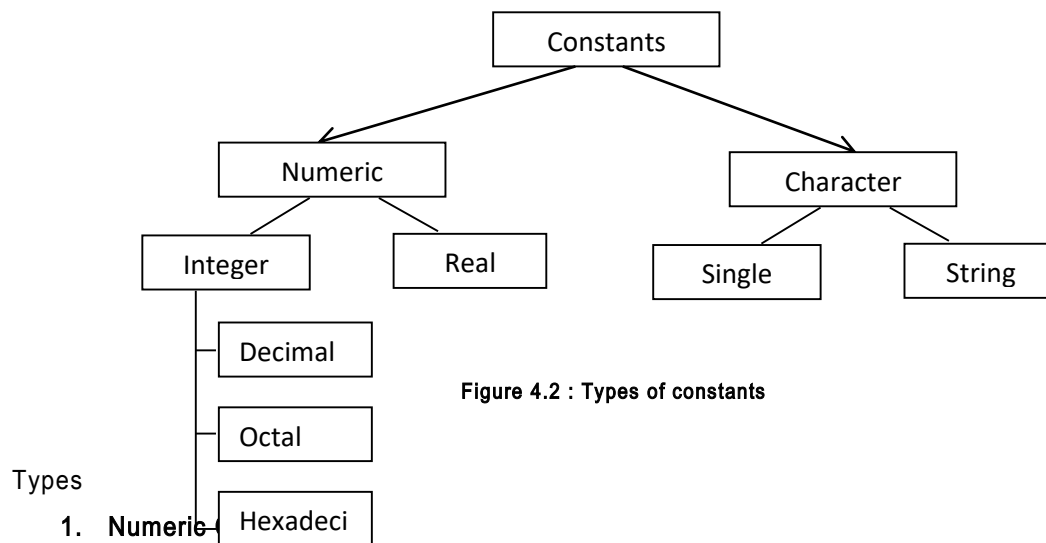


Figure 4.2 : Types of constants

- a. Integer Constant: A number without decimal digits is an integer constant. It is of three types.
 - i. Decimal: It consists of set of digits 0 to 9 prefixed by an optional plus or minus sign. Embedded spaces, commas and other non-digit characters are not allowed.
e.g. 234,-42, +78 are valid decimal integer constants.
e.g. 45 900, 50,000, \$6000 are invalid decimal integer constants.
 - ii. Octal: It consists of any combination of digits from 0 to 7 with a leading zero.
e.g. 056, 0, 0456 are valid octal integer constants.
 - iii. Hexadecimal: A sequence of digits preceded by 0X or 0x. It contains digits from 0 to 9 and alphabets A to F or a to f, e.g. 0X2, 0x9a, 0X8B are valid hexadecimal constants.
- b. Real constant: A number with decimal digits is called as a real constant. It is also called as floating point constant. It is usually used to represent quantities that vary continuously such as heights, temperature etc. It is formed with the digits 0 to 9 and a decimal point. The number can be preceded by optional + or – sign. The digits before or after the decimal point can be omitted. No white spaces or special characters are allowed. It can be represented with f for floating and l for double precision floating constants. It can be represented in two forms.
 - i. Fractional form: Numbers are written with decimal point.
e.g. 0.0083,-0.86,678.29, +567.0,+.5
 - ii. Exponential form: This is useful for representing very large numbers and very small numbers. The general form is mantissa e exponent. The mantissa is either a real number expressed in decimal notation or an integer. The exponent is an integer number with an optional plus or minus sign. ‘e’ can be in lowercase or uppercase.
The number 215.65 can be written as 2.1565e2

The number 7500000000 may be written as 7.5E9.

The number -0.000000368 may be represented as -3.68E-7

2. Character Constant:

- a. **Single character constant:** It represents a single character enclosed within single quotes. All character constants have an equivalent integer value which is their ASCII values.e.g. '9' , '%' , 'A'
- b. **String constant:** It represents a set of characters enclosed within double quotes. It may be a sequence of alphabets, numbers, special characters and blank space.
e.g. "YCMOU","UNIVERSITY", "4567", "?% *,"

C supports some **backslash character constants** for output functions. These are known as **escape sequences**.

Constant	Meaning
'\a'	audible alert
'\b'	backspace
'\f'	form feed
'\n'	new line
'\r'	carriage return
'\t'	horizontal tab
'\v'	vertical tab
'\''	single quote
'\"'	double quote
'\?'	question mark
'\\'	backslash
'\0'	null character

Declaring Variables as constants: There are two ways:

1. Using the keyword **const** : Suppose you want the value of PI to be 3.14, it can be done as follows:
const float PI = 3.14;
2. Using **#define** : It must be done before the main() as shown below
#define PI 31.4;

4.9 Operators

4.9.1 Types of Operators: 'C' has a rich set of operators. An operator is a symbol which represents some operation on the operands. 'C' supports following types of operators.

1. Arithmetic operators
2. Relational operators

3. Logical operators
4. Assignment operators
5. Short hand assignment operators
6. Conditional/Ternary Operator
7. Bitwise Operators.
8. Special Operators.

1. **Arithmetic Operators:** They are used to perform arithmetic operations. Used to perform basic arithmetic operations. They can be categorized as Unary and Binary operators.

a. Binary operators: They need two operands.

Operator	Meaning	Example
+	Addition	$x=2+3 =5$
-	Subtraction	$x=3-2 =1$
*	Multiplication	$x=2*3 =6$
/	Division	$x=6/3 =2$
%	Modulo division(remainder)	$x=7\%2 =1$

b. Unary operators: They need only one operand.

Operator	Meaning	Example
-	Unary minus	-10
++	Increment (Pre/Post)	++x or x++
--	Decrement(Pre/Post)	--x or x--

2. **Relational Operators:** They are used to compare the values of the variables. They are also known as comparison Operators.

Operator	Meaning	Example
>	Greater than	$6>5$
>=	Greater than or equal to	$per \geq 75$
<	Less than	$age < 18$
<=	Less than equal to	$age \leq 18$
==	Equal to	$per == 75$
!=	Not equal to	$qty \neq 5$

3. **Logical Operators:** Sometimes we have to check more than one condition at a time then these operators are used. The logical operators are AND (&&), OR(||) and NOT(!) .The AND operator checks for all the conditions to be true then only it will return true else false. OR condition checks for any one or all conditions from the given condition to be true then it returns true otherwise if all conditions are false then it returns false. NOT operator checks for the negative condition

Operator	Meaning	Example	Evaluation
&&	AND	(6>2) && (3>12)	False
	OR	(4>1) (11>12)	True
!	NOT	!(6>1)	False

4. **Assignment Operator** : It is used to assign a value to a variable.

Operator	Meaning	Example
=	assignment	x=5

5. **Short hand assignment operators** : They are used to simplify the coding of certain type of assignment statements.

Operator	Meaning	Example
+=	Value of LHS variable will be added to RHS	x+=5 is equivalent to x=x+5
-=	RHS value will be subtracted from LHS variable and assigned to LHS variable	x-=5 is equivalent to x=x-5
=	Value of LHS variable will be multiplied by the RHS value and it is assigned to LHS variable	x=5 is equivalent to x=x*5
/=	Value of LHS variable will be divided by the RHS value and it will be assigned to LHS variable	x/=5 is equivalent to x=x/5
%=	Value of LHS variable will be divided by the RHS value and the remainder will be stored in the LHS variable	x%=5 is equivalent to x=x%5

6. **Conditional/Ternary Operator** : It is denoted by ?: It is called as ternary as it requires three operands. It can be used instead of if statement. Due to its use the code size can reduce. Its syntax is : **expression1 ? expression 2/value if true : expression 2/value if false**

e.g. if(f>s)

max= f;

else

max =s;

The above code can be rewritten using ternary operator as follows:

max= f>s ? f : s

7. **Bitwise Operators** : They are used to do bit by bit operations .They are used for testing the bits, shifting them right or left. Mostly used for low level programming.

Operator	Meaning	Example
&	Bitwise AND	x=5 , y =6 , x & y x = 0000 0101 y = 0000 0110 x & y = 0000 0100
	Bitwise OR	x=5 , y =6 , x y x = 0000 0101 y = 0000 0110 x y = 0000 0111
^	Bitwise Exclusive OR	x=5 , y =6 , x ^ y x = 0000 0101 y = 0000 0110 x ^ y = 0000 0011
<<	Bitwise left shift	x=5, y =x<<2 x = 0000 0101 y=x<<2 = 0000 01** = 0001 0100
>>	Bitwise right shift	x=5, y =x>>2 x = 0000 0101 y=y>>2 = **00 0101 = 0000 0001
~	Bitwise Complement	x=5, y =~x x = 0000 0101 y=~x = 1111 1010

8. Special Operators: There are some special operators which perform special operations.

Operator	Meaning	Example
,	Comma Operator. Used to string several expressions. Also used in loops. Evaluates from left to right, has lowest precedence.	x= (a=5,b=10,b-a) will evaluate to 5 for(i=1,sum=0;i<5;i++,j--) sum+=i*j; evaluates to 70
sizeof	Unary compile time operator, returns length in bytes of the variable. Used mostly for dynamic memory allocation	char c; sizeof(x)→1 int x ; sizeof(x)→2
& and *	Pointer operators	int k,*p; p=&k;

. and ->	Member selection	used to refer member elements
----------	------------------	-------------------------------

4.9.2 Operator Precedence and Associativity:

The precedence is used to determine how an expression involving more than one operator is evaluated. Operators are given different levels of precedence. Operators are listed in order of **precedence** (highest to lowest). Their **associativity** Indicates in what order operators of equal precedence in an expression are applied.

Operator	Description	Associativity
()	Parentheses: grouping or function call	left-to-right
[]	Brackets (array subscript)	
.	Member selection via object name	
->	Member selection via pointer	
++ --	Postfix increment/decrement	
++ --	Prefix increment/decrement	right-to-left
+ -	Unary plus/minus	
! ~	Logical negation/bitwise complement	
(type)	Cast (convert value to temporary value of type)	
*	Dereference	
&	Address (of operand)	
sizeof	Determine size in bytes on this implementation	
* / %	Multiplication/division/modulus	left-to-right
+ -	Addition/subtraction	left-to-right
<< >>	Bitwise shift left, Bitwise shift right	left-to-right
< <=	Relational less than/less than or equal to	left-to-right
> >=	Relational greater than/greater than or equal to	
== !=	Relational is equal to/is not equal to	left-to-right
&	Bitwise AND	left-to-right
^	Bitwise exclusive OR	left-to-right
	Bitwise inclusive OR	left-to-right
&&	Logical AND	left-to-right
	Logical OR	left-to-right
? :	Ternary conditional	right-to-left
=	Assignment	right-to-left
+= -=	Addition/subtraction assignment	
*= /=	Multiplication/division assignment	
%= &=	Modulus/bitwise AND assignment	

$\wedge$ = =	Bitwise exclusive/inclusive OR assignment	
<<= >>=	Bitwise shift left/right assignment	
,	Comma (separate expressions)	left-to-right

e.g.

In the expression: $a + b * c / d$

the * and / operations are performed before the + because of precedence. Further, b is multiplied by c before it is divided by d because of associativity.

6.Special Symbols : Square brackets [], used for single and multi-dimensional arrays, Curly brackets{} used for opening and closing of the code, Hash # used for header files.

4.10 Expression: An expression is a combination of variables, constants and operators arranged according to the syntax of the language. Expressions can be arithmetic, relational, logical and conditional Expressions are evaluated using an assignment statement.

e.g. $ans = a*b*c$, $ans = (a*b)*(c*d)$, $ans = p/q+r$

4.11 .1Data Types: The figure 4.3 describes different types of Data types.Data types are the specifications provided with the variable or a function name. Data types in C determine the following:

- Type of the data stored in a variable.
- Range of the data a variable can contain.
- Number of bytes it occupies in the memory.

Broadly they are categorized into **three** types:

1. Primary /Fundamental data types.
2. Derived data types.
3. User defined data types.

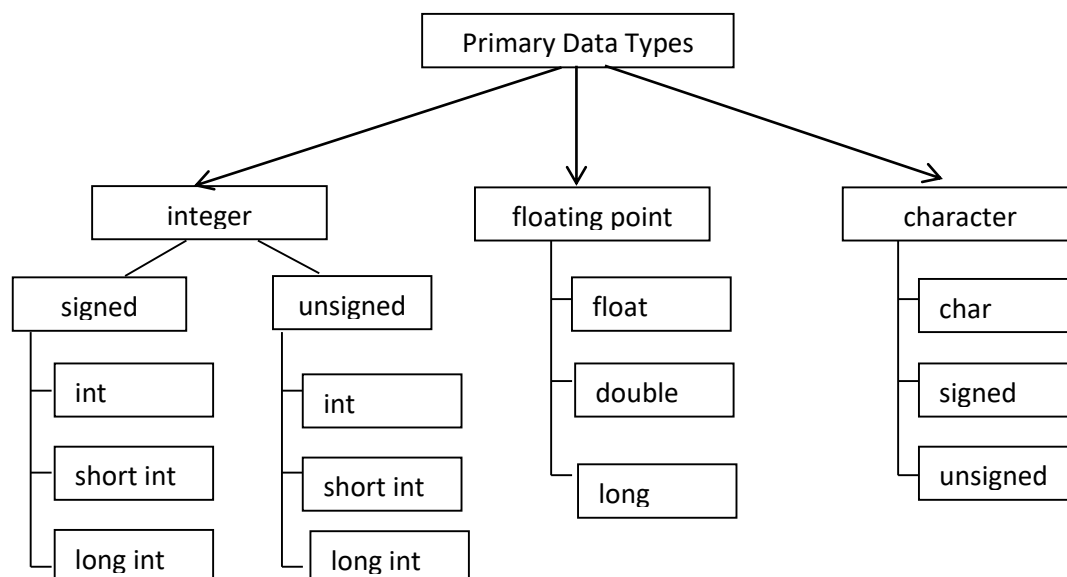


Figure 4.3

1. **Primary /Fundamental data types:** The following **five** data types fall under Primary data types.
 1. Character data type: Stores a single character. Keyword used is char. e.g. char choice;
 2. Integer data type: Stores integer values only i.e. whole numbers only. No decimal points. It can be negative or positive. Keyword used is int. e.g. int rollno;
 3. Float data type: Stores values with decimal values, i.e. Real numbers. Keyword used is float. e.g. float per;
 4. Double data type: Stores decimal values of larger range or larger precision. Keyword used is double. e.g. double amount;
 5. Void data type: Used for functions when no value is to be returned. Keyword used is void. We also use the void data type to specify empty parameters of a function. The void data type means nothing or no value.

4.11.2 Modifiers/Qualifiers in 'C':

- In addition qualifiers/modifiers like short, long, signed, unsigned can be used to get the extended data types. The amount of memory space to be allocated for a variable is derived by modifiers.
- Modifiers are prefixed with basic data types to modify (either increase or decrease) the amount of storage space allocated to a variable.
- For example, storage space for int data type is 4 byte for 32 bit processor. We can increase the range by using long int which is 8 byte. We can decrease the range by using short int which is 2 byte. Note that storage size and range for int and float data type will vary depend on the CPU processor (8,16, 32 and 64 bit)
- short can be used with int data-type.
- long can be used with int and double data-type.
- signed and unsigned can be used with both int and char data-types.
- Using signed int is redundant as int is signed by default.

Size and range of Integer type on 16-bit machine:

Type	Size(bytes)	Range	Format Specifier
int or signed int	2	-32,768 to 32767)	(%d, %d
unsigned int	2	0 to 65535	(%d)
short int or signed short int	1	-128 to 127	(%hd)
unsigned short int	1	0 to 255	(%uh)
long int or signed long int	4	-2,147,483,648 to 2,147,483,647	(%ld)
unsigned long int	4	0 to 4,294,967,29	(%lu)

Size and range of float type on 16-bit machine:

Type	Size(bytes)	Range	Format Specifier
Float	4	3.4E-38 to 3.4E+38	(%f)

double	8	1.7E-308 to 1.7E+308	(%lf)
long double	10	3.4E-4932 to 1.1E+4932	(%Lf)

Size and range of char type on 16-bit machine:

Type	Size(bytes)	Range	Format Specifier
char or signed char	1	-128 to 127	(%c)
unsigned char	1	0 to 255	(%c)

2. **Derived data types:** Array, pointer, structure and union are called derived data type in C language.

3. **User defined data type:** 'C' has a feature of type definition. It allows the user to define the variable of the type of existing type variable.

Syntax: typedef [data_type] new_data_type

```
typedef float amount ;
amount total;
```

Another user defined data type is enumerated data type. It consists of named integer constants as a list. It start with 0 (zero) by default and value is incremented by 1 for the sequential identifiers in the list.

Syntax: enum data-type/identifier {alue1, value2, value3.....valuen}

where data-type is the name of the user defined data type.

e.g. enum month {Jan, Feb, Mar};

```
enum month m1, m2;
```

```
m1 = Jan; m2 = Feb;
```

4.11.3 Data Type Conversion: 'C' allows users to manipulate the expression of different data types. I.e. if one variable of integer data type has to store multiplication of float and integer then it is necessary to convert integer values to float. This is called data type conversion. There are two types of conversion. They are: 1. Implicit 2. Explicit

An **Implicit** conversion is done internally by the compiler. It covertes implicitly as shown in the hierarchy diagram. Usually implicitly it converts from lower data type to higher data type. C performs automatic conversions of type in order to evaluate the expression For example, if we have an integer data type value and a double data type value in an expression then C will automatically convert integer type value to double in order to evaluate the expression

An **Explicit** conversion is done by the users. The conversion of one data type to another is known as type cast, here one can do type casting from lower to higher or higher to lower data type. We can forcefully change the type of the operand by using a cast. **Syntax :** (type) expression

e.g. a = (int) 19.5 : Here 19.5 will be converted to 19 by truncating the original value.

```
c = (double) total/num: The answer will be in floating values.
```

4.12 Storage classes for variables in 'C': Variables 'C' don't only have data type but also have storage class that provides information about their location and visibility. The storage class decides the portion

of the program within which the variables are recognized i.e. scope of a variable. There are four storage classes as follows:

1. **auto**: Local variable known only to the function in which it is declared. The default storage class is auto. e.g. auto int counter.
2. **static** : Local variable which exists and retains its value even after the control is transferred to the calling program. e.g. static int l;
3. **extern** : Global variables known to all functions in the file. e.g. extern long total
4. **register**: Local variable which is stored in the register of the processor. e.g. register int count;

4.13 Basic Structure of 'C' Program: Any C program must consist of 6 main parts as shown below.

1. **Documentation Section:** It consists of the Comments. Comments can be single line or multiline.

```
/* Prog No 1*  
Aim : Use of printf */  
//Use of printf
```

2. **Link Section :** It consists of the Header Files

```
#include<stdio.h>
```

3. **Definition Section:** It consists of Symbolic constants and macros

```
#define pi 3.14
```

4. **Global Declarations:** It consists of declaration of global variables

```
char name[20]; int age;
```

5. **main() : Function section**

```
{  
    Local declaration part (local variables)  
    Executable part (statements)  
}
```

6. **Subprogram section**

```
Function1 () { }  
Function2 () { }...
```

All sections except the main function section may be absent when they are not required.

4.14 Executing a 'C' program:

1. **Creating the program:** The Source program is written using an editor.
2. **Compiling the program:** The source program is translated into a form that is suitable for executing the computer. Every statement is checked for syntax errors. The compiled program is called as an object program.
3. **Linking the program :** The process of putting together other program files and functions required by the program, e.g. if the program is using pow() function, then the object code of this function should be brought from the math library and linked to the main program. The compiled and the linked program is called as executable object code.

4. Executing the program: This loads the executable code into the memory and executes the instructions. In case source code is modified the entire process of compiling, linking and executing the program should be done again. Logical errors can be checked when the program is executed.

Writing a Simple 'C' Program:

```
/* Sample Program */
/* Purpose: Area of a Rectangle */
#include <stdio.h>
#include <conio.h>
main ()
{
    int l, b, area;
    l = 4;
    b = 6;
    area = l * b;
    printf ("Area is = %d\n", area);
}
```

Explanation:

1. #include is called as Preprocessor directive. It commands that the contents of the file stdio.h should be included

Note: All preprocessor begin with # sign

must be written in first column

It must not end with a ;

Only one preprocessor is allowed in a single line

2. main (): It defines a function .It indicates the start of a 'C' program. No semicolon.

3. Braces: {}

4. l,b, area are variables (declaration statements)

5. l = 4 ; b=6 (assignment statements)

6. area = l*b (arithmetic expression)

7. printf (to display the answer)

8. getch() (to pause ,to see the output, wait till the user enters some key)

SUPPLEMENTARY TEXT :

Assignment No 1 : W.a.p to add two numbers [Use of printf, scanf, and mathematical operators]

Algorithm:

1. Start
2. Read the first number and store it in memory variable fno
3. Read the second number and store it in memory variable sno
4. Add them and store the answer in the memory variable sum
5. Display the answer in a suitable format

Program :

```
/* Documentation section */
/* Prog No 1*/
/* Aim: To add two nos*/
```

```
/*Link Section */
#include<stdio.h>
#include<conio.h>
```

```
/*Definition Section*/
#define pi 3.14
```

```
/*Declaration Section */
int fno, sno, sum;
```

```
void main()
{
    clrscr();
```

```
/*Data Entry */
printf("Enter the first no");
scanf("%d",&fno);
printf("Enter the second no");
scanf("%d",&sno);
```

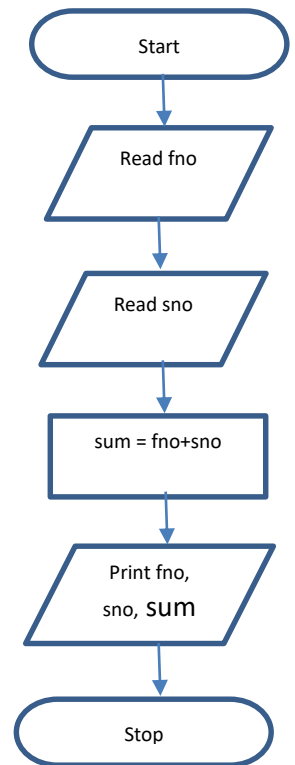
```
/*Processing */
sum = fno+sno;
```

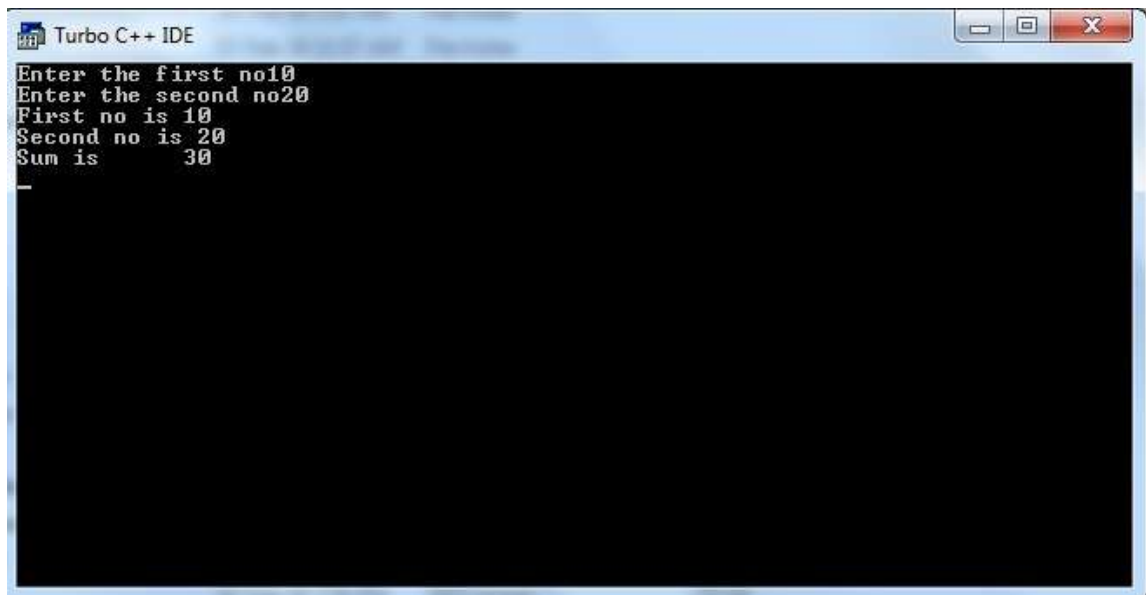
```
/*Display */
printf("First no is %d\n",fno);
printf("Second no is %d\n",sno);
printf("Sum is %d\n",sum);
```

```
getch();
```

```
}
```

Flowchart:





```
Turbo C++ IDE
Enter the first no10
Enter the second no20
First no is 10
Second no is 20
Sum is      30
_
```

SUMMARY :

Various generations of languages were discussed. We started with Machine language , Assembly language, High level Language, Fourth generation Language (4GL). We saw that 'C' is classified as a Middle Level Language. Various basic Constructs used for programming in 'C' language were discussed at a length.

SOLVED PROBLEMS AND SELF CHECK QUESTIONS

SOLVED PROBLEMS

- 1) 'C' is a ____ language.
a) **Middle level** b) low level c) high level d) assembly
- 2) To use comments in a 'C' program ____ is used.
a) /* */ b) REM c) ! d) #
- 3) To define a new data type in 'C' language we use ____ .
a) **typedef** b) switch c) #define d) static

SELF-TEST 01

Author should provide here about 5-7 MCQs, on this Unit

1. There are ____ keywords in 'C' language.
a) 48
b) 32
c) 255
d) 80
2. A data type which is empty or considered as no value ____
a) int
b) char
c) void
d) enum
3. When one of the operands is real and the other is integer, the expression is called as ____ arithmetic expression
a) Mixed
b) Integer
c) Real
d) Complex
4. Conditional operator is ____ operator.
a) ternary
b) binary
c) unary
d) decisional
5. ____ key is used to compile a 'C' program
a) Control + F9
b) Control + F7
c) Alt + F9
d) Alt + F7

EXERCISE

- Q1. Explain the different generation of software languages.
- Q2. Write a note on History of 'C' language.
- Q3. Discuss the Features of the 'C' language
- Q4. Write a note on 'Constants'
- Q4. Discuss different types of Operators in 'C' language.
- Q6. Describe operator precedence and associativity?
- Q7. What is meant by local and global variables? Also discuss the storage classes while declaring a variable.

**REFERENCE MATERIAL :[WWW.GEEKSFORGEEEKS.COM](http://www.geeksforgeeks.com),
[HTTPS://WWW.W3SCHOOLS.COM/C/](https://www.w3schools.com/c/)**

REFERENCES

MOOCs

YOUTUBE VIDEOS

WIKIPEDIA

BLOGS

<https://www.learnonline.com/> (datatypes)

<https://www.studytonight.com/c/datatype-in-c.php/variables-in-c.php>

<https://fresh2refresh.com/c-programming/c-data-types/>

<http://www.eiilmuniversity.co.in/downloads/Computer-Programming.pdf>
oms.bdu.ac.in/ec/admin

<https://www.javatpoint.com/storage-classes-in-c>

<https://www.studytonight.com/c/storage-classes-in-c.php>

OER (SLIDES, PDFs, TUTORIALS, ETC)

REFERENCE BOOKS

ANSWERS TO SELF-CHECK QUESTIONS

- i) (b) 2.(c) 3.(a) 4.(a) 5.(c)

STUDYING UNIT 5 : MANAGING INPUT AND OUTPUT OPERATIONS

UNIT OBJECTIVES

After successful completion of this unit, you will be able to

- ❖ Reading a Character, Writing a Character
- ❖ Formatted Input, Formatted Output.

HOW TO STUDY FROM EXISTING TEXT BOOK AND SUPPLEMENTARY TEXT

5.1 Formatted/Unformatted Input/output functions:

Input and Output are the processes of accepting data and displaying the results to the user.

There are numerous library functions available for I/O. These can be classified into two broad categories:

- (a) Console I/O functions - Functions to receive input from keyboard and write output on the monitor.
- (b) File I/O functions - Functions to perform I/O operations on a floppy disk or hard disk.

The screen and keyboard together are called a console. Console I/O functions can be further classified into two categories— **formatted** and **unformatted** console I/O functions. The basic difference between them is that the formatted functions allow the input read from the keyboard or the output displayed on the monitor to be formatted as per our requirements. For example, if values of average marks and percentage marks are to be displayed on the screen, then the details like where this output would appear on the screen, how many spaces would be present between the two values, the number of places after the decimal points, etc. can be controlled using formatted functions. The functions available under each of these two categories are shown in **Figure**. These functions are declared and related macros are defined in “stdio.h” which is a header file in C language

Console Input/output Functions					
Formatted Function			Unformatted Functions		
Type	Input	Output	Type	Input	Output
char	scanf()	printf()	char	getch() getche() getchar()	putch() putchar()
int	scanf()	printf()	int	-	-
float	scanf()	printf()	float	-	-
string	scanf()	printf()	string	gets()	puts()

Table 5.1

Unformatted I/O functions :

1. **getchar()** : It is used to read a single character from the keyboard. The getchar() function waits until a key is pressed and then returns its value.

Syntax: character_variable = getchar();

where character variable refers to some previously declared character variable

e.g. char c;

```
c=getchar();
```

2. **getch():** The getch() is also used to read a single character from the keyboard, but it returns this character immediately, one does not have to press enter after the character is pressed. It does not display the character typed from the keyboard.

Syntax: character_variable = getch();

where character variable refers to some previously declared character variable

e.g. char c;
 c=getch();

3. **getche () :** The getche () is also used to read a single character from the keyboard, it returns this character immediately, one does not have to press enter after the character is pressed. Here the character is echoed/displayed on the monitor unlike getch() where the character typed from the keyboard is not displayed.

Syntax: character_variable = getche();

where character variable refers to some previously declared character variable

e.g. char c;
 c=getche();

4. **gets ():** This function reads a string of characters entered from the keyboard during execution until the user presses enter. A null character is automatically entered at the end of the string. White spaces, tabs are allowed while entering the string.

Syntax: gets(string_name)

where string name refers to some previously declared string variable

e.g. char name[80];
 gets(name);

5. **putchar():** This function prints a single character on the screen.

Syntax: putchar(character_variable)

where character variable refers to some previously declared character variable

e.g. char c;
 c=getchar();
 putchar(c)

6. **puts() :** This function prints the string on the screen. The puts () function can only output a string of characters, it cannot contain format specifiers.

Syntax: puts(string_name)

where string name refers to some previously declared string variable

e.g. char name [80];
 gets (name);
 puts (name);

1. Formatted I/O functions :

Formatted input in 'C' is done using the **scanf()** function and the formatted output is done using the **printf()** function:

1. scanf() function :

The syntax of scanf () function is:

```
scanf ("format specifiers", <arg1> [<, arg2>.....<, argn>]);
```

"format specifier" specifies the data type for the variable to be read.

arg1....argn specifies the address of the variable where the keyed in data will be stored.

"format specifier" and variable list is separated by commas.

"format specifier" consists of

1. Conversion character '%'
2. A data type character (or type specifier)
3. An optional number specifying the input width.

• Types of formatted input

- Integer input
- Real number input
- Character input
- Mixed-mode input

• Integer input

The format specifier for integer input is written as: %d

d specifies to scanf() function that an integer has to be read.

e.g.

```
int a, b;
```

```
scanf ("%d %d", &a, &b);
```

• Real number input

Unlike integer numbers, the field width of real numbers is not specified and therefore scanf() reads real numbers using the simple specification '%f' for both integer part as well as fraction part. %f is also used for reading decimal notations as well as exponential notations.

e.g.

```
float a, b, c;
```

```
scanf ("%f %f %f", &a, &b, &c);
```

On the data line, if following input is given.

```
123.45 56.78 E-1 9012
```

scanf () will assign 123.45 to "a". 5.678 to "b" and 9012 to "c".

If we wish to read number of double data, we use '%lf' instead of simple '%f'.

- **Character Input**

To read a single character from the keyboard we can use the `getchar ()` function. The same can be achieved using the `scanf ()` function. Besides reading a single character from the keyboard, `scanf ()` function can also be used to read a string made up of more than one character from the keyboard.

The format specifier used with `scanf ()` function to read a character or a string is

`%nc` for characters

`%ns` for string

`%nc` reads data from the keyboard into variable defined as `char`. it will wait till `nth` character is keyed in.

`%ns` reads data from the keyboard into a character array defined earlier in the program.

e.g.

```
char a, b;
```

```
scanf ("%c %c", &a, &b);
```

- **Mixed-mode input**

It is possible to use `scanf ()` statement to input variable of different data types. Appropriate format specifiers should be used for different data types. When an attempt is made to read an item that does not match the type expected, `scanf ()` immediately terminates any further reading and returns the values already read.

e.g.

```
int count; char code;float ratio;char name[20];
```

```
scanf ("%d %c %f %s", &count,&code,&ratio,name)
```

will read data 15 Q 1.56 Tea

Data type specifiers for scanf

No	Code	Meaning
1	<code>%c</code>	Reads a single character
2	<code>%d</code>	Reads a decimal integer
3	<code>%e</code>	Reads a floating point value
4	<code>%f</code>	Reads a floating point value
5	<code>%g</code>	Reads a floating point value
6	<code>%h</code>	Reads a short integer
7	<code>%i</code>	Read a decimal, hexadecimal or octal integer
8	<code>%o</code>	Reads an octal integer
9	<code>%s</code>	Reads a string
10	<code>%u</code>	Reads an unsigned decimal integer
11	<code>%x</code>	Reads a hexadecimal integer

11	%[.]	Reads a string of words
----	-------	-------------------------

2. printf() function :

The syntax of printf () function is:

```
printf ("format specifiers", <arg1> [<, arg2>.....<, argn>]);
```

"format specifier" consists of

1. Characters that will be printed on the screen as they appear.
2. Format specifications that define the output format for display of each item
3. Escape sequence characters such as \n, \t.

"arg1, arg2.....argn" are the variable whose values are formatted and printed according to the specifications given in the format specifier.

A simple format specification has the following form:

%n.d data-type

n is an integer that specifies the total number of columns for the output value and d is another integer number that specifies the number of digits to the right of the decimal point or the number of characters to be printed from a string. Both n and d are optional.

The printf() function does not automatically add the new line character at the end of the display. Therefore, we can use multiple printf () statements to display a single like of output. A new line can be introduced using the printf() function with help of the new line character '\n'.

Types of formatted output

- Integer output
- Real number output
- Character or string output
- Mixed-mode output
- **Integer output**

The format specifier used for printing an integer number is

%nd

n specifies the minimum field width for the output.

In case, the number being printed is greater than the specified width, the entire number will be printed. 'C' overrides the minimum field width definition for this case.

The numbers are printed right-justified by default leading blanks will appear for a number of lesser width than the width specified.

e.g.

Consider the following declaration

```
int a=1234;
```

We will now see how printf () will display the output under different formats

Format	Output	Meaning						
printf ("%d",a);	<table><tr><td>1</td><td>2</td><td>3</td><td>4</td></tr></table>	1	2	3	4	Normal number 1234 will be printed		
1	2	3	4					
printf ("%6d", a);	<table><tr><td></td><td></td><td>1</td><td>2</td><td>3</td><td>4</td></tr></table>			1	2	3	4	Two blank spaces are printed at the beginning as width maintained is 6 digits
		1	2	3	4			
printf ("%2d",a);	<table><tr><td>1</td><td>2</td><td>3</td><td>4</td></tr></table>	1	2	3	4	Normal number 1234 will be printed		
1	2	3	4					
printf ("% -6d",a);	<table><tr><td>1</td><td>2</td><td>3</td><td>4</td><td></td><td></td></tr></table>	1	2	3	4			Two blank spaces will be printed after the number as width maintained is 6 digits
1	2	3	4					
printf ("%06d", a);	<table><tr><td>0</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr></table>	0	0	1	2	3	4	Two blank zeros will be printed before the number
0	0	1	2	3	4			

It is possible to force left-justified printing of numbers using the minus '-' sign directly after the % sign. It is possible to replace leading blanks with zeros. To print long int. we specify used %ld instead of %d in the format specification.

- Real number output**

The format specifier used to display real number in floating point notation with decimals is:

%n.df

The integer value 'n' is the maximum number of positions that are to be used for the display of the whole number part of the value and the integer 'd' indicates the number of digits to be displayed after the decimal point (precision). The value pointed i.e. rounded off to 'd' decimal points and right-justified in a field of 'n' columns.

e.g. Consider the following assignment

float a = 98.7654

We will now see how printf () will display the output under real number format

Format	Output	Meaning											
printf ("%7.4f",a);	<table><tr><td>9</td><td>8</td><td>.</td><td>7</td><td>6</td><td>5</td><td>4</td></tr></table>	9	8	.	7	6	5	4	Total 7 positions and 4 decimal places				
9	8	.	7	6	5	4							
printf ("%7.2f", a);	<table><tr><td></td><td></td><td>9</td><td>8</td><td>.</td><td>7</td><td>7</td></tr></table>			9	8	.	7	7	Total 7 positions and only 2 decimal places				
		9	8	.	7	7							
printf("%-7.2f",a)	<table><tr><td>9</td><td>8</td><td>.</td><td>7</td><td>7</td><td></td><td></td></tr></table>	9	8	.	7	7			Total 7 positions and only 2 decimal places but alignment within the field is left				
9	8	.	7	7									
printf("%10.2e",a);	<table><tr><td></td><td></td><td>9</td><td>.</td><td>8</td><td>8</td><td>e</td><td>+</td><td>0</td><td>1</td></tr></table>			9	.	8	8	e	+	0	1	Converting number into 'notation	
		9	.	8	8	e	+	0	1				
printf("%11.4e", a);	<table><tr><td>-</td><td>9</td><td>.</td><td>8</td><td>7</td><td>6</td><td>5</td><td>e</td><td>+</td><td>0</td><td>1</td></tr></table>	-	9	.	8	7	6	5	e	+	0	1	Printing number with '-' and e notation
-	9	.	8	7	6	5	e	+	0	1			

- **Character or String output:**

Single character can be easily output to desired location using %nc format. The character will be printed right justified in the field of n column. The character can be printed left or right justified, for making left justified put the '-' sign.

Displaying strings:

The general format for displaying the strings is %n.p

where 'n' is the field width for display and 'p' instructs that only the first 'p' characters of the string are to be displayed. The display is right justified.

The following examples show the effect of variety of specifications in printing a string 'NEW DELHI 110001', containing 16 characters including blanks.

Specification : %s

Output:

N	E	W		D	E	L	H	I		1	1	0	0	0	1				
---	---	---	--	---	---	---	---	---	--	---	---	---	---	---	---	--	--	--	--

Specification : %20s

Output:

					N	E	W		D	E	L	H	I		1	1	0	0	0	1
--	--	--	--	--	---	---	---	--	---	---	---	---	---	--	---	---	---	---	---	---

Specification: %20.10s

Output:

											N	E	W		D	E	L	H	I	
--	--	--	--	--	--	--	--	--	--	--	---	---	---	--	---	---	---	---	---	--

Specification: %5s

Output:

N	E	W		D																
---	---	---	--	---	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

Specification: %-20.10s

Output:

N	E	W		D	E	L	H	I												
---	---	---	--	---	---	---	---	---	--	--	--	--	--	--	--	--	--	--	--	--

Specification: %5s

Output:

N	E	W		D	E	L	H	I		1	1	0	0	0	1					
---	---	---	--	---	---	---	---	---	--	---	---	---	---	---	---	--	--	--	--	--

- **Mixed mode output**

Similar to the mixed-mode operations with the scanf () function, a programmer can use mixed mode operations with the printf () function. It is permitted to mix data types in one printf statement. e.g. printf ("%d %f %s %c " , a, b, c, d);

Data type specifiers for printf

No	Code	Meaning
1	%c	Prints a single character
2	%d	Prints a decimal integer
3	%e	Prints a floating point value in exponent form
4	%f	Prints a floating point value without exponent form
5	%g	Prints a floating point value either e type or f type depending on the value
6	%i	Prints a signed decimal integer
7	%o	Prints an octal integer without leading zero
8	%s	Prints a string
9	%u	Prints an unsigned decimal integer
10	%x	Prints a hexadecimal integer without leading 0.
<p>The following letters may be used as prefix for certain conversion characters.</p> <p>h : for short integers l : for long integers L : for long double</p>		

Output Format flags

Flags	Meaning
-	Output is justified within the field. Remaining fields will be blank
+	+ or – will precede the signed numeric item.
0	Causes leading zeroes to appear
#(with o or x)	Causes octal and hexadecimal numbers to be preceded by O and Ox respectively
#(with e,f or g)	Causes a decimal point to be present in all floating point numbers, even if it is a whole number. Also prevents the truncation of trailing zeros in g type conversion

SUPPLEMENTARY TEXT :

Assignment No 1 : W.a.p to add two numbers [Use of printf, scanf, and mathematical operators]

Algorithm:

6. Start
7. Read the first number and store it in memory variable fno
8. Read the second number and store it in memory variable sno
9. Add them and store the answer in the memory variable sum
10. Display the answer in a suitable format

Program :

```
/* Documentation section */
/* Prog No 1*/
/* Aim: To add two nos*/

/*Link Section */
#include<stdio.h>
#include<conio.h>

/*Definition Section*/
#define pi 3.14

/*Declaration Section */
int fno, sno, sum;

void main()
{
    clrscr();

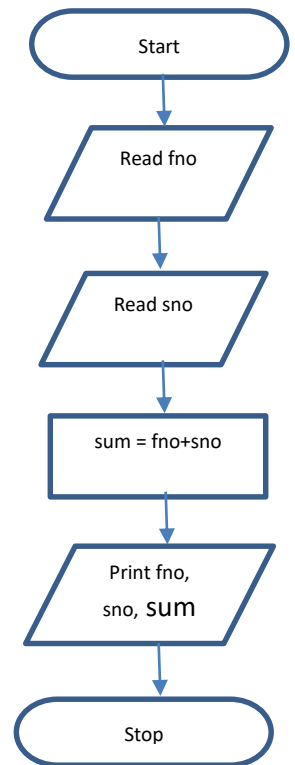
    /*Data Entry */
    printf("Enter the first no");
    scanf("%d",&fno);
    printf("Enter the second no");
    scanf("%d",&sno);

    /*Processing */
    sum = fno+sno;

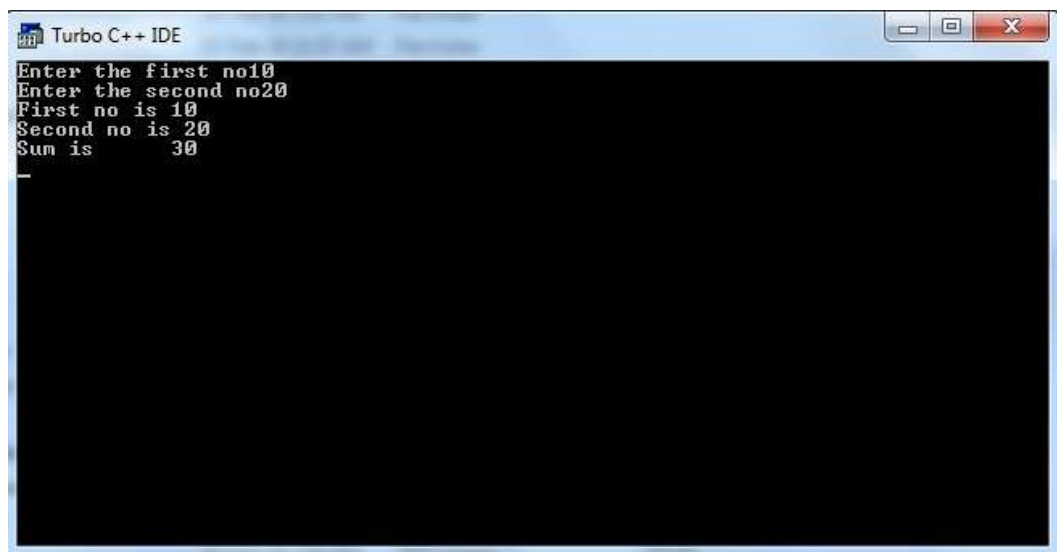
    /*Display */
    printf("First no is %d\n",fno);
    printf("Second no is %d\n",sno);
    printf("Sum is %d\n",sum);

    getch();
}
```

Flowchart:



Output :

A screenshot of the Turbo C++ IDE window. The title bar reads 'Turbo C++ IDE'. The main text area is black with white text. The text displayed is: 'Enter the first no10', 'Enter the second no20', 'First no is 10', 'Second no is 20', and 'Sum is 30'. There is a small white cursor line at the bottom left of the text area.

```
Turbo C++ IDE
Enter the first no10
Enter the second no20
First no is 10
Second no is 20
Sum is 30
```

SUMMARY :

Managing Input and Output Operations were studied . How a Character is read from an Input device and how it written to an Output device were discussed. Various 'C' Library functions were studied. Finally formatted Input/Output concept was discussed.

SOLVED PROBLEMS AND SELF CHECK QUESTIONS

SOLVED PROBLEMS

1. getche() function is defined in ____ header file.
a) **conio.h** b) stdio.h c) ctype.h d) stdlib.h
- 2) ____ is a collection of characters.
a) comment b) **string** c) structure d) union
- 3) %d and %i can be used as format specifiers for integer data type.
a) **TRUE** b. FALSE
- 4) For hexadecimal input we ____ format specifier is used.
a) x b) X c) o **d) Both a & b**
- 5) double data type requires ____ bytes.
a) 2 b)4 **c)8** d) 16

SELF-TEST 01

Author should provide here about 5-7 MCQs, on this Unit

1. _____ are used to print the value of a variable like integer, string.
a) Format Specifiers
b) Escape Sequence
c) Modifiers
d) All of the above
2. \n , \t, \b are known as _____.
a) Modifiers
b) Escape Sequence
c) Format Specifiers
d) Symbolic constants
3. Range of unsigned char data type will be ____
a) -128 to 127
b) 0 to 255
c) -32768 to 32767
d) 0 to 65535

%.[]	Reads a string of words

4. ____ format specifier is used to read a string of words.

- a. %s
- b. %i
- c. %[..]
- d. %d

5. The flag ____ when used with printf will left justify the output.

- a) -
- b) +
- c) o
- d) #

EXERCISE

Q1. What is meant by formatted input? Explain with examples.

Q2. Which format specifiers are used with scanf.

Q3. Explain the format specifiers used with printf with examples.

Q4. Differentiate the three functions getch(), getche() and getchar().

Q5. Write and explain the syntax of printf ().

Q6. Explain the scanf() function in details?

Q7. Explain gets() and puts() functions?

Q8. What is meant by local and global variables? Also discuss the storage classes while declaring a variable.

**REFERENCE MATERIAL :[WWW.GEEKSFORGEES.COM](http://www.geeksforgeeks.com),
[HTTPS://WWW.W3SCHOOLS.COM/C/](https://www.w3schools.com/c/)**

REFERENCES

MOOCs

YOUTUBE VIDEOS

WIKIPEDIA

BLOGS

<https://www.learnonline.com/> (datatypes)

<https://www.studytonight.com/c/datatype-in-c.php/variables-in-c.php>

<https://fresh2refresh.com/c-programming/c-data-types/>

<http://www.eiilmuniversity.co.in/downloads/Computer-Programming.pdf>

oms.bdu.ac.in/ec/admin

<https://www.javatpoint.com/storage-classes-in-c>

<https://www.studytonight.com/c/storage-classes-in-c.php>

OER (SLIDES, PDFs, TUTORIALS,ETC)

REFERENCE BOOKS

ANSWERS TO SELF-CHECK QUESTIONS

1.(a) 2.(b) 3.(a) 4(c) 5(a)

UNIT 6 DECISION MAKING AND LOOPING

UNIT OBJECTIVES

After successful completion of this unit, you will be able to understand:

- ❖ Meaning of Control Structures, Conditional and Unconditional.
- ❖ Decision making structures such as if, if else, nested if, conditional operator, switch statement.
- ❖ Looping structures such as while loop, do-while loop, for loop.
- ❖ Control Structures such as break, continue, goto, exit

5.1 Control Structures: In a sequential execution of a program, the computer starts executing with the first statement in the main program, then the second and so on until the last statement is executed. Sometimes we may require changing the execution of the statements depending upon the various conditions. A situation can arise where a transfer of program control is required by comparing two quantities. This is called as **conditional branching** where a logical decision is carried out inside the computer. Sometimes it is required to transfer the control to some remote part of the program. This is known as **unconditional branching**.

Another situations arises where number of repetitions will not be known in advance .So the group of consecutive statements are repeated for a specified number of times or until some condition is satisfied. This is called as **looping**

Thus Control structures are of two types:

1. Branching : (transfer)
 - a. Conditional
 - b. Unconditional
2. Looping : (repetition)

5.2 Decision making structures : Decision making structures are used to execute a part of the statement based on some condition and exclude the other. It is required to carry out a logical test and then take a specific action. The statements which perform these operations are called as conditional branching. Following are the statement:

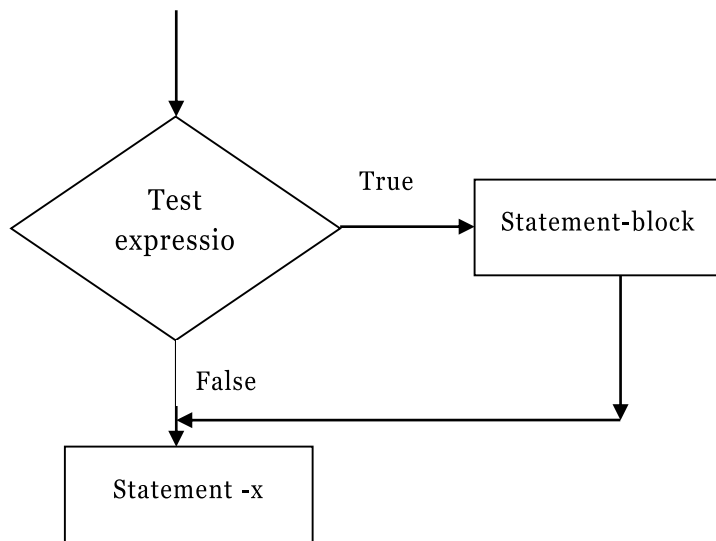
- simple if statement
- if else statement
- nested if ... else statement
- else if ladder
- switch statement

Simple if statement: Simple statement is used to take a particular decision regarding selection of a particular condition. It is used to check whether condition is true or not. In simple if statement only true part is included.

Syntax: if(test expression)

```
{  
    Statement block;  
}  
Statement -x;
```

If the test expression is true, the statement block will be executed, otherwise the statement block will be skipped and the execution will jump to the statement-x. Note that if the condition is true both the statement-block and the statement-x are executed in sequence.



e.g.

```
if(category == 1)
{
    marks = marks+bonus_marks;
}
printf("%f", marks)
```

If the student belongs to category 1 then additional bonus_marks are added to his marks otherwise no increase in his marks.

W.a.p to find greatest among three numbers using simple if and logical operators.

/* Use of Simple If*/

```
#include <stdio.h>
```

```
void main()
```

```
{
```

```
    int f,s,t;
```

```
    printf("Enter the three numbers :");
```

```
    scanf("%d %d %d",&f,&s,&t);
```

```
    if(f>s && f>t)
```

```
    {
```

```
        printf("%d is largest",f);
```

```
    }
```

```

if(s>f && s > t)
{
    printf ("%d is largest",s);
}
if(t>f && t>s)
{
    printf ("%d is largest",t);
}
if(f== s && s == t)
{
    printf("All numbers are equal");
}
}

```

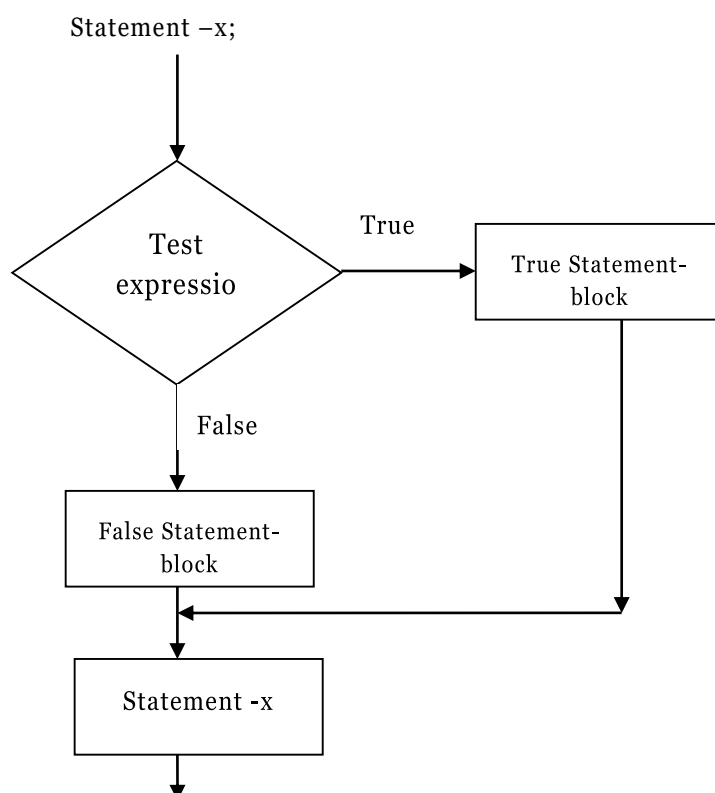
if else statement : It is the extension of simple if statement.. In simple if statement only true part is included, whereas here we also have false part.

Syntax: if(test expression)

```

{
    True Statement block;
}
else
{
    False Statement block;
}

```



If the test expression is true, then the true statement block will be executed otherwise the false statement block will be executed. In either case either true block or false block will be executed not both.

```
e.g.    if(num%2== 0)
        printf("%d is even",num);
    else
        printf("%d is odd",num);
    }
```

```
printf("%f", marks)
```

If the student belongs to category 1 then additional bonus_marks are added to his marks otherwise no increase in his marks.

W.a.p to decide whether a person can vote or cannot vote.

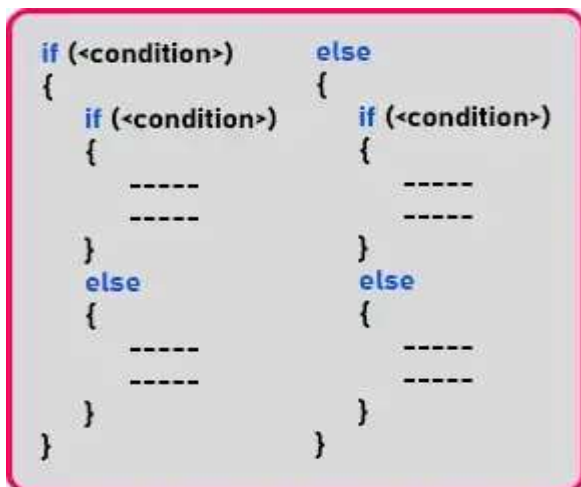
/* Use of if else */

```
#include <stdio.h>

main()
{ int age;
  char name[20];
  printf("Enter your name :");
  scanf("%s",name);
  printf("Enter your age :");
  scanf("%d",&age);
  if(age>=18)
      printf(" %s, you are eligible to vote as your age is %d",name,age);
  else
      printf(" %s, you are not eligible to vote as your age is %d",name,age);
}
```

Concept of Nested IF statement to make decisions.

Use: If inside If (Multiple Ifs nested among each other)



Example: if (gender=='M')

```
{
```

```
        if(score>=75)
            printf("Eligible for Computer Department");
        else
            printf("Eligible for Marketing Department");
    }
else
{
    if(score>=75)
        printf("Eligible for Accounts Department");
    else
        printf("Eligible for Quality Control Department");
}
```

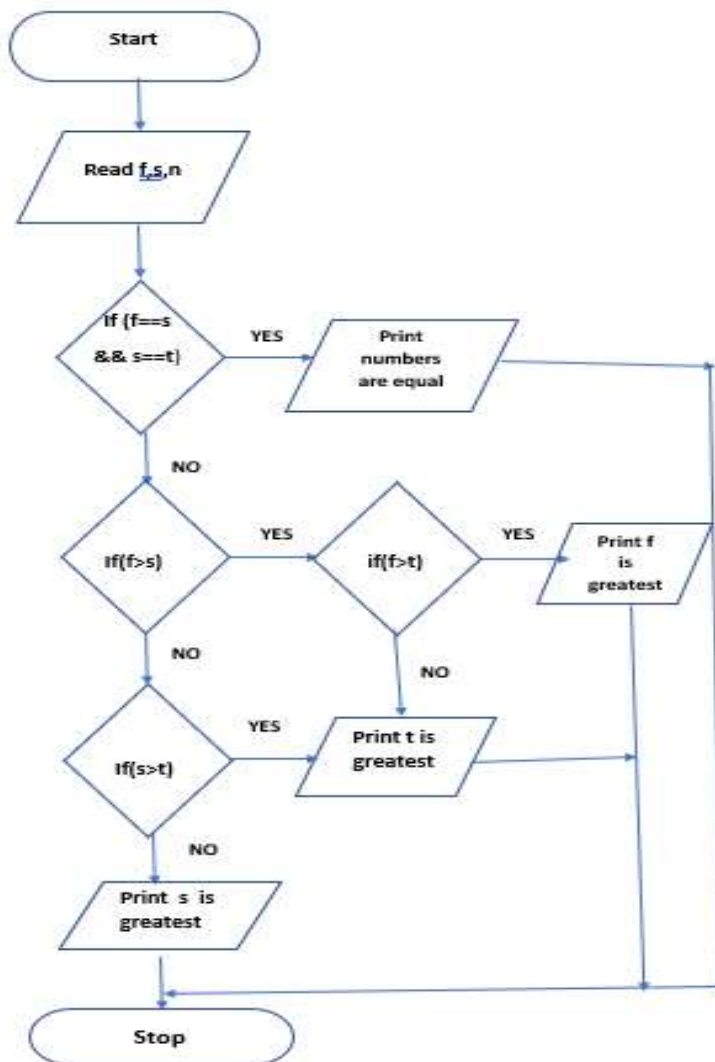
W.A.P to decide greatest among three numbers. Write an algorithm and draw the flowchart also.

Topics Covered: Use of Nested IF statement

Algorithm:

1. Start
2. Read the three numbers and store them in memory variables f,s,t
3. Using if statement decide whether they are equal or not.
4. Using Nested if decide which number is greatest
5. Display the answer in a suitable format.
6. Terminate

Flowchart to show the use of Use of Nested IF statement to find greatest among three numbers.



/*Documentation Section */

/* Prog No : Nested IF */

/*W.a.p to find greatest among 3 nos f,s,t*/

/*Link Section */

#include<stdio.h>

#include<conio.h>

main()

{

int f,s,t;

clrscr();

printf("Enter the three nos :");

scanf("%d%d%d",&f,&s,&t);

if(f==s&&s==t)

printf("Nos are equal");

else

{

if(f>s)

{

if(f>t)

printf("Among %d,%d,%d :%d is greatest",f,s,t,f);

else

printf("Among %d,%d,%d :%d is greatest",f,s,t,t);

}

else

{

if(s>t)

printf("Among %d,%d,%d :%d is greatest",f,s,t,s);

else

printf("Among %d,%d,%d :%d is greatest",f,s,t,t);

}

}

getch();

}

OUTPUT:

```
Enter the three nos :10 20 30
Among 10,20,30 :30 is greatest

Enter the three nos :10 50 20
Among 10,50,20 :50 is greatest

Enter the three nos :60 20 10
Among 60,20,10 :60 is greatest_

Enter the three nos :10 10 10
Nos are equal_
```

Concept of Conditional Operator: (Substitute for IF statement)

Use : To make decisions , it is Ternary Operator as it requires 3 operands.

Syntax :

```
if(conditionn)
    { true part}
else
    {false part}
condn? true part : false part;
```

Example :

Using If statement :

```
if(f>s)
    printf("%d is greatest",f);
else
    printf("%d is greatest",s);
```

Using Conditional Operator:

```
f>s? big=f: big=s;
printf("Among %d,%d, %d is greatest",f,s,big);
```

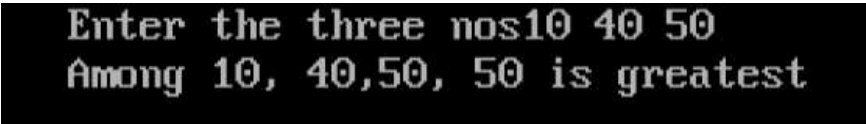
W. A.P to decide greatest among three numbers using Conditional Operator

/*Documentation Section */

/*W.a.p to find greatest among 3 nos f,s,t using conditional operator*/

```
#include<stdio.h>
#include<conio.h>
main()
{
int f,s,t,big;
clrscr();
printf("Enter the three nos");
scanf("%d%d%d",&f,&s,&t);
if(f==s&&s==t)
printf("Nos are equal");
else
{
f>s?(big=f>t?f:t):(big=s>t?s:t);
printf("Among %d, %d,%d, %d is greatest",f,s,t,big);
}
getch();
}
```

OUTPUT:



```
Enter the three nos10 40 50
Among 10, 40,50, 50 is greatest
```

Use of switch case:

Use : It is a structured way used instead of if.

Syntax:

```
switch(expression)
{
case value-1:
statements;
break;
case value-2:
statements;
break;
default:
}
```

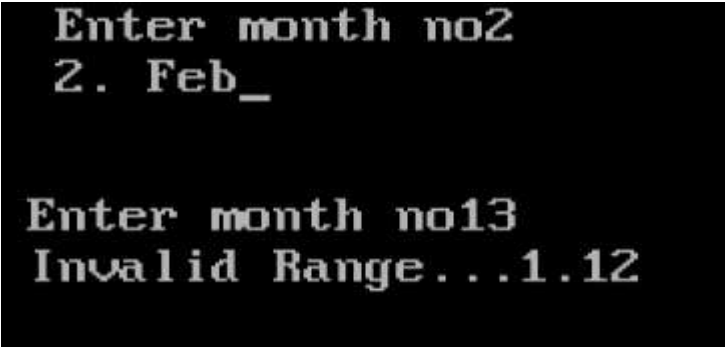
Example :

```
switch(month)
{
case 2:
    printf("2.Feb");
    break;
case 1:
    printf("1.Jan");
    break;
default:
    printf("Invalid...Range 1..12");
}
```

W. A.P to enter the month number, the program should give its name in words.

```
#include<stdio.h>
#include<conio.h>
main()
{
int m;
clrscr();
printf("Enter month no");
scanf("%d",&m);
switch(m)
{
case 1:
    printf("1. Jan");
    break;
case 2:
    printf("2. Feb");
    break;
case 3:
    printf("3. Mar");
    break;
default :
    printf("Invalid Range...1.12");
}
getch();
}
```

OUTPUT



```
Enter month no2
2. Feb_
```

```
Enter month no13
Invalid Range...1.12
```

W. A.P to print whether a character entered is a vowel or not , using switch case.

Topics Covered: Use of Switch Case

```
/* Documentation Section*/
```

```
/* Prog No */
```

```
/* Aim : Switch case*/
```

```
/* Link Section */
```

```
#include<stdio.h>
```

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

```
#include<ctype.h>
```

```
/*Definition section*/
```

```
#define pi 3.14
```

```
/*Declaration section */
```

```
main()
```

```
{
```

```
    char ch;
```

```
    char resp;
```

```
    clrscr();
```

```
    up:flushall();
```

```
    printf("\n Enter the character");
```

```
    scanf("%c",&ch); // format specifiers
```

```
    textcolor(YELLOW+BLINK);
```

```
    switch(tolower(ch))
```

```
{
```

```
    case 'a':
```

```
        cprintf("%c is a vowel\n",ch);
```

```
        break;
```

```
    case 'e':
```

```
        cprintf("%c is a vowel\n",ch);
```

```
        break;
```

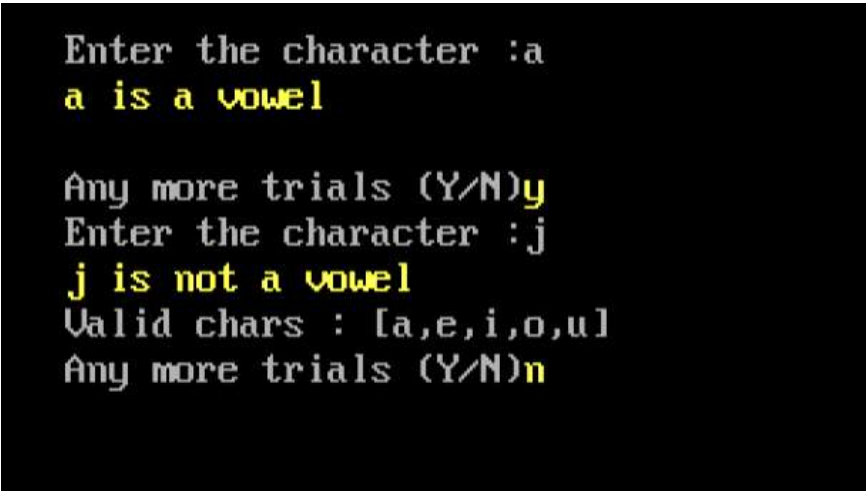
```
    case 'i':
```

```

    cprintf("%c is a vowel\n",ch);
    break;
case 'o':
    cprintf("%c is a vowel\n",ch);
    break;
case 'u':
    cprintf("%c is a vowel\n",ch);
    break;
default :
    cprintf("%c is not a vowel\n",ch);
    printf("Valid chars : [a,e,i,o,u]");
}
printf("\n Any more trials (Y/N)");
resp=getche();
if(resp=='Y' || resp=='y')
    goto up;
getch();
}

```

OUTPUT :



```

Enter the character :a
a is a vowel

Any more trials (Y/N)y
Enter the character :j
j is not a vowel
Valid chars : [a,e,i,o,u]
Any more trials (Y/N)n

```

Write a Menu driven program to convert Rs. To Dollars and Dollars to Rs. using switch case. Topics Covered: Use of Switch Case.

```

/* Documentation Section*/
/* Prog No  : Aim : Switch case*/
/* Link Section */
#include<stdio.h>
#include<conio.h>
#include<ctype.h>
#include<stdlib.h>

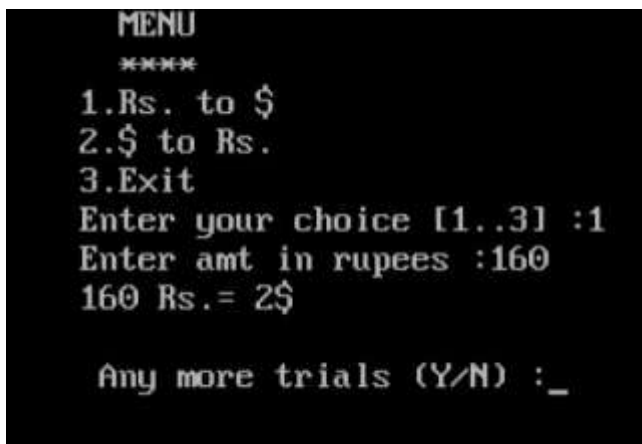
/*Definition section*/
/*Declaration section */
main()
{ int r,d,ch;
char resp;
up:clrscr();
flushall();
puts(" MENU");
puts(" ****");
puts("1.Rs. to $");
puts("2.$ to Rs.");
puts("3.Exit");
printf("Enter your choice [1..3] :");
scanf("%d",&ch);

switch(ch)
{ case 1:
printf("Enter amt in rupees :");
scanf("%d",&r);
d=r/80;
printf("%d Rs. = %d$ \n",r,d);
break;
case 2:
printf("Enter no of dollars :");
scanf("%d",&d);
r=d*80;
printf("%d $ = %d Rs \n",d,r);
break;
case 3:
exit(1);

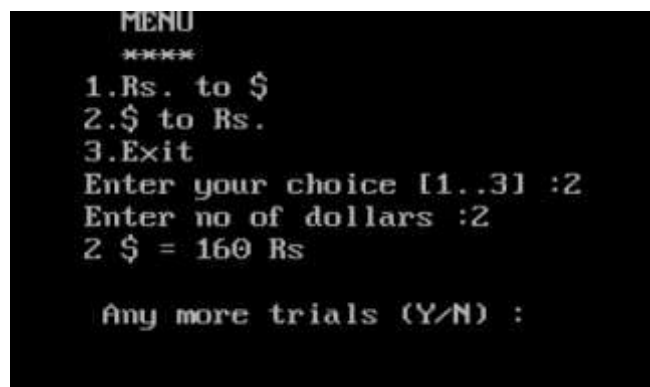
```

default:

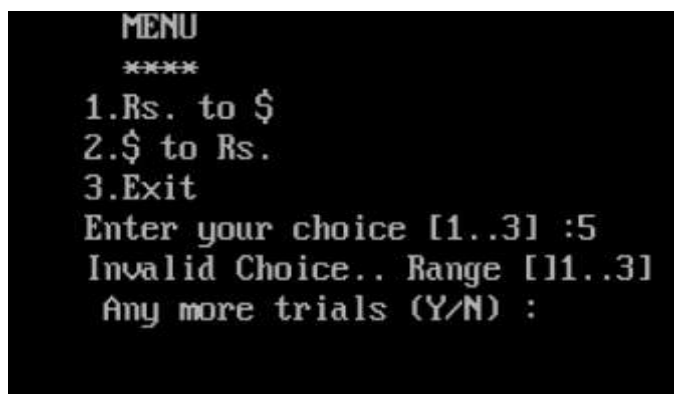
```
printf("Invalid Choice.. Range [1..3]");  
}  
printf("\n Any more trials (Y/N) :");  
resp=getche();  
if(resp=='Y' || resp=='y')  
    goto up;  
getch();  
}
```



```
MENU  
****  
1.Rs. to $  
2.$ to Rs.  
3.Exit  
Enter your choice [1..3] :1  
Enter amt in rupees :160  
160 Rs.= 2$  
  
Any more trials (Y/N) :_
```



```
MENU  
****  
1.Rs. to $  
2.$ to Rs.  
3.Exit  
Enter your choice [1..3] :2  
Enter no of dollars :2  
2 $ = 160 Rs  
  
Any more trials (Y/N) :
```



```
MENU  
****  
1.Rs. to $  
2.$ to Rs.  
3.Exit  
Enter your choice [1..3] :5  
Invalid Choice.. Range [1..3]  
Any more trials (Y/N) :
```

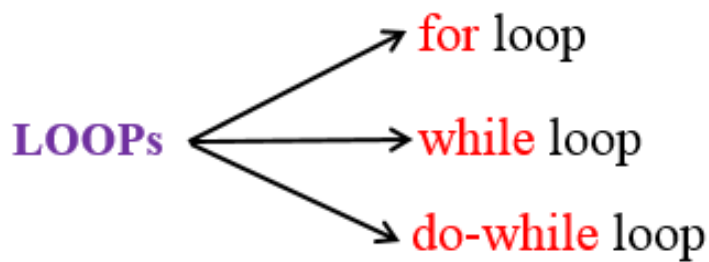
Concept of LOOPS:

Loops: For loop / do –while / while – do

Loop is something which is repeated again and again.

Loops are classified broadly into two types:

1. **Counter Controlled:** Some counter is set and the loop has to execute those many times. (for loop)
2. **Sentinel Controlled:** Some word such as END/STOP/9999 is used to terminate the loop. (do –while, while-do)



{list of statements}

Example : for(i=1;i<=10;i++)
 printf("YCMOU");

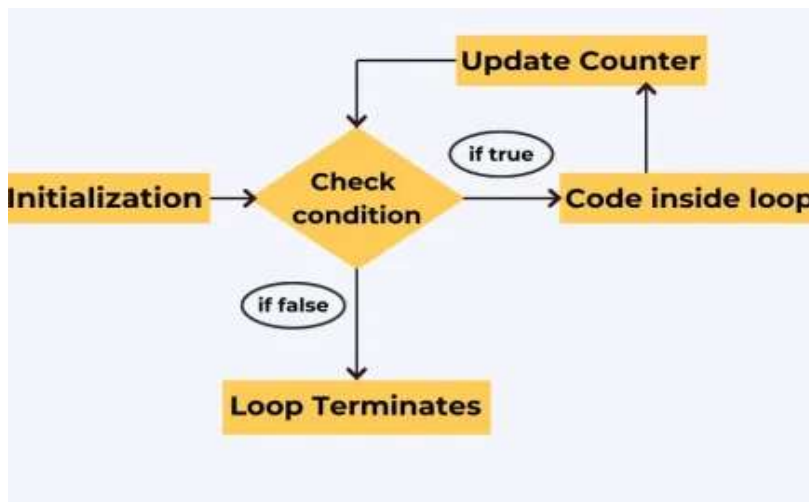
Concept of For Loop.

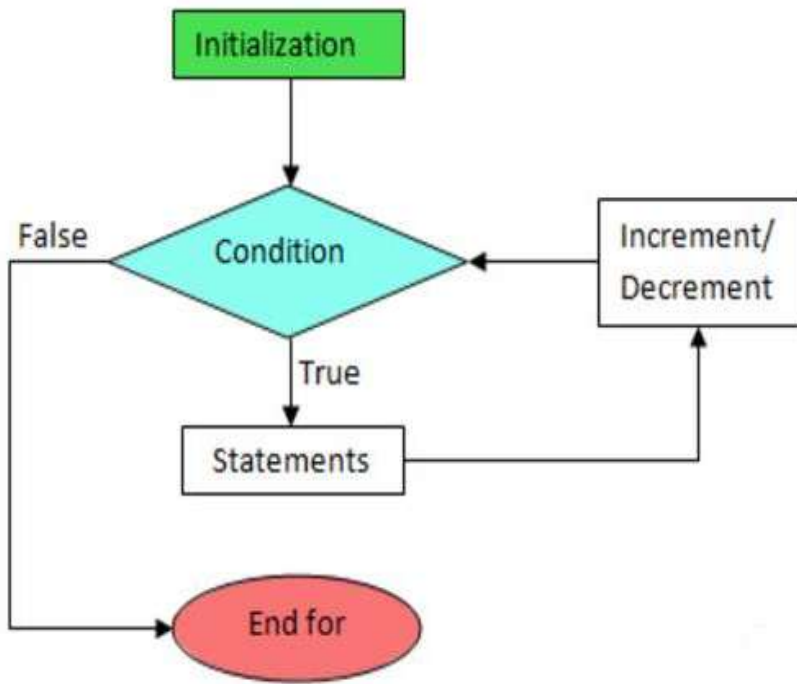
Use : Used to repeat some task again and again.

Syntax :

for (counter variable = initial value;
condition; increment/decrement)

Diagrammatic representation of FOR LOOP





W.A.P to print your name 10 times.

```
/* Documentation Section*/

/* Prog No  */

/* Aim : for loop*/

/* W.a.p to print your name 10 times */

/* Link Section */

#include<stdio.h>

#include<conio.h>

#include<ctype.h>

#include<stdlib.h> // for exit

/*Definition section*/

#define pi 3.14

/*Declaration section */

main()

{

char name[20]; // collection of characters

int i;

clrscr();

printf("Enter your name");

scanf("%s",name);

printf("10 times\n");

for(i=1;i<=10;i++)

printf("%s\t",name);

printf("5 times\n");

for(i=1;i<=10;i=i+2)

    printf("%s\t",name);

printf("\n5 times decrement\n");

for(i=10;i>=1;i=i-2)

    printf("%s\t",name);

getch();

}
```

```
Enter your name :John
10 times
John  John  John  John  John  John  John  John  John  John
5 times
John  John  John  John  John
5 times decrement
John  John  John  John  John
```

Concept of loops, if-else, flags, continue, break :

W.A.P to test whether a number is prime or not .

Algorithm :

1. Start
2. Read the number to be tested in the memory variable 'n'
3. Test whether the number is prime or not.
4. Display the answer in a suitable format.
5. Terminate.

/* Documentation Section */

/* Prog No 13 Aim : Prime or not */

/*Link Section */

#include<stdio.h>

#include<conio.h>

/*Definition Section*/

#define pi 3.14

/*Declaration*/

int n,d,r;

int isprime;

main()

{ clrscr();

/*Input */

printf("Enter the no");

scanf("%d",&n);

/*processing*/

isprime=1;

for(d=2;d<=n-1;d++)

{

r=n%d;

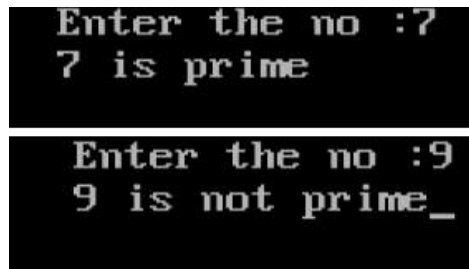
if(r==0)

{

isprime=0;

```
    printf("%d is not prime",n);  
  
    break;  
}  
else  
  
    continue;  
}  
if(isprime==1)  
    printf("%d is prime",n);  
getch();  
}
```

OUTPUT



```
Enter the no :7  
7 is prime  
-----  
Enter the no :9  
9 is not prime_
```

W.A.P to find sum and average of 'n' numbers.

```
/*Documentation section*/

/* Link Section */

#include<stdio.h>

#include<conio.h>

/*Definition Section*/

/*Global Declaration*/

main()

{

    int n, sum=0,i,num;

    float avg;

    clrscr();

    printf("Enter no of numbers :");

    scanf("%d",&n);

    for(i =1;i<=n;i++)

    {

        printf("Enter the element :");

        scanf("%d",&num);

        sum=sum+num;

    }

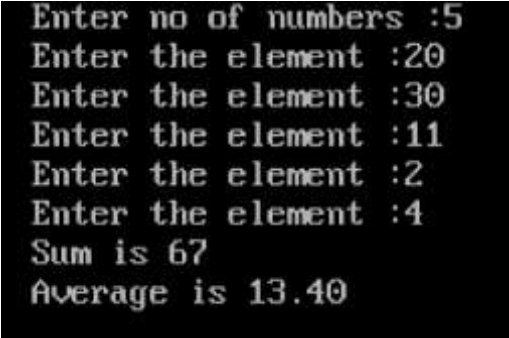
    avg = (float)sum/n; // typecasting

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

    printf("Average is %.2f\n",avg);

    getch();

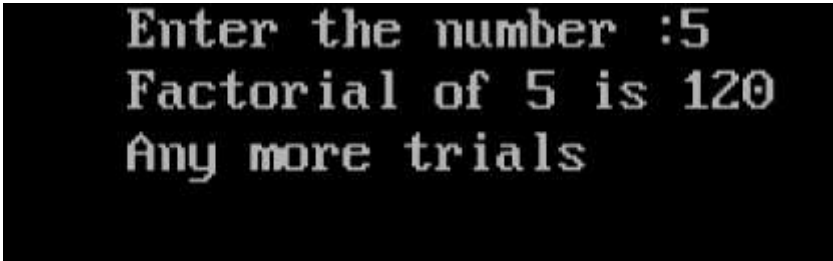
}
```

A screenshot of a terminal window with a black background and green text. It shows the execution of the C program. The user enters 5 for the number of elements. Then, five elements are entered: 20, 30, 11, 2, and 4. The program outputs the sum as 67 and the average as 13.40.

```
Enter no of numbers :5
Enter the element :20
Enter the element :30
Enter the element :11
Enter the element :2
Enter the element :4
Sum is 67
Average is 13.40
```

W.A.P to find factorial of a number

```
/* 5! = 1.2.3.4.5 = 120 */  
#include<stdio.h>  
#include<conio.h>  
#include<ctype.h>  
main()  
{  
    int num,i;  
    long int p;  
    char resp;  
    l1:clrscr();  
    p=1;  
    printf("Enter the number");  
    scanf("%d",&num);  
    for(i=1;i<=num;i++)  
        p=p*i;  
    printf("Factorial of %d is %ld\n",num,p);  
  
    printf("Any more trials");  
    resp=getche();  
    if(toupper(resp)=='Y')  
        goto l1;  
    getch();  
}
```



Enter the number :5
Factorial of 5 is 120
Any more trials

/*W.A.P to print table of a number */

Topics Covered: Concept of for loop.

```
#include<stdio.h>
```

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

```
#define mr 12
```

```
main()
```

```
{ int n,i;
```

```
    char resp;
```

```
l1:clrscr();
```

```
printf("Enter the number");
```

```
scanf("%d",&n);
```

```
for(i=1;i<=mr;i++)
```

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

```
printf("Any more trials");
```

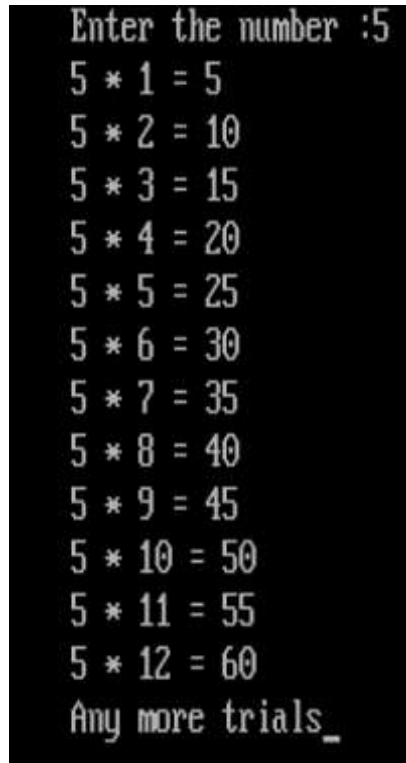
```
resp=getche();
```

```
if(toupper(resp)=='Y')
```

```
    goto l1;
```

```
getch();
```

```
}
```



```
Enter the number :5
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
5 * 6 = 30
5 * 7 = 35
5 * 8 = 40
5 * 9 = 45
5 * 10 = 50
5 * 11 = 55
5 * 12 = 60
Any more trials_
```

/*W.a.p to print the following pattern */

```
*
**
***
****
*****

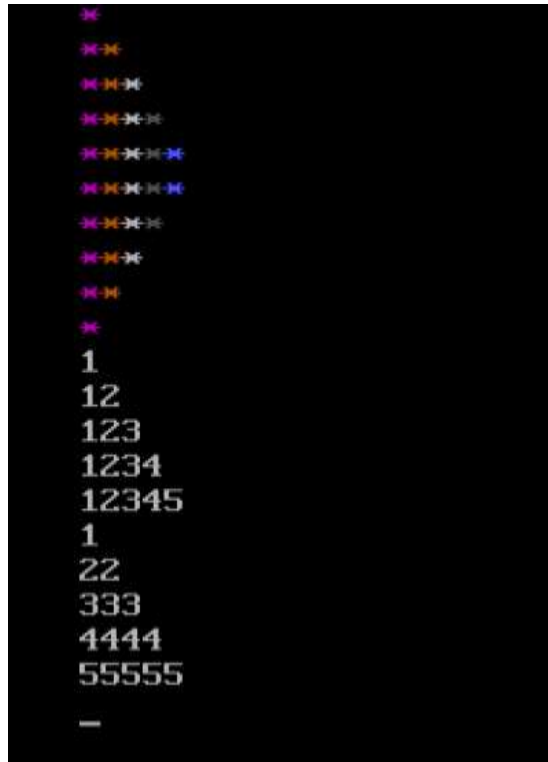
#include<stdio.h>
#include<conio.h>
main()
{ int i,j; clrscr();
  for(i=1;i<=5;i++)
  {   for(j=1;j<=i;j++)
      { textcolor(RED+j);
        cprintf("*");
      }
    printf("\n");
  }

  for(i=5;i>=1;i--)
  {
    for(j=1;j<=i;j++)
    {
      textcolor(RED+j);
      cprintf("*");
    }
    printf("\n");
  }
```

```

    }
for(i = 1;i<=5;i++)
{
    for(j=1;j<=i;j++)
        printf("%d",j);
    printf("\n");
}
for(i = 1;i<=5;i++)
{
    for(j=1;j<=i;j++)
        printf("%d",i);
    printf("\n");
}
getch();
}

```



LOOPS : ‘ do –while’ and ‘while-do’ loops

Do While: (Bottom tested/Exit controlled) (Unit 05)

Use : In this ,the task is performed first and the condition is checked at the end.

Syntax :

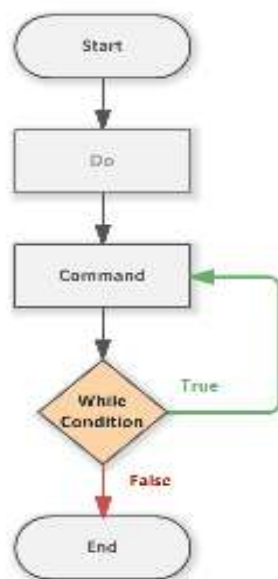
do

{

list of statements;

} while (condition);

DO-WHILE



While-do : (Top tested/Entry Controlled)

Use : In this ,the condition is checked first and then the task is performed.

Syntax :

```
while(condition)
```

```
{
```

```
    list of statements;
```

```
}
```

Example :

```
int i = 1;
```

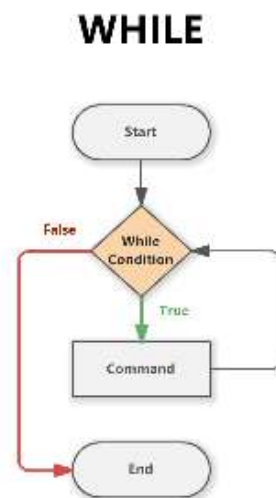
```
while(i<=10)
```

```
{
```

```
    printf("YCMOU");
```

```
    i=i+1;
```

```
}
```



Comparison of while –do and do-while loop:

Aspect	While Loop	Do-While Loop
Condition Check	Beginning	End
Minimum Executions	0	1
Syntax (C-style)	while (condition) { ... }	do { ... } while (condition);
Variable Initialization	Variables are initialised before the execution of the loop.	Variables may initialise before or within the loop.
Exit Control	Pre-test	Post-test
Execution Guarantee	No guarantee	At least once
Semicolon	Not required after }	Required after condition
Initialization	Before loop	Can be within the loop body
Use Case	When you need to check the condition before any execution	When you want to ensure at least one execution
Example Scenario	Checking if a file exists before processing	Getting user input at least once

Choosing the Right Loop:

- If you know the number of iterations and want a structured approach, use for.
- If you want to execute the loop body at least once, use do-while.
- if you have an unknown number of iterations or a long-running condition, use while

W.A.P to find sum and average of 'n' numbers using do -while.

/* Documentation Section */

/* Prog No 18 Aim : Use of do while */

/*Link Section */

#include<stdio.h>

#include<conio.h>

/*Definition Section*/

/*Declaration*/

main()

{

int n, num, sum=0,i=1;

float avg;

clrscr();

printf("\nEnter no of numbers :");

scanf("%d",&n);

printf("Enter data..\n");

do

{

printf("\nEnter the number :");

scanf("%d",&num);

sum = sum+num;

i = i+1;

}while(i<=n);

avg = (float)sum/n;

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

printf("Average is : %.2f\n",avg);

getch();

}

Enter no of numbers :5

Enter data..

Enter the number :10

Enter the number :20

Enter the number :30

Enter the number :40

Enter the number :50

Sum is : 150

Average is : 30.00

W.A.P to reverse a number and decide whether it is palindrome or not.

(Using while –do)

Logic behind reversing a number.

n= 123 , rev=0

d = n%10 123%10 = 3

rev = rev*10+d 0+3 = 3

n= n/10 123/10 = 12.3 = 12

d = n%10 12%10 = 2

rev = rev*10 +d 3*10 +2 = 32

n = n/10 12/10 = 1.2 = 1

d = n%10 1%10 = 1

rev = rev*10+d 32 * 10 + 1 = 321

n = n/10 1/10 = 0.1 = 0

/*Documentation Section*/

/* Program No 19 : Use of while do. To reverse a number */

/* Also decide whether it is Palindrome or not */

/*Link Section*/

#include<stdio.h>

#include<conio.h>

/*Definition Section*/

/*Global declaration*/

main()

{

int n, rev, d,temp;

char resp;

do

{

clrscr();

printf("Enter the number ");

scanf("%d",&n);

temp = n;

rev=0;

while(n!=0)

{ d = n%10;

rev = rev*10+d;

n =n/10;

}

printf("Original number is : %d \n",temp);

printf("Its reverse is : %d \n",rev);

if(rev==temp)

printf("%d is palindrome.",temp);

else

printf("%d is not palindrome.",temp);

printf("Any more trials (Y/N)");

resp=getche();

```
}while(toupper(resp)!='Y');  
printf("Press any key to exit..");  
getch(); }
```

```
Enter the number 123  
Original number is : 123  
Its reverse is      : 321  
123 is not palindrome.  
Any more trials (Y/N) y  
Enter the number 121  
Original number is : 121  
Its reverse is      : 121  
121 is palindrome.  
Any more trials (Y/N) n  
Press any key to exit..
```

W.A.P to print Fibonacci sequence .

In mathematics, the Fibonacci sequence is **a sequence in which each number is the** sum of the two preceding ones.

The list of first 20 terms in the Fibonacci Sequence is: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181.

```
/*Documentation Section*/
```

```
/*Program No 20 : Use of while do. To generate Fibonnaci sequence */
```

```
/*Link Section*/
```

```
#include<stdio.h>
```

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

```
/*Definition Section*/
```

```
/*Global declaration*/
```

```
main()
```

```
{
```

```
int i,f1,f2,f3,n;
```

```
char resp;
```

```
clrscr();
```

```
do
```

```
{ f1=0;f2=1;
```

```
printf("\nEnter the number of terms ");
```

```
scanf("%d",&n);
```

```
i = 1;
```

```
do
```

```
{ if(i==1)
```

```
printf("%d ",f1);
```

```
else
```

```
if(i==2)
```

```
printf("%d ",f2);
```

```
else
```

```
{
```

```
f3=f1+f2;
```

```
printf("%d ",f3);
```

```
f1=f2; f2=f3;
```

```

        } i++;

    }while(i<=n);

printf("\nAny more trials (Y/N)");

    resp=getche();

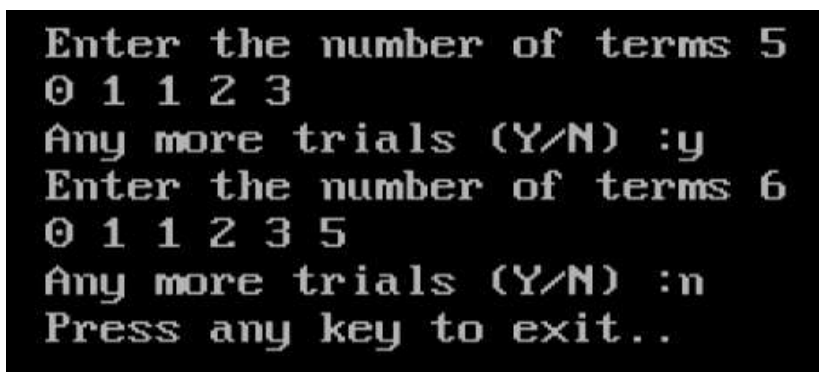
}while(toupper(resp)=='Y');

printf("\nPress any key to exit..");

getch();

}

```



```

Enter the number of terms 5
0 1 1 2 3
Any more trials (Y/N) :y
Enter the number of terms 6
0 1 1 2 3 5
Any more trials (Y/N) :n
Press any key to exit..

```

Difference between Continue and Break:

	Break	Continue
1	Used to terminate the loop.	Used to continue to the next iteration.
2	Used in loops and with switch case.	Used only in loops.
3	Control is transferred outside the loop	Control remains in the same loop.
4	In this the flow control of the loop is interrupted entirely.	In this only the current iterations flow is interrupted.
5	Syntax : break;	Syntax : continue;

SUMMARY :

This chapter stressed on various Decision making constructs. Both Controlled and Uncontrolled constructs were discussed. Goto is an uncontrolled construct. Decision making constructs such as if -else , nested if, Conditional Operator were discussed. Various loops such as for-loop, do-while, while-do were studied. The keyword break, continue were also discussed.

SOLVED PROBLEMS AND SELF CHECK QUESTIONS

SOLVED PROBLEMS

- 1) Iterative constructs are specified by means of :
 - a) If statements
 - b) Loops**
 - c) Switch case
 - d) Nested ifs.
- 2) ____ statement is used for bypassing the loop statements which has yet not executed unconditionally and directly transfer the control to the beginning of the loop.
 - a) continue**
 - b) next
 - c) break
 - d) goto

SELF-TEST 01

MCQ 01: The expression to check if 'a' is not equal to 100 is :

- a) If (a<>100)
- b) If (a!=100)**
- c) If (a NOT 100)
- d) If (a NOT EQUALS 100)

MCQ 02: The expression: a= 14; if(a) would return ____

- a) TRUE**
- b) FALSE

MCQ 03 : Conditional Operator is also known as _____

- a) Binary operator.
- b) Ternary operator.**
- c) Unary Operator.
- d) Logical Operator.

MCQ 04: The default statement is _____

- a) Optional.**
- b) Mandatory.

MCQ 05 : The____ statement is a multiway decision maker.

- a) If.

- b) Nested if.
- c) **switch.**
- d) While do.

MCQ 06 : The ____ loop is also known as top tested loop.

- a) do-while.
- b) **while - do**
- c) for.
- d) All of the above.

MCQ 07 : The continue statement skips the statements after it and re-enters the loop after evaluating the condition.

- a) **TRUE**
- b) FALSE

MCQ 08 : The exit command jumps out of the innermost if or an else_

- a) TRUE
- b) **FALSE**

MCQ 09 : The ____ statement explicitly takes the control of the program where one wants.

- a) switch
- b) **goto**
- c) redirect
- d) exit

MCQ 10 : ____ loop will execute atleast once.

- a) while-do
- b) **do-while**

EXERCISE

- Q1. Write the use, syntax and an example of nested if statement.
- Q2. What role is played by the default label in a switch statement.
- Q3. What is the essential difference between a while and do while loop.
- Q4. Explain the ternary operator with an example.
- Q5. Write the difference between continue and break.

REFERENCE MATERIAL

REFERENCES : [GEEKSFORGEEKS.COM](https://www.geeksforgeeks.com), [UNSTOP.COM](https://unstop.com)

MOOCs

YOUTUBE VIDEOS

WIKIPEDIA

BLOGS : [UNSTOP.COM](https://unstop.com)

OER (SLIDES, PDFs, TUTORIALS)

REFERENCE BOOKS

ANSWERS TO SELF-CHECK QUESTIONS

- 1.(b) 2. (a) 3.(b) 4.(a) 5.(c)
- 6.(b) 7.(a) 8.(b) 9.(b) 10.(b)

STUDYING UNIT 7: ARRAYS

UNIT OBJECTIVES

After successful completion of this unit, you will be able to understand:

- Understand the meaning of Arrays.
- Identify different types of Arrays.
- Explain the uses of Arrays.
- Recognize the limitations of Arrays.
- Declare and initialize one-dimensional, two-dimensional, and character arrays.
- Access elements of arrays effectively.

7.1 Concept of Arrays (Subscripted Variables)

Array: It is a basic data structure.

Definition:

It is a collection of similar data type elements allocated contiguous memory blocks referred by a common name.

Consider the variable declaration: - `int m1, m2, m3;`

Variables m1,m2,m3 will be allocated memory randomly.

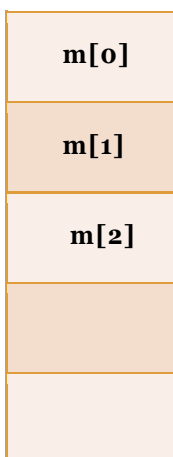
Due to random allocation processing of elements will be slow.



If done like this: `int m [3];`

Here all three variables will be allocated contiguous memory blocks.

Due to contiguous allocation processing of elements will be faster.



Types:

1. **One dimensional Array:** `int marks[10], roll[10];`

`m[0] m[1] m[2] m[3] m[4] m[5] m[6] m[7] m[8] m[9]`

10	20	30	40	50	60	70	80	90	100
----	----	----	----	----	----	----	----	----	-----

2. **Two dimensional Array:** `int m[2][3];`

m[0][0]	m[0][1]	m[0][2]
m[1][0]	m[1][1]	m[1][2]

3. **'n' dimensional:** `int m[2][2][3]; int m[2][2][2][3];`

m[0][0][0]	m[0][0][1]	m[0][0][2]
M[0][1][0]	m[0][1][1]	m[0][1][2]

m[1][0][0]	m[1][0][1]	m[1][0][2]
m[1][1][0]	m[1][1][1]	m[1][1][2]

Applications:

1. Surveys .
2. Statistical Analysis.
3. Matrices Operations.
4. Sorting and Filtering lists
5. Searching elements within a list.

Advantage:

Processing is faster.

Limitations:

Being Static memory space may be underutilized
Also once declared one cannot demand more.

/* W.a.p to create 1D array.Print it in original and reverse order.Also find max,min and sum*/

```
#include<stdio.h>
#include<conio.h>
#define mx 200
int m[mx];
main()
{
    int noe,i,max,min,sum=0;
    clrscr();
    printf("Enter no of elements");
    scanf("%d",&noe);
    printf("Enter data\n");
    for(i=0;i<noe;i++)
    {
        printf("Enter element ");
        scanf("%d",&m[i]);
    }
    max=m[0]; min=m[0];
    for(i=0;i<noe;i++)
    {
        sum=sum+m[i];
        if(m[i]>max)
            max=m[i];
        if(m[i]<min)
            min=m[i];
    }
    printf("Array is \n");
    for(i=0;i<noe;i++)
        printf("%d\t",m[i]);
    printf("\nArray in reverse order \n");
    for(i=noe-1;i>=0;i--)
        printf("%d\t",m[i]);
    printf("\nSum is :%d\n",sum);
    printf("Largest element is :%d\n",max);
    printf("Smallest element is:%d\n",min);
    getch();
}
```

```

Enter no of elements5
Enter data
Enter element 1
Enter element 3
Enter element 2
Enter element 4
Enter element 5
Array is
1      3      2      4      5
Array in reverse order
5      4      2      3      1
Sum is :15
Largest element is :5
Smallest element is:1

```

// W.a.p for voting Analysis P22.c

```

#include<stdio.h>
#include<conio.h>
#include<ctype.h>
#define mx 100
int vote[mx];
main()
{
int i,nov=0,p1ctr=0,p2ctr=0,p3ctr=0,invctr=0;
char resp;
clrscr();
/*printf("Enter no of voters");
scanf("%d",&nov);
tot=nov;
*/
printf("Enter data\n");
do{
nov++;
printf("Cast your vote[1/2/3]");
scanf("%d",&vote[i]);
if(vote[i]==1)
p1ctr++;
else
if(vote[i]==2)

```

```

    p2ctr++;
else
    if(vote[i]==3)
        p3ctr++;
else
    invctr++;
printf("Any more voters Y/N \n");
flushall();
resp=getchar();
}while(toupper(resp)=='Y');
printf("Statistical Analysis\n");
printf("PARTY1 PARTY2 PARTY3 INVALID TOTAL\n");
printf(" %d %d %d %d %d\n",p1ctr,p2ctr,p3ctr,invctr,nov);
printf("-----\n");
getch();}

```



```

Enter data
Cast your vote(1/2/3)1
Any more voters Y/N
y
Cast your vote(1/2/3)1
Any more voters Y/N
y
Cast your vote(1/2/3)2
Any more voters Y/N
y
Cast your vote(1/2/3)2
Any more voters Y/N
y
Cast your vote(1/2/3)3
Any more voters Y/N
y
Cast your vote(1/2/3)4
Any more voters Y/N
n
Statistical Analysis
PARTY1 PARTY2 PARTY3 INVALID TOTAL
2      2      1      1      6
-----

```

/* W.a.p to create 2D array and print */

```

#include<stdio.h>
#include<conio.h>
#define mr 10
#define mc 10
int m[mr][mc];
main()
{
    int r,c, i,j,sum=0;
    clrscr();
    printf("Enter no of rows");
    scanf("%d",&r);
    printf("Enter no of cols");

```

```

scanf("%d",&c);
printf("Enter data :");
for(i=0;i<r;i++)
{
    for(j=0;j<c;j++)
    {
        printf("Enter element : ");
        scanf("%d",&m[i][j]);
        sum = sum+m[i][j];
    }
}
printf("Array is \n");
for(i=0;i<r;i++)
{
    for(j=0;j<c;j++)
        printf("%d ", m[i][j]);
    printf("\n");
}
printf("Sum is %d\n",sum);
getch();
}

```

```

Enter no of rows2
Enter no of cols3
Enter data :Enter element : 1
Enter element : 2
Enter element : 3
Enter element : 4
Enter element : 5
Enter element : 6
Array is
1 2 3
4 5 6
Sum is 21

```

//W.a.p to create a 2d matrix and generate its transpose

```
#include<stdio.h>
#include<conio.h>
#define mr 10
#define mc 10
int m[mr][mc];
int t[mr][mc];
main()
{
    int r,c,i,j;
    clrscr();
    printf("Enter no of rows");
    scanf("%d",&r);
    printf("Enter no of cols");
    scanf("%d",&c);
    printf("Enter data\n");
    for(i=0;i<r;i++)
    {
        for(j=0;j<c;j++)
        {
            printf("Enter element");
            scanf("%d",&m[i][j]);
            t[j][i]=m[i][j];
        }
    }
    printf("Original Array is \n");
    for(i=0;i<r;i++)
    {
        for(j=0;j<c;j++)
        printf("%3d ",m[i][j]);
        printf("\n");
    }
    printf("Transposed Array is \n");
    for(i=0;i<c;i++)
    {
        for(j=0;j<r;j++)
        printf("%3d ",t[i][j]);
        printf("\n");
    }
}
```

```
}  
getch();  
}
```

```
Enter no of rows2  
Enter no of cols3  
Enter data  
Enter element1  
Enter element2  
Enter element3  
Enter element4  
Enter element5  
Enter element6  
Original Array is  
  1  2  3  
  4  5  6  
Transposed Array is  
  1  4  
  2  5  
  3  6
```

/*W.a.p to create 2 matrices add and subtract them */

```
#include<stdio.h>
#include<conio.h>
#define mr 10
#define mc 10
int m1[mr][mc],m2[mr][mc],m3[mr][mc],m4[mr][mc];
main()
{
int r,c,i,j;
clrscr();
printf("Enter no of rows");
scanf("%d",&r);
printf("Enter no of cols");
scanf("%d",&c);
printf("Enter data for Matrix A\n");
for(i=0;i<r;i++)
{
for(j=0;j<c;j++)
{
printf("Enter element [%d][%d]",i+1,j+1);
scanf("%d",&m1[i][j]);
}
}
printf("Enter data for Matrix B\n");
for(i=0;i<r;i++)
{
for(j=0;j<c;j++)
{
printf("Enter element [%d][%d]",i+1,j+1);
scanf("%d",&m2[i][j]);
}
}
for(i=0;i<r;i++)
{
for(j=0;j<c;j++)
{
m3[i][j]=m1[i][j]+m2[i][j];
m4[i][j]=m1[i][j]-m2[i][j];
```

```

}
}
printf("Matrix A is \n");
for(i=0;i<r;i++)
{
for(j=0;j<c;j++)
printf("%3d ",m1[i][j]);
printf("\n");
}
printf("Matrix B is \n");
for(i=0;i<r;i++)
{
for(j=0;j<c;j++)
printf("%3d ",m2[i][j]);
printf("\n");
}
printf("Matrix C is \n");
for(i=0;i<r;i++)
{
for(j=0;j<c;j++)
printf("%3d ",m3[i][j]);
printf("\n");
}
printf("Subtracted Matrix D is \n");
for(i=0;i<r;i++)
{
for(j=0;j<c;j++)
printf("%3d ",m4[i][j]);
printf("\n");
}
getch();
}

```

```
Enter no of rows2
Enter no of cols2
Enter data for Matrix A
Enter element [1][1]1
Enter element [1][2]2
Enter element [2][1]3
Enter element [2][2]4
Enter data for Matrix B
Enter element [1][1]5
Enter element [1][2]6
Enter element [2][1]7
Enter element [2][2]8
Matrix A is
  1  2
  3  4
Matrix B is
  5  6
  7  8
Matrix C is
  6  8
 10 12
Subtracted Matrix D is
 -4 -4
 -4 -4
```

SUMMARY :

To store the similar data contiguously in the memory so that the processing is faster we studied the concept of arrays. Arrays are also called as subscripted variables. Different types of arrays such as One dimensional, Two Dimensional and 'n' Dimensional were discussed. Applications and Limitations of arrays were also studied.

SOLVED PROBLEMS AND SELF CHECK QUESTIONS

SOLVED PROBLEMS

- 1) Array is a collection of ____ data type elements:
 - a) dissimilar
 - b) similar**
 - c) unique
 - d) all of the above.
- 2) Three dimensional arrays are also known as ____ dimensional arrays.
 - a) 'n'**
 - b) Two
 - c) three
 - d) Jagged.

SELF-TEST 01

MCQ 01: Arrays are ____ type of data structure.

- a) dynamic
- b) Static**
- c) volatile
- d) persistent

MCQ 02: Arrays are also known as subscripted variables.

- a) TRUE**
- b) FALSE

MCQ 03 : Arrays are used for ____

- a) Surveys
- b) Statistical analysis
- c) Matrix operations.
- d) All of the above.

MCQ 04: Address of an array can be calculated in ____ ways.

- a) Row major.
- b) Column major.
- c) Both a & b
- d) Offset method.

MCQ 05 : Array can be passed as a parameter to a function.

- a) TRUE
- b) FALSE

EXERCISE

- Q1. What is an ARRAY? explain with an example.
- Q2. Enlist the applications of an array.
- Q3. What are the limitations of an Array, how can it be overcome.
- Q4. Discuss different types of arrays with diagrams and snippets.
- Q5. Explain Row Major and Column Major representations of an array.
- Q6. What is the difference between an ordinary variable and an array.
- Q7. Explain the concept of base address of an array.
- Q8. Write a program to test whether a square matrix is a magic square or not.

REFERENCE MATERIAL

REFERENCES : [GEEKSFORGEEKS.COM](https://www.geeksforgeeks.com/), [UNSTOP.COM](https://www.unstop.com/)

[HTTPS://WWW.W3SCHOOLS.COM/C/](https://www.w3schools.com/c/)

MOOCs

YOUTUBE VIDEOS

WIKIPEDIA

BLOGS : [UNSTOP.COM](https://www.unstop.com/)

OER (SLIDES, PDFs, TUTORIALS)

REFERENCE BOOKS

ANSWERS TO SELF-CHECK QUESTIONS

- 1.(b) 2. (a) 3. (d) 4. (c) 5. (a)

UNIT 8: STRINGS

UNIT OBJECTIVES

After successful completion of this unit, you will be able to

- ❖ Declaration and initialization of string variables
- ❖ String handling functions from standard library

8.1 Declaration and initialization of string variables

String: It is a collection of characters. There are many String functions to handle strings. The header file required for these functions is string.h

A string literal or string constant is a sequence of zero or more characters enclosed within “ ”. A string is an array of characters terminated by a special character called NULL character ('\0').

Example

“Welcome to C” is a string and it is stored in memory as:

W	e	I	c	o	m	e		t	o		C	\0
1000												1012

Each character is stored as 1 byte as its ASCII code. Since the string is stored as an array, it is possible to access individual characters.

1. Declaring and Initializing String Variables

Since a string is a character array, it can be stored in a string variable which is declared as follows.

char stringname [length];

This length determines the maximum number of characters in the string.

Examples: char country [10] ; char name [20] ;

There are two ways to initialize strings.

- Character by character, separated by commas

Example: char city [] = { 'p', 'u', 'n', 'e', '\0' } ;

- Set of characters enclosed in “ ”

Example: char city [] = “pune”;

The compiler automatically stores the null character at the end of the string.

We can also use a pointer to point to a string. Consider the following two Declarations.

```
char amessg [] = “C programming language”;    /*array */
```

```
char *pmessg= “C programming language “;      /* pointer */
```

amessg

C programming language \0



amessage is an array big enough to hold the sequence of characters and '\0'. Even if the characters are later changed, amessage will always refer to the same storage. On the other hand, pmessage is a pointer, which points to a string constant. pmessage may be later made to point elsewhere.

8.2 String Input/Output

We have already seen two string i/o functions-gets and puts. Similarly the functions printf and scanf can also be used with the format specifier %s. The scanf function does not allow a string with embedded spaces. gets() allows a string to contain spaces.

Example: A snippet to accept the name of a person and display a greeting.

```
char name [80] ;
printf ( "Enter your name" ) ;
gets (name) ;
printf ( "Good Morning %s", name ) ;
```

sprintf and sscanf

- **sprintf ()** is the same as printf except that the output is written into a string rather than displayed on the output device. The return value is equal to the number of characters actually placed into the array.

The string is terminated with '\0' and it must be long enough to hold the result.

Syntax:

```
int sprintf (char * buffer, char * format, argument list);
```

Example

```
char str [80] ;
sprintf ( str, "%s %d %f", "Hello", 2, 5.0 ) ;
```

In this case, Hello 2 5.0 will be stored in the string str.

- **sscanf()** is identical to scanf except that data is read from the string pointed to by buf rather than stdin. The return value is equal to the number of fields that were actually assigned values.

Syntax:

```
int sscanf ( char *buf , char *format ,arg_list ) ;
```

Example

```
char s [20];
int n1,n2 ;
sscanf ( "hello 10 20", "%s %d", s , &n1, &n2 ) ;
```

This assigns hello to string s and 10 and 20 to n1 and n2 respectively.

Format Specifiers used with Strings:

- %s format specifier to accept and display a string. One can control the width and justification of the displayed string.

```
% [-] [^\n] s
```

Where:

- - is used for left justifying the output

[^\\n] - is used to accept a string which even contains white spaces.

Example:

```
char message [ ] = "Hello";  
printf ("%10s",message);      // displays Hello in a total width of 10  
printf ("% -10s", message);   // total width of 10 and left justified
```

8.3 Standard Library Functions

C provides of large number of string handling functions in the header file string.h. The following table lists some of the commonly used functions. Each function accepts one or more strings as an argument. Most of the functions return a string. Table 8.1 describes the various standard library functions.

Name	Prototype	Description
strlen	size_t strlen (char * s)	Returns the length of the string (number of characters excluding the NULL character).
strcpy	char* strcpy(char *d, char *s)	Copies the contents of string s to d and returns pointer to d.
strcat	char* strcat(char* s1,char*s2)	Concatenates a copy of s2 to s1 and terminates s1 with a null. Returns s1.
strcmp	int strcmp(char*s1,char*s2)	Compares s1 and s2 and returns, -ve if s1 is less than s2 0 if s1 is equal to s2 +ve if s1 is greater than s2
strcmpi	int strcmp(char*s1,char*s2)	Compares s1 and s2 ignoring the case and returns similar results as strcmp.
strlwr	char *strlwr(char *s)	Converts a string pointed to by s to lowercase.
strupr	char *strupr(char *s)	Converts a string pointed to by s to uppercase.
strncat	char *strncat(char *s1,char *s2, int n)	Concatenates first n characters of s2 to s1 and returns s1. S2 is unchanged.
strrev	char *strrev(char *s)	Reverses the string s and returns the reversed string.
strchr	char *strchr(char *s,char ch)	Returns a pointer to the first occurrence of character ch in strings s.
strstr	char *strstr(chars,char *s2)	Returns a pointer to the first occurrence of s2 in s1. Returns null if no match is found.

strset	char * strset(char *s,char ch)	Sets all characters in pointed to by s to the value of ch.
strnset	char *strnset(char *s,char ch,int n)	Sets the first n characters of string s to the value of ch.
atoi	int atoi(char *s)	Converts a string pointed to by s into an integer, returning the result. Similarly there is atoll and atof.
strncmp	int strncmp(char*s1,char*s2,int n)	Compares first n characters of s1 and s2. Returns <0 if s1 is less than s2, 0 if they are the same, >0 if s1 is greater than s2.
strdup	char *strdup(char *s)	Duplicates a string at another location and returns NULL if space could not be allocated or returns a pointer to the storage location containing duplicated string.
strchr	char *strchr(char *s, int c)	Returns a pointer to the last occurrence of character c in string s, NULL if not found.
strtok	char *strtok(char *s1,char *s2)	Searches s1 for tokens that are separated by delimiters in s2. Returns the pointers to the first character of first token in s1.

Table 8.1

Exploring Frequently used Library Functions with Snippets :

- 1. strlen() :** The strlen() function returns the length of the given string. It doesn't count null character '\0'.

Example :

```
#include<stdio.h>
#include <string.h>
void main()
{
char ch[20]={ 'j', 'a', 'v', 'a', 't', 'p', 'o', 'i', 'n', 't', '\0 '};
printf("Length of string is: %d",strlen(ch));
```

```

}
1. Length of string is: 10
2g is: 1

```

Output:

Length of string is: 10

2. strcpy()

The strcpy(destination, source) function copies the source string in destination. . . string is: 10

Example:

```

#include<stdio.h>
#include <string.h>
void main()
{
char ch[20]={ 'j', 'a', 'v', 'a', 't', 'p', 'o', 'i', 'n', 't', '\0 '};
char ch2[20];
strcpy(ch2,ch);
printf("Value of second string is: %s",ch2);
}

```

Output:

Value of second string is: javatpoint

3. strcat()

The strcat(first_string, second_string) function concatenates two strings and result is returned to first_string.

Example:

```

#include<stdio.h>
#include <string.h>
void main()
{
char ch[10]={ 'h', 'e', 'l', 'l', 'o', '\0 '};
char ch2[10]={ 'c', '\0'};
strcat(ch,ch2);
printf("Value of first string is: %s",ch); }

```

Output:

Value of first string is: helloc

4. strcmp()

The strcmp(first_string, second_string) function compares two string and returns 0 if both strings are equal.

Example :

```

#include<stdio.h>
#include <string.h>
void main()
{
char str1[20],str2[20];
printf("Enter 1st string: ");
gets(str1);//reads string from console

```

```

printf("Enter 2nd string: ");
gets(str2);
if(strcmp(str1,str2)==0)
printf("Strings are equal");
else
printf("Strings are not equal");
}

```

Output :

```

Enter 1st string: hello
Enter 2nd string: hello
Strings are equal

```

5. strcmpi()

The **strcmpi()** function is a built-in function in C and is defined in the "string.h" header file. The strcmpi() function is same as that of the strcmp() function but the only difference is that strcmpi() function is not case sensitive and on the other hand strcmp() function is the case sensitive.

```
int strcmpi (const char * str1, const char * str2 );
```

str1: The first string.

str2: The second string.

Returns: This function returns 0 if the given two strings are same, a negative value if the length of str1 is less than the length of str2 and if the length of str1 is greater than str2 then this function returns a positive value.

Example :

```

#include <stdio.h>
#include <string.h>
void main( )
{
char str1[] = "geeks" ;
char str2[] = "geeks" ;
int j = strcmpi ( str1, str2 );
printf ( "The function returns = %d",j );
}

```

Output :

The function returns = 0

```
#include <stdio.h>
#include <string.h>
void main( )
{
char str1[ ] = "geeks" ;
char str2[ ] = "ForGeeks" ;
int i = strcmpi ( str1, str2 ) ;
printf ( "The function returns = %d", i ) ;
}
```

Output:

The function returns = 1

6 . strrev()

The strrev(string) function returns reverse of the given string.

Example :

```
#include<stdio.h>
#include <string.h>
void main()
{
char str[20];
printf("Enter string: ");
gets(str);//reads string from console
printf("String is: %s",str);
printf("\nReverse String is: %s",strrev(str));
}
```

Output

```
Enter string: javatpoint
String is: javatpoint
Reverse String is: tnioptavaj
```

7 . strlwr()

The strlwr(string) function returns string characters in lowercase. **Example:**

```
#include<stdio.h>
#include <string.h>
void main()
{
char str[20];
printf("Enter string: ");
gets(str);//reads string from console
printf("String is: %s",str);
printf("\nLower String is: %s",strlwr(str));
}
```

Output

```
Enter string: JAVATpoint
String is: JAVATpoint
Lower String is: javatpoint
```

8 . strupr()

The strupr(string) function returns string characters in uppercase.

Example:

```
#include<stdio.h>
#include <string.h>
void main()
{
    char str[20];
    printf("Enter string: ");
    gets(str);//reads string from console
    printf("String is: %s",str);
    printf("\nUpper String is: %s",strupr(str));
}
```

Output:

```
Enter string: javatpoint
String is: javatpoint
Upper String is: JAVATPOINT
```

9. strstr() :

The strstr() function returns pointer to the first occurrence of the matched string in the given string. It is used to return substring from first match till the last character.

Syntax :

char *strstr(**const char** *string, **const char** *match)

string: It represents the full string from where substring will be searched.

match: It represents the substring to be searched in the full string.

Example:

```
#include<stdio.h>
#include <string.h>
void main()
{
    char str[100]="this is javatpoint with c and java";
    char *sub;
    sub=strstr(str,"java");
    printf("\nSubstring is: %s",sub);
}
```

Output:

```
javatpoint with c and java
```

8.4 Array of Strings

An array of strings is a two-dimensional array. This is often required in applications dealing with a list of names, etc.

An array of strings can be declared as

```
char str [10] [20] ;
```

str is an array of 10 strings; each storing maximum 20 characters

str[0]	20 chars
	20 chars
	:
str[9]	

An array of strings can be initialized as follows:

Example:

```
char cities [4] [10] = { "Pune" , "Mumbai" , "Delhi" , "Chennai"};
```

They are stored as:

	0	1	2	3	4	5	6	7	8	9
cities [0]	P	u	n	e	\0					
cities [1]	M	u	m	b	a	i	\0			
cities [2]	D	e	I	h	i	\0				
cities [3]	C	h	e	n	n	a	i	\0		

SUMMARY :

In this chapter we elaborated the concept of Strings. String handling plays an important role while programming. Every string is terminated with a null character. Different ways of declaration and initialization of strings were seen. The main header file string.h is required for using the string library functions. Built in sting functions save a lot of programming effort.

SOLVED PROBLEMS AND SELF CHECK QUESTIONS

SOLVED PROBLEMS

Author should provide here about 2-3 solved problems, on this Unit topics.

- 1) ____ function is used for accepting multi word strings.
 - a) scanf()
 - b) **gets()**
 - c) cin
 - d) cout
- 2) String functions are defined in the header file ____
 - a) **string.h**
 - b) ctype.h
 - c) stdio.h
 - d) conio.h
- 3) __ function is used to find the length of a string.
 - a) **strlen()**
 - b) strlenlength()
 - c) length()
 - d) count()
- 4)strupr() function is used to convert the entire string to____
 - a) Camelcase
 - b) Togglecase
 - c) Lowercase
 - d) **Uppercase**
- 5) The library function used to find the last occurrence of a character in a string is
 - a) strnstr()
 - b) laststr()
 - c) **strrchr()**
 - d) strstr()

SELF-TEST 01

Author should provide here about 5-7 MCQs, on this Unit

MCQ 01 : __ format specifier is used to accept a string with white spaces.

- a) %[^\n]
- b) %s
- c) %u
- d) %c

MCQ 02 : The string is automatically ended with __ character.

- a) \n
- b) \ Ø
- c) \t
- d) \b

MCQ 03: The ____ function is identical to scanf except that data is read from the string pointed to by buf rather than stdin.

- a) sscanf()
- b) sprintf()
- c) puts()
- d) printf()

MCQ 04: The ____ function can be used to test whether a string is palindrome or not.

- a) strcpy()
- b) strcat()
- c) strlen()
- d) strrev()

MCQ 05: If the two strings are identical, then strcmp() function returns

- a) 0
- b) -1
- c) 1
- d) NULL

MCQ 06: Is there any difference between the two statements?

```
char *ch = "YCMOU";  
char ch[] = "YCMOU";
```

- a) YES
- b) NO

EXERCISE

Author should provide 5-10 short answer questions (of 5 Marks) here

- Q1. What is String elaborate with suitable examples?
- Q2. How is a string declared and initialized explain with examples?
- Q3. Differentiate between printf and sprintf, scanf and sscanf functions.
- Q4. Explain any five string functions with their usage, support examples.
- Q5. Write a 'C' program to decide whether a given string is Palindrome or not.
- Q6. Write a 'C' program to accept 'n' names and display them and search for a particular name.

REFERENCE MATERIAL

REFERENCES: [geeksforgeeks.com](https://www.geeksforgeeks.com/), [unstop.com](https://www.unstop.com/)

<https://www.w3schools.com/c/>, [javatpoint.com](https://www.javatpoint.com/), [idiabix.com](https://www.idiabix.com/)

MOOCs

YOUTUBE VIDEOS

WIKIPEDIA

BLOGS : UNSTOP.COM

OER (SLIDES, PDFs, TUTORIALS)

REFERENCE BOOKS

ANSWERS TO SELF-CHECK QUESTIONS

1.(a) 2. (b) 3. (c) 4. (d) 5. (a) 6. (a)

UNIT 9: FUNCTIONS

UNIT OBJECTIVES

After successful completion of this unit, you will be able to understand.

- ❖ Need of functions.
- ❖ Scope and lifetime of variables.
- ❖ Defining functions, function call, call by value, call by reference, return values, Storage classes.
- ❖ Category of function- No argument No return value, No argument with return value, argument with return value.
- ❖ Recursion.
- ❖ Command line Arguments

9.1 Need of functions.

A function is a sub program or set of instructions written to do a particular task. When function body is written, we call it as a function definition.

Programs written by the programmer may be too large and complex and also difficult to debug and test. If such a large program is divided into sub programs and compiled separately, task becomes easy. 'C' supports such type of structuring and these subprograms may be combined into a single unit. These sub programs are called functions and are much easier to understand.

Due to functions program becomes logical and user friendly. The most important feature is that one can call a function many times in the same program or in the external program. It helps in avoiding repetitions of the code. One has to just write a function once and call as many times as needed. Thus, Functions are sub modules used to develop a program.

Advantage:

- Due to functions, programs can be developed in a modular way. Thus, coding becomes easier
- The length of the program gets reduced because functions are called whenever they are required only.
- Debugging becomes easier, it becomes easy to find out errors from a function rather than the entire program.
- Repetition of code is avoided.
- Functional approach saves time and memory space.

Types of Functions

1. Built in / Library functions: They are Defined in the library files e.g. pow (), printf (), scanf ().

2. User defined: They are Defined by the user according to the need.

9.2 Scope and lifetime of Variables:

Scope of a variable defines in which part of the program the variable is actually available for use. Thus, the scope of variable is the region or part of the program in which the variable is visible or accessible.

Lifetime of variables is a period in which variable holds the given value at the time of program execution. This the period of time for which memory is allocated to a variable. It is the time for which the variable exists or persists in memory.

They are called as Local and Global variables. Global variables are available throughout the program whereas local variables are accessible by the respective functions only in which they are declared.

When a program consists of multiple functions then it is very important to know whether a variable is local or global.

Example:

int b; // Global variable, available to main as well as other functions.

void main ()

```
{  
    int a; // local variable available within main only
```

```
}
```

void func1()

```
{  
    int c;    // local variable to func1 , others can't access it  
  
    // func1 can access global variable 'b' , but not 'a'  
  
}
```

9.3 Defining functions, function call, call by value, call by reference, return values.**Steps for writing a function (How to write a function):**

The basic **three steps** involved while writing a function are:

1. Decide the prototype of the function.
2. How to Call a function.
3. Define the Function.

1. Decide the prototype:

- Name of the function.
- Data type of the parameters it is going to take.
- Return data type.

The general prototype of function will be as follows:

return data type function- name (data type of arguments)

e.gs.

int cube (int);

void line (int);

void line (void);

int greatest (int, int,int)

2. How to call a function:**a. Call by reference:**

When functions are called by passing their addresses they are called as call by reference.

ans = cube(&a)

b. Call by value:

When functions are called by passing the values they are called as call by value

ans = cube(a) // a is actual parameter

big = greatest(a,b,c)

Return data type indicates what data type will the function return to the calling program.

3. Define the function:

This part contains the actual code of the function. The actual statements to be executed are included in the function definition.

```
int cube (int p) //p is formal parameter
{
int ans;
ans = p*p*p;
return(ans);
}
```

9.4 Difference between Call by Value and Call by Reference.

No	Call by Value	Call by reference
1	The value of the actual argument is copied into the formal argument.	The formal argument represents the actual argument
2	The function cannot access the original variable.	The function can access the original variable.
3	Any changes made to the formal argument are not made to the actual argument.	Any changes made to the formal argument are reflected in the actual argument.
4	No need to use pointers.	Requires pointer to be used.
5	Used when we only need the value of a variable but don't need to modify it.	Used when a variable has to be modified by the function.

Example to Demonstrate the Difference between Call by Value and Call by Reference. The Figures 19.1 and 9.2 show the working of the two programs.

Call by value is copying the value of a variable in another variable. So any change made in the copy will not affect the original location.

Note: Any change, which is done within the function is local and will not change the outside function.

```
#include<stdio.h>
#include<conio.h>
void fun(int);
void main()
{
    int a=10; //a is read as 10
    clrscr();
    printf("%d ",a);
    fun(a);
    printf("%d ",a); //a is read as 10
    getch();
}
void fun(int x) //x is read as 10
{
    printf("%d ",x);
    x++;
    printf("%d ",x); //x is read as 11
}
```

OUTPUT :

10 10 11 10

Working of Call by Value program:

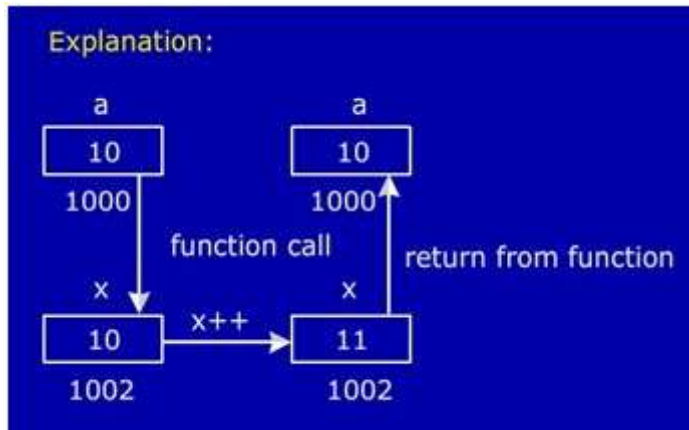


Figure 9.1

In the call by reference method, instead of passing values to the function that is being called, references/pointers to the original variables are passed

Note: Any modification which is done to variable 'a' will change the outside function

```
#include<stdio.h>
#include<conio.h>
void fun(int *);
void main()
{
    int a=10; //a is read as 10
    clrscr();
    printf("%d ",a);
    fun(&a);
    printf("%d ",a); //a is read as 11
    getch();
}

void fun(int *x) //a is read as 10
{
    printf("%d ",*x);
    (*x)++;
    printf("%d ",*x); //x is read as 11
}
```

Output



```
10 10 11 11
```

Working of Call by Value program:

Explanation:

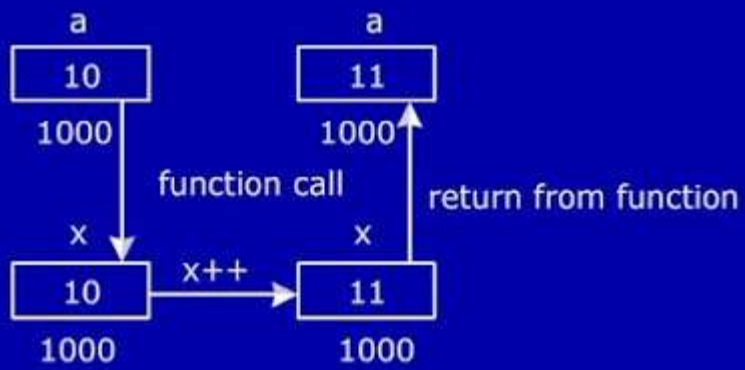


Figure 9.2

9.5 The return statement: The keyword return is used to terminate execution of the function and return the control to the calling function. It can also be used to return a single value to the calling function.

Syntax :

return ;

OR

return expression;

OR

return (expression);

A function can contain multiple return statements but function execution will stop as soon as the first return statement is encountered during the execution.

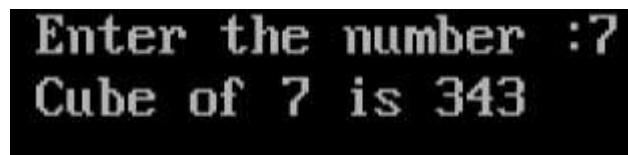
Write a Program to find cube of a number using functions.

```
#include<stdio.h>
#include<conio.h>
//prototypes
int cube(int);
int getdata(void);
void display(int,int);
void main(void)
{
    int n,ans;
    clrscr();
    n=getdata();
    ans=cube(n);
    display(n,ans);
    getch();
}

int cube(int x)
{
    return(x*x*x);
}

int getdata(void)
{
    int num;
    printf("Enter the number :");
    scanf("%d",&num);
    return(num);
}

void display(int x,int y)
{
    printf("Cube of %d is %d\n",x,y);
}
```



Write a program using functions to reverse a number and decide whether it is palindrome or not.

```
#include<stdio.h>
#include<conio.h>
//prototype
int getnum(void);
int reverse(int);
void display(int,int);
main()
{
    int n,rev;
    clrscr();
    n=getnum();
    rev=reverse(n);
    display(n,rev);
    getch();
}

int getnum(void)
```

```

{
int n;
printf("Enter the number");
scanf("%d",&n);
return(n);
}
void display(int x,int y)
{
printf("Reverse of %d is %d\n",x,y);
if(x==y)
    printf("%d is palindrome",x);
else
    printf("%d is not palindrome",x);
}

```

```

int reverse(int n )
{
int d, rev=0;
while(n!=0)
{
d=n%10;
rev=rev*10+d;
n=n/10;
}
return(rev);
}

```

```

Enter the number :121
Reverse of 121 is 121
121 is palindrome

```

```

Enter the number :123
Reverse of 123 is 321
123 is not palindrome

```

Write a program using functions to swap two numbers using the concept of call reference

```
/* Documentation Section*/
/* To swap two numbers */
/* Link Section */
#include<stdio.h>
#include<conio.h>
#define pf printf // pf is macro
#define sf scanf
/* Prototypes*/
int getno(void);
void swap(int *, int *);
main()
{
    int a, b;
    clrscr();
    a = getno();
    b=getno();
    printf("\n....Before swapping...\n");
    printf("Value of a is %d \n",a);
    printf("Value of b is %d \n",b);
    swap(&a, &b);
    printf("\n....After swapping....\n");
    printf("Value of a is %d \n",a);
    printf("Value of b is %d \n",b);
    getch();
}

void swap(int *a,int *b)
{ int temp;
  temp = *a;
  *a = *b;
  *b = temp;
}

int getno()
{
    int num;
    printf("Enter the number : ");
    scanf("%d",&num);
    return(num);
}
```

```
Enter the number : 20
Enter the number : 30

....Before swapping...
Value of a is 20
Value of b is 30

...After swapping...
Value of a is 30
Value of b is 20
```

9.6. Storage Classes: It refers to the manner in which memory is allocated to the variables.

The storage class determines the scope and lifetime of a variable.

The storage class of a variable determines:

1. Where a variable is stored.
2. Its default initial values.
3. Scope of the variable
4. Lifetime of the variable.

Following storage classes are there:

1. Automatic: (**auto**) Local variable known only to the function in which it is declared. Default storage class is auto.
2. Static: (**static**) Local variable which exists and retains its value even after the control is transferred to the calling function
3. Extern: (**extern**) Global variables known to all functions in file.
4. Register: (**register**) Local variable which is stored in the register, due to this executing speed increase.

9. 7 Category of Functions:

1. Functions with no arguments and no return values:

A function which does not return any value and to which we do not pass any arguments is called as a function with no return value and with no arguments.

e.g. void add(void);

2. Functions with arguments and no return values:

In this type of category of functions we pass arguments as an actual arguments and the called function receives these values in its definition as formal parameters ,but no output or return values are sent back by called function to the calling function.

e.g. void fun1()

```
{  
    fun2(a);  
}  
  
void fun2(int a)  
  
{  
  
  
}
```

Here fun1() is calling function and fun2() is called function.

3. Functions with no arguments but returns a value.

The function which does not accept arguments but returns the value is a function without arguments and returning value.

e.g. void main ()

```
{  
    ans = addprime(); // sum of first 10 prime nos.  
}  
  
int addprime() // function of our concern  
  
{ int sum;  
    // processing ...  
    return(sum);  
}
```

4. Functions with arguments and returning a value.

Functions may have one or more than one argument with it which can be of different data types. Also, this function returns a value of required data type.

e.g. **int add (int f, int s, in t);**
 long int factorial (int num);

9.8 Recursion:

Recursion: It refers to the process in which a function calls itself. A termination condition should be specified so that a function can end.

e.g. Sum of first 'n' natural numbers

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

sum = 1+2+3+4+.....+(n-1)+n

=1+2+3+4+.....+(n-2)+(n-1)+n;

Write a program using the concept of recursion to find the sum of the first 'n' numbers.

```
/* Documentation Section*/  
/* Use of Recursion */  
/*Link Section */  
#include<stdio.h>  
#include<conio.h>  
// prototype  
int addition(int);  
void main ()  
{  
    int n;  
    clrscr();  
    printf("Enter the value of n :");  
    scanf("%d", &n);  
    if(n<=0)  
        printf("Number should be greater than zero\n");  
    else  
        printf("Sum of 1 to %d = %d\n",n, addition(n));  
  
    getch();  
}  
  
int addition (int n)  
{  
    if(n==1)  
        return(1);  
    else  
        return(n+addition(n-1)); // return(5+4+3+2+1)
```

```
}
```

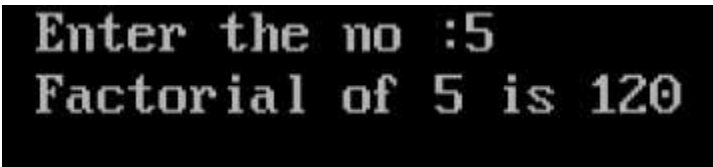
Write a program using the concept of recursion to find the factorial of a number.

```
n!= 1          if n=0
  = n*(n-1)    if n>0
5! = 5*4 !
= 5*4*3!
=5*4*3*2!
=5*4*3*2*1
```

```
/* Documentation Section*/
/* Use of Recursion */
/*Link Section */
#include<stdio.h>
#include<conio.h>
// prototype
int factorial(int n);

void main()
{
    int n;
    clrscr();
    printf("Enter the no :");
    scanf("%d",&n);
    if(n<0)
        printf("Number should be +ve");
    else
        printf("Factorial of %d is %d\n",n,factorial(n));
    getch();
}

int factorial(int n)
{
    if(n==0)
        return (1);
    else
        return(n*factorial(n-1));
}
```



Advantages of Recursion:

The program code is much compact in recursion and is easier to write.

Disadvantages:

Recursion functions use more storage since a stack is used and takes more computing time than iterative function statements.

9.9 Command Line arguments:

When you want to execute the program from command line by providing input at command line we use the concept of command line arguments.

void main(int argc, char *argv[])

==> argc : argument count

==> argv : argument vector

Example :

d:>add 10 20

argc variable will contain the value as 3

argv[0] = add.c

argv[1] = 10

argv[2] = 20

Write a program to add two numbers using the concept of command line arguments.

```
/*Documentation Section*/
/*To add two numbers using command line arguments*/
/*Link Section */
#include<stdio.h>
#include<conio.h>
void main(int argc, char *argv[])
{
    int f,s,sum;
    clrscr();
    if(argc>3)
    {
        printf("Excess parameters");
        getch();
    }
    else
    if(argc<3)
    {
        printf("Insufficient parameters");
        getch();
    }
    else
    {
        f = atoi(argv[1]);
        s = atoi(argv[2]);
        sum = f+s;
        printf("First no is %d\n",f);
        printf("Second no is %d\n",s);
        printf("Sum is %d\n",sum);
    }
}
```

```
getch();}
```

SUMMARY:

SOLVED PROBLEMS AND SELF CHECK QUESTIONS

SOLVED PROBLEMS

- 1) Functions are of two types built in and ____
 - a) predefined.
 - b) **user defined.**
 - c) Auto.
 - d) all of the above.
- 2) One can call a function by call by value method and ____
 - a) **Call by reference.**
 - b) Call by name.
 - c) Using CALL.
 - d) Recursion.
- 3) When a function does not return anything, we say it returns____
 - a) null
 - b) **void**
 - c) NULL
 - d) o
- 4) argc stands for ____
 - a) Argument centric.
 - b) Argument counter.
 - c) Argument vector.
 - d) **Argument count.**
- 5) Recursion uses __ data structure for its processing.
 - a) Tree.
 - b) Queue.
 - c) **Stack.**
 - d) Hash Table.

SELF-TEST 01

MCQ 01: Which of the following is a correct format for declaration of function?

- a) a) return-type function-name (argument type);
- b) b) return-type function-name (argument type) {}
- c) c) return-type (argument type) function-name;
- d) d) all of the above

MCQ 02: The value obtained in the function is given back to main by using _____ keyword.

- a) return
- b) static
- c) new
- d) volatile

MCQ 03: Functions are primarily used for ____

- a) Decision making
- b) Variable declaration
- c) Code organization and reusability
- d) Printing Output

MCQ 04: What is the correct way to declare a function in C that takes no arguments and returns an integer.

- a) int func ()
- b) func(int)
- c) func()
- d) int func(int)

MCQ 05: How is the scope of a local variable defined in C?

- a) Limited to the function where it is declared.
- b) Limited to the entire program.
- c) Limited to the block where it is declared.
- d) Limited to the main() function.

MCQ 06: A recursive function is ____

- a) A function that calls itself.
- b) A function that uses a loop.
- c) A function that returns no value.
- d) A function that takes no arguments.

MCQ 07 : Functions cannot return more than one value at a time?

- a) YES
- b) NO

MCQ 08: The maximum combined length of the command-line arguments including the spaces between adjacent arguments is

- a) 128 characters
- b) 256 characters
- c) 67 characters
- d) It may vary from one operating system to another

MCQ 09 : The scope of a variable declared inside a function is _____

- a) Local to the function.
- b) Global.
- c) Static.
- d) Automatic.

MCQ 10 : What is the purpose of the 'extern' keyword when declaring a function in 'C'.

- a) To specify that the function is not defined in the current file
- b) To declare a function as a constant
- c) To declare a function with a variable number of arguments
- d) To specify that the function is recursive

SUMMARY :

In this Chapter we explored the Need of functions, due to functions a modular approach of development came into existence. Further we studied the Scope and lifetime of variables, in this we came across the concept of global and local variables.

We studied how to create a user defined function. During this study we studied the prototype of function. Function can be called in two ways i.e., call by value and call by reference. Also, we explored in brief the four storage classes.

Categories of functions were discussed depending upon the return data type and the datatype of the parameters a function accepts.

Some repetitive work can be developed using the concept of recursion.

At the end we came across , how to give input from the command lines.

EXERCISE

- Q1. What is a function, enlist the two types of functions?
- Q2. Discuss the steps required for creating a new user defined function?
- Q3. Differentiate between Call by value and Call by reference methods of calling a function.
- Q4. Explain the concept of Scope and lifetime of variables.
- Q5. What is meant by Recursive Function, discuss its advantages and disadvantages.
- Q6. Explain the concept of command Line arguments.
- Q7. Discuss the four categories of functions with suitable examples.
- Q8. Explain the use of extern, static keyword related to functions.
- Q9. Write a program using function.
- Q10. Discuss the concepts a. Modularity b. Reusability c. Abstraction related to functions.

REFERENCE MATERIAL

REFERENCES: geeksforgeeks.com, unstop.com

<https://www.w3schools.com/c/>, javatpoint.com, idiabix.com

<https://www.examveda.com>,

<https://estudies4you.blogspot.com>

MOOCs

YouTube VIDEOS

WIKIPEDIA

BLOGS : UNSTOP.COM

OER (SLIDES, PDFs, TUTORIALS)

REFERENCE BOOKS

ANSWERS TO SELF-CHECK QUESTIONS

1.(a) 2. (a) 3. (c) 4. (a) 5. (a) 6.(a) 7.(a) 8.(d) 9.(a) 10.(a)

UNIT 10: POINTERS

UNIT OBJECTIVES

After successful completion of this unit, you will be able to understand.

- ❖ Understanding pointers.
- ❖ Declaring pointer variable, initialization of pointer variable, accessing address of a variable.
- ❖ Pointers arithmetic and Pointer expressions.
- ❖ Arrays and Pointers.
- ❖ Pointer to Pointer.
- ❖ Pointers to Functions.

10.1 Understanding Pointers : Pointers:

- Memory consists of millions of cells. Each cell has its own address. these locations can be accessed using a variable or memory addresses, these memory addresses are nothing but the pointers.
- It is a variable that represents the location of a data item such as a variable or an array element.
- It is an address of a memory location.
- Due to use of pointers processing is faster as we are actually passing the address.
- Memory consists of millions of cells. Each cell has its own address. these locations can be accessed using a variable or memory addresses, these memory addresses are nothing but the pointers,

10.2 Declaring, initialization of a Pointer and Accessing address of a Variable.

While declaring pointers one should know the data type of the value which we are going to store in the memory.

Syntax:

Data type *pointer_name;

```
int a;           /* a is an integer variable*/
```

```
int *p;          /* p is a pointer variable to integer*/
```

* is called as an indirection operator

Once a pointer variable is declared one has to assign the address of the variable to it. This is called as pointer initialization,

Syntax:

pointer_name = &(variable name);

& is called as address operator

Example:

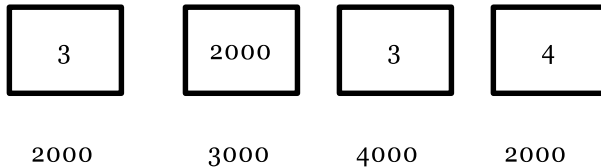
	a	p	Variable name
a = 10	10	2000	Contents
p = &a	2000	4000	Address

*p = 20 [value at address stored in p i.e. value of a will become '20']

e.g. char * ptr1; float *ptr2

GIVE THE OUTPUT OF THE FOLLOWING PROGRAM

```
void main ()
{
int i=3, *j,k;    /* j is pointer to an integer*/
j = &i;          /* j stores address of i*/
k =*j;           /* stores contents of the address stored in j*/
*j =4;           /* store 4 to the location pointed by j*/
printf ("i=%d, *j = %d, k = %d\n", i, *j, k);
}
```



OUTPUT : i=4 *j =4 k= 3

Printing the address of a variable:

One can print the address of variable by using the %u format specifier, * is used to print the contents of the pointer.

Write a program to print the address of a variable.

```
void main()
{ int i=3;
  printf("\n Address of i= %u\n",&i);
  printf("\n Value of i = %d",i);
  printf ("\n Value of i = %d", *(&i));
}
```

OUTPUT

Address of i = 6485

Value of i =3

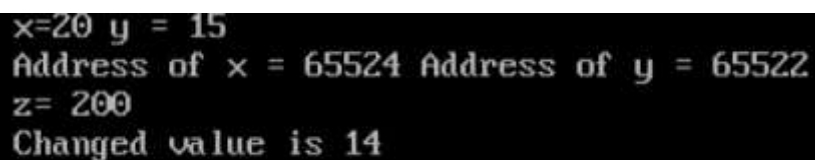
Value of i =3

10.3 Pointer Arithmetic and Pointer expressions: Pointer arithmetic means which type of operations can be performed on the pointers.

1. We can add integers or subtract integers from pointers.
e.g. $P1+3$, $p1-2$ are valid.
2. We can subtract one pointer from another pointer.
e.g. $p1-p2$, $p2-p1$ are valid.
3. We can use increment and decrement operations on pointers.
e.g. $p1++$, $--p2$ are valid.
4. Two pointers cannot be added. $P1+p2$ are invalid
5. Two pointers cannot be divided or multiplied.
e.g. $P1*p2$, $p1/p2$, $\&p1/5$ are invalid operations.
6. Pointers can be compared using relational operators if they are of same data type.
 $P1==p2$, $p1<p2$, $p1!=p2$

Write a program to illustrate the use of pointer expression.

```
void main()
{
    int x, y, z, *p1, *p2;
    clrscr();
    x = 20;
    y = 15;
    p1 = &x;
    p2 = &y;
    printf("x=%d y = %d\n", x, y);
    printf("Address of x = %u Address of y = %u\n", p1, p2);
    z = *p1**p2-100;
    *p1 = *p1-6;
    printf("z= %d\n", z);
    printf("Changed value is %d \n", x);
    getch();
}
```



```
x=20 y = 15
Address of x = 65524 Address of y = 65522
z= 200
Changed value is 14
```

10.4 Arrays and Pointers: For arrays one can easily assign its address by storing its base address i.e. address of the zeroth

element.

0	1	2	3	4	Array Location
50	60	70	80	90	

1000 1002 1004 1006 1008 Memory address e.g. `int a[4] = {10,20,30,40};`

The array name 'a' is a constant pointer which stores the address of first element a[0].

```
int x[10];
```

```
int *p;
```

```
p = x; or p = &x[0];
```

Write a program to initialize an integer array and print it using the concept of pointers.

```
void main()
{
int a[]={10, 20, 30, 40, 50};
int *p1,i;
clrscr();
p1= a; // p1 points to first element of an array.
i=0;
printf("Subscript \t Address \t Value \n");
while(i<5)
{
printf(" %d \t\t %u \t\t %d\n",i,p1,*p1);
p1++;i++;
}
getch();
}
```

Subscript	Address	Value
0	65516	10
1	65518	20
2	65520	30
3	65522	40
4	65524	50

We can have Array of Pointers also, It is a collection of addresses.

Syntax:

data_type * arrayname[size];

e.g. int *p[5];

p is array of 5 integer pointers p[0] to p[4].

10.4 Pointer to Pointer: As we have a pointer to a variable we can create a pointer to pointer. It stores the address of a pointer variable. Pointers to pointers are used in handling of strings , multidimensional arrays , linked lists , trees and graphs.

```
int data=20, *p1;
```

```
p1 = &data;
```

```
int **p2; /* multiple indirection */
```

p2 is pointer to pointer i.e. p2 stores the address of pointer variable

p1.

Write a program to demonstrate the concept of Pointer to Pointers.

```
void main()
{
int data = 20, *p1;
int **p2;
clrscr();
p1= &data;
printf("data = %d\n",*p1);
printf("Address of data = %u\n",p1);
p2=&p1;
```

```
printf("Address of p1 = %u\n",p2);  
printf("data = %d", **p2);  
getch();  
}
```

```
data = 20  
Address of data = 65524  
Address of p1 = 65522  
data = 20
```

10.5 Pointers to Functions: Pointers can be used with functions in following ways.

1. Functions can take pointers as parameters.

Syntax :

```
return_data_type function name(type *pointer);
```

e.g. void display(int *ptr);

```
int string_len(char *str);
```

```
void swap(int *p1, int *p2);
```

2. Functions can return a pointer.

Syntax :

```
return_data_type * function name(argument list);
```

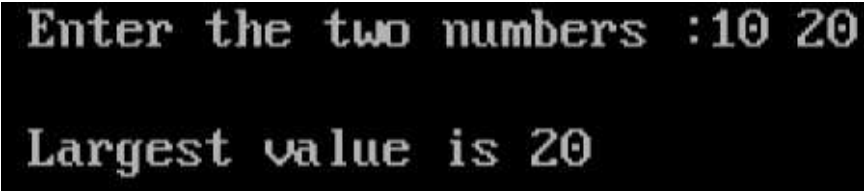
e.g. int *f1 display(int);

f1 is function which accepts an integer parameter and returns a pointer to an integer.

Write a program to demonstrate the concept of functions returning pointer.

```
#include<stdio.h>
#include<conio.h>
// prototype
int *largest(int *, int *);
void main()
{
    int f,s,*big;
    clrscr();
    printf("Enter the two numbers :");
    scanf("%d%d",&f,&s);
    big =largest(&f,&s);
    printf("\nLargest value is %d\n",*big);
    getch();
}
```

```
int *largest(int *p1, int *p2)
{
    if(*p1>*p2)
        return(p1);
    else
        return(p2);
}
```



```
Enter the two numbers :10 20
Largest value is 20
```

SOLVED PROBLEMS AND SELF CHECK QUESTIONS

SOLVED PROBLEMS

1) * is called as _____ operator.

- a) multiplication
- b) **indirection.**
- c) asterisk
- d) all of the above.

2) Addition of two pointers is a _____.

- a) **Invalid**
- b) valid.

3) Comment on the following pointer declaration.

```
int *ptr, p;
```

a) ptr is a pointer to integer, p is not

b) ptr and p, both are pointers to integer

c) ptr is a pointer to integer, p may or may not be

d) ptr and p both are not pointers to integer

4) What is the meaning of the following declaration?

```
int (*p)[5];
```

- a) It will result in compile error.
- b) p is a pointer to 5 integers.
- c) p is a pointer to integer array.
- d) **p is a pointer to an array of 5 integers.**

5) What is the output of the following code.

```
void main()
{
    int ***r, **q, *p, i=8;
    p = &i;
    q = &p;
    r = &q;
    printf("%d, %d, %d\n", *p, **q, ***r);
    getch()
}
```

- a) **8, 8, 8**
- b) 4000, 4002, 4004

c) 4000, 4004, 4008

d) 4000, 4008, 4016

SELF-TEST 01

MCQ 01: What will be the output of the following program?

```
void main()
{
    int x=30, *y, *z;
    y=&x; /* Assume address of x is 500 and integer is 4 byte size */
    z=y;
    *y++=*z++;
    x++;
    printf("x=%d, y=%d, z=%d\n", x, y, z);
    getch();
}
```

- a) x=31, y=502, z=502
- b) x=31, y=500, z=500
- c) x=31, y=498, z=498
- d) x=31, y=504, z=504

MCQ 02: Are the expression `*ptr++` and `++*ptr` are same ?

- a) True
- b) False

MCQ 03: If ptr is a pointer to int, having value ptr=100. After ptr++, what is the value of ptr?

- a) 100
- b) 101
- c) 102
- d) 103

MCQ 04: Pointer variable is declared using preceding _____ sign

- a) &
- b) %
- c) *
- d) ^

MCQ 05: A Pointer is?

- (a) A keyword used to create variables.
- (b) A variable that stores address of an instruction.
- (c) A variable that stores address of other variable.
- (d) All of above

SUMMARY:

In this Chapter we saw that Pointers provide a powerful and flexible way to manipulate data. However, they should be used correctly and carefully. We discussed the concept of pointer operations. We also studied various valid and invalid pointer arithmetic operations. Pointer to Pointer concept was discussed. Further Arrays and Pointer were discussed. Pointer of arrays was discussed. Finally Functions and pointers were studied.

EXERCISE

- Q1. What is a pointer? Why should one use pointers?
- Q2. Explain with an example how a pointer variable is declared and initialized with an address of a variable.
- Q3. Enlist the various arithmetic operations which can be performed with pointers.
- Q4. Explain the advantages of pointers and do they have any disadvantage.
- Q5. Write a program to swap two values using a function swap(), use pointers as parameters.
- Q6. Explain the concept of Pointer to Pointer with an example?

REFERENCE MATERIAL

REFERENCES: [geeksforgeeks.com](https://www.geeksforgeeks.com/), [unstop.com](https://www.unstop.com/)

<https://www.w3schools.com/c/>, [javatpoint.com](https://www.javatpoint.com/), [idiabix.com](https://www.idiabix.com/)

https://www.examveda.com, https://www.sankalandtech.com, <https://gtu-mcq.com/>,
<https://www.sanfoundry.com/>

<https://testbook.com/>

MOOCs

YOUTUBE VIDEOS

WIKIPEDIA

BLOGS : UNSTOP.COM

OER (SLIDES, PDFs, TUTORIALS)

REFERENCE BOOKS

ANSWERS TO SELF-CHECK QUESTIONS

1.(d) 2. (b) 3. (c) 4. (c) 5. (c)

UNIT 11: STRUCTURES AND UNIONS

UNIT OBJECTIVES

After successful completion of this unit, you will be able to understand.

- ❖ Defining structures.
- ❖ Declaring and accessing structure members, initializing of structure.
- ❖ Arrays of structure.
- ❖ Difference between array and structure.
- ❖ Defining a Union.
- ❖ Declaring and accessing union members.
- ❖ Difference between Structure and Unions.

11.1 Introduction to Structures: If we need to store a single value at a time then we use simple data types as per our requirement. e.g. int, char, float etc. To store multiple values of same data type we use an array. In an array we can't store elements of different data type, the solution to this is structure.

Structure is a collection of dissimilar elements, allocated contiguous memory blocks referred by a common name.

It is a user defined data type.

e.g.

```
int roll[10];
```

```
char name[10][20];
```

```
float ht[10];
```

These are three fields representing an entity. All three fields are of different data types, if it is to be represented with an array we will require three arrays and it will be difficult to maintain if fields are in large number. Such applications can be easily represented using a structure.

11.2 Declaring and accessing structure members, initializing of structure.

The above three fields can be **declared** using a structure as given below.

```
struct stud
```

```
{  
    int roll;  
    char name [20];  
    float ht;  
} s;
```

's' is a structure variable of type stud;

Thus, the **Syntax** of declaring a Structure is as:

Syntax :

```
struct tag
```

```
{
```

data type variable name;

};

We can access the individual elements of a structure by using '.' Operator.

Syntax:

structure-variable . member name

In the above example roll number can be accessed as s.roll, name can be accesses using s.name and height can be accessed using s.ht;

One can accept the values from the user or one can even initialize the members variable as shown below:

```
struct stud s={1,"Raj",5.5}
```

Using typedef with Structures

The typedef keyword can be used to give a new name to the structure. The new name can be used as a data type to create variables, passing values to functions. The keyword 'struct' need not be used again.

Syntax

```
typedef struct name
{
    //members
}new_name;
```

Example

i.

```
typedef struct
{ char name [20] ;
  int rollno ;
  float percentage ;
} student ;
```

ii.

```
typedef struct studrec
{ char name [20] ;
  int rollno;
  float percentage;
} student;
```

In (i) student is the name of the new type. In (ii) studrec is the tag name which is not needed but used for clarity.

Variables of this type can be created by the statement:

```
student s1, s2;
```

11.3 Array of Structures :

struct stud

```
{  
    int roll;  
    char name [20];  
    float ht;  
} s;
```

In the above declaration one can have only one record, but if we want many records of type 's' then we have to use the concept of array of structures.

struct stud

```
{  
    int roll;  
    char name [20];  
    float ht;  
} s[10];
```

Here now 's' is an array of structure stud.

Write a program to create a structure 'stud' having member elements as rno, name, th, pr, tot. Use the concept of Array of structures.

```
#include<stdio.h>  
#include<conio.h>  
#define mx 100  
struct stud  
{  
    int rno;  
    char name[20];  
    int th,pr,tot;  
}s[mx];  
void main()  
{  
    int i, nos;  
    clrscr();
```

```

printf("Enter no of students :");
scanf("%d",&nos);
for(i=0;i<nos;i++)
{
printf("Enter roll no :");
scanf("%d",&s[i].rno);
printf("Enter Name :");
scanf("%s",s[i].name);
printf("Enter theory marks :");
scanf("%d",&s[i].th);
printf("Enter practical marks :");
scanf("%d",&s[i].pr);
s[i].tot=s[i].th+s[i].pr;
}
printf("ROLLNO NAME THEORY PRACTICAL TOTAL\n");
printf(".....\n");
for(i=0;i<nos;i++)
{
printf("%d\t %s\t %d \t%d \t %d\n",s[i].rno,s[i].name,
s[i].th,s[i].pr,s[i].tot);
}
getch();
}

```

```

Enter no of students :2
Enter roll no :1
Enter Name :Raj
Enter theory marks :50
Enter practical marks :60
Enter roll no :2
Enter Name :John
Enter theory marks :70
Enter practical marks :80
ROLLNO NAME THEORY PRACTICAL TOTAL
.....
1      Raj      50      60      110
2      John     70      80      150

```

Write a program to create structure stud having following fields rno, name,marks[3],total, marks is an integer array.

```
#include<stdio.h>
#include<conio.h>
#define mx 100
struct stud
{
int rno;
char name[20];
int m[3];
int tot;
}s[mx];
main()
{
int i, nos;
clrscr();
printf("Enter no of students :");
scanf("%d",&nos);
printf("Enter data\n");
for(i=0;i<nos;i++)
{
printf("Enter the rollno :");
scanf("%d",&s[i].rno);
printf("Enter the name :");
scanf("%s",s[i].name);
printf("Enter marks of subject 1 :");
scanf("%d",&s[i].m[0]);
printf("Enter marks of subject 2 :");
scanf("%d",&s[i].m[1]);
printf("Enter marks of subject 3 :");
scanf("%d",&s[i].m[2]);
s[i].tot=s[i].m[0]+s[i].m[1]+s[i].m[2];
}
printf("RNO  NAME  SUB1  SUB2  SUB3  TOTAL\n");
for(i=0;i<nos;i++)
printf("%d\t%s\t%d\t%d\t%d\t%d\n",s[i].rno,s[i].name,
s[i].m[0],s[i].m[1],s[i].m[2],s[i].tot);
```

```
getch();
```

```
}
```

```
Enter no of students :2
Enter data
Enter the rollno :1
Enter the name :Raj
Enter marks of subject 1 :10
Enter marks of subject 2 :20
Enter marks of subject 3 :30
Enter the rollno :2
Enter the name :Tom
Enter marks of subject 1 :30
Enter marks of subject 2 :40
Enter marks of subject 3 :50
RNO    NAME    SUB1    SUB2    SUB3    TOTAL
1      Raj     10      20      30      60
2      Tom     30      40      50      120
```

11.4 Difference between an array and a structure.

Srno	Parameter	Array	Structure
1	Data Collection	Array is a collection of homogeneous data	Structure is a collection of heterogeneous data.
2	Element Reference:	Array elements are referred by subscript	Structure elements are referred by its unique name
3	Access Method:	Array elements are accessed by its position or subscript	Structure elements are accessed by its object as '.' operator
4	Data type:	Array is a derived data type	Structure is user defined data type.
5	Syntax:	array_name[size];	<pre>struct struct_name { Structure_element_1; Structure_element_2; ---- ----- Structure_element_n; }struct_var_nm;</pre>
6	Example];	int rn_array[5]	<pre>struct item_mst { int rno; char m_array[50]; } it;</pre>

11.5 Nested Structures.

A structure can be declared within a structure. Such a structure is called a nested structure. The inner structure is completely enclosed inside the outer structure.

```
struct outer
{
    // members of outer structure

    struct inner
    {
        // members of inner structure

    } inner_structure_variable ;
} outer_structure_variable;
```

Example

```
struct student
{
    char name [20] ;
    int roll no ;
    struct date
    {
        int day;
        int month ;
        int year ;
    } birthdate ;
    float percentage;
}s ;
```

Here, s is a variable of structure student.

Accessing members of nested structures

The members of the outer structure variable can be accessed using the syntax:

Syntax

Outer structure variable.inner structure variable.inner member;

Example:

The members of variable s are:

```
s.name
s.rollno
s.birthdate.day
s.birthdate.month
s.birthdate.year
s.percentage
```

The same nesting can be done in another way as shown below.

```
struct date
{
    int day;
    int month;
    int year;
};
```

```
struct student
{ char name [20] ;
  int rollno ;
  struct date birthdate;
  float percentage;
} s;
```

Here, date is declared as a separate structure. Thus, it can be used as an individual structure as well.

11.6 Unions:

- ❖ Defining a Union.
- ❖ Declaring and accessing union members.
- ❖ Difference between Structure and Unions.

A union is similar to a structure but only one of its members can be used at a time.

Definition

A union is a variable that contains multiple members of possibly different data types grouped together under a single name. However, all of them share the same memory area. Hence, only one of the members will be active at a time.

11.6 Declaration of a Union , Accessing Members of the Union

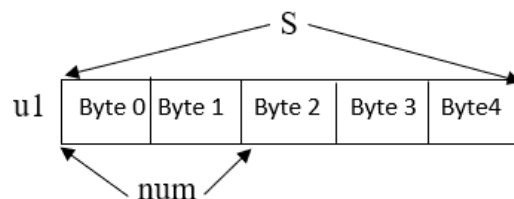
A union is declared in the same way as a structure except that the keyword “union” is used instead of ‘struct’.

Syntax

```
union tag
{
    union_members;
} variable;
```

e.g.

```
union u
{ char s[5];
  int num;
} u1;
```



The variable `u1` will be allocated memory to accommodate the largest member of the union. In this example, it will be allocated 5 bytes. Only One member, i.e., either the string or the integer `num` will be stored and can be accessed at a time. Both do not exist simultaneously. Thus, Union is used where mutually exclusive case arises.

Accessing Members of the Union

Union members can be accessed using the dot operators, i.e.,

`union_variable.member`

e.g. `u1.num`, `u1.name`

Thus a Union is similar to a structure. The only difference is that a union variable type allocates a storage space equal to the largest variable type.

`struct mystruct`

```
{
    int x;      2
    float y;    4
};             ==> 6 bytes
```

`union myunion`

```
{
```

```

int x;
float y;
}u;          ==> 4 bytes
struct stud
{
    int rno;
    int year;
    char name[20];
}s;          ==> 24 bytes

```

```

union stud
{
    int rno;
    int year;
    char name[20];
}u;          ==> 20 bytes

```

1. In structures each member has its own storage location
In union all members use the common location
2. In structures we can handle many members at a time.
In Unions we can handle only one member at a time.

11.7 Difference between a Structure and a Union.

Srno	Parameter	Structure	Union
1	Definition	A structure is a user-defined data type that groups different data types into a single entity.	A union is a user-defined data type that allows storing different data types at the same memory location.
2	Keyword	The keyword struct is used to define a structure	The keyword union is used to define a union
3	Size	The size is the sum of the sizes of all members, with padding if necessary.	The size is equal to the size of the largest member, with possible padding.
4	Memory Allocation	Each member within a structure is allocated unique storage area of location.	Memory allocated is shared by individual members of union.
5	Data Overlap	No data overlap as members are independent.	Full data overlap as members shares the same memory.
6	Accessing Members	Individual member can be accessed at a time.	Only one member can be accessed at a time.

SOLVED PROBLEMS AND SELF CHECK QUESTIONS

SOLVED PROBLEMS

- 1) ____ keyword is used to declare a structure in C
 - a) union.
 - b) **struct.**
 - c) class.
 - d) Enum.
- 2) Which of the following cannot be a structure member?
 - a) **Function.**
 - b) Array.
 - c) Structure.
 - d) All of the above.
- 3) Union differs from structure in the following way.
 - a) All members are used at a time.
 - b) **Only one member can be used at a time.**
 - c) Union cannot have more members.
 - d) All of the above.
- 4) Size of the unions is the size of the largest element of the Union.
 - a) False.
 - b) **True.**
- 5) How are structure elements accessed in 'C'.
 - a) Using the arrow (->) operator.
 - b) **Using the dot (.) operator.**
 - c) Using the asterisk (*) operator.
 - d) Using the percent (%) operator

SELF-TEST 01

MCQ 01: 1. Which of the following best describes a structure in C programming?

- a) A built-in data type.
 - b) A collection of variables of different data types under one name.
 - c) A special function to manipulate data.
 - d) A reserved keyword for defining loops
- What will be the output of the following program?

MCQ 02: What is the purpose of using structures or unions in C?

- a) To improve the efficiency of programs
- b) To organize related data types together
- c) To reduce the complexity of code
- d) To provide object-oriented programming features

MCQ 04: What does the sizeof operator in C return for a union.

- a) Total size of all union members.
- b) size of the largest member.
- c) size of the smallest member.
- d) size of the union itself.

MCQ 05: What is the purpose of using unions in C programs?

- a) To organize related data types together
- b) To define functions with multiple return values
- c) To save memory
- d) To create constants MCQ

SUMMARY:

In this Chapter we studied the concept of structures which allows us to group elements of different data types. Structure is a user define data type. Further we saw the concept of Unions, which is similar to a structure but only one of its members can be used at a time. It is used to save memory . Differences between an Array and a Structure ,Structure and a Union were discussed.

EXERCISE

- Q1. What is a Structure , how is declared and initialized explain with an example?
- Q2. What is an Union , how is it declared and initialized explain with an example?
- Q3. Differentiate between an Array and a Structure.
- Q4. Differentiate between a Structure and a Union.
- Q5. Explain nesting of a structure with an example. Explain considering date of birth a member element of s structure student.
- Q6. Write a 'C' program to create a structure 'employee' with member elements empid, name and salary. Accept details of 'n' employees and display the records.

REFERENCE MATERIAL

REFERENCES: [geeksforgeeks.com](https://www.geeksforgeeks.com/), [unstop.com](https://www.unstop.com/)

<https://www.w3schools.com/c/>, [javatpoint.com](https://www.javatpoint.com/), [idiabix.com](https://www.idiabix.com/)

<https://www.examveda.com/>, <https://www.sankalandtech.com/>, <https://gtu-mcq.com/>, <https://www.sanfoundry.com/>

<https://testbook.com/>, <https://letsfindcourse.com/>

[https://www.lkouniv.ac.in/site/writereaddata/siteContent/202005082120331143zeeshan_Difference Table_4.pdf](https://www.lkouniv.ac.in/site/writereaddata/siteContent/202005082120331143zeeshan_Difference_Table_4.pdf)

MOOCs

YOUTUBE VIDEOS

WIKIPEDIA

BLOGS : UNSTOP.COM

OER (SLIDES, PDFs, TUTORIALS)

REFERENCE BOOKS

ANSWERS TO SELF-CHECK QUESTIONS

1.(b) 2. (b) 3. (c) 4. (b) 5. (c)

UNIT 12: FILE HANDLING

UNIT OBJECTIVES

After successful completion of this unit, you will be able to understand.

- ❖ Introduction to Streams.
- ❖ Types of Files.
- ❖ Operations on text files.
- ❖ Standard Library.
- ❖ Input Output Function.
- ❖ Random access to files.

12.1 Introduction to Files and Streams:

Console input and Output functions such as `printf` and `scanf` are console oriented functions. However, most applications require a large amount of data. If this data has to be entered through the standard input device, it will take a lot of time. Moreover, once the execution is over, the data is lost. If the data is stored in files in memory it can be used whenever required. The input to a program as well as the output of a program can be stored in files. The 'C' library provides several functions to perform operations on files. Input /Output is done with streams. A stream is a sequence of bytes of data. A sequence of bytes flowing into a program is an input stream, and the one flowing out is an output stream. A File is a collection of data stored in one unit identified by a filename. IT can be a document, picture , audio etc. A file thus represents a sequence of bytes on the disk where a group of related data is stored .File is created for permanent storage of data. It is used to store large volume of persistent data.

Predefined streams:

There are 5 predefined streams, which are automatically opened when a C program starts executing and are closed when the program terminates.

Types of predefined streams		
i.	stdin	
ii.	stdout	
iii.	stderr	
iv.	stdprn *	
v.	stdaux *	* supported under DOS

The table 12.1 elaborates the various streams.

Name	stream	Meaning	Device
stdin	Standard input	Standard input device (opened for input)	Keyboard
stdout	Standard output	Standard output device (opened for output)	Screen
stderr	Standard error	Standard error output device (opened for output)	Screen
stdprn*	Standard printer	Standard printer (opened printer for output)	Printer (LPT 1)
stdaux*	Standard auxiliary	Standard auxiliary device (opened for input and output)	Serial port (COM 1)

Table 12.1

The output of a function like printf () or puts () goes to the stream **stdout**.

The scanf () receives its input from stream **stdin**. To perform input and output operations on files, a stream has to be associated with the file.

12.2 Types of Files:

1. Text files

Text files are the normal **.txt** files. You can easily create text files using any simple text editors such as Notepad. When you open those files, you'll see all the contents within the file as plain text. You can easily edit or delete the contents.

They take minimum effort to maintain, are easily readable, and provide the least security and takes bigger storage space

2. Binary files

Binary files are mostly the **.bin** files in your computer.

Instead of storing data in plain text, they store it in the binary form (0's and 1's).

They can hold a higher amount of data, are not readable easily, and provides better security than text files

Difference between Text and Binary files.

Srno	Text Files	Binary Files
1	Data is stored in the form of collection of alphabets and numbers.	Data is stored in the form of binary 0 and 1's.(Low level)
2	Human understandable and readable form.	Machine understandable and readable form.
3	Errors can be identified and detected ,can do modifications.	Identifying errors is difficult and we cannot detect. Modifications not possible.
4	Debugging is very easy.	Debugging is not easy.
5	Text file must be converted into intermediate code or machine understandable code. Here we use language translators to convert high level code into binary code.	Binary file cannot be converted into intermediate language. No need of language translators.
6	Execution is slow as compared to binary files, as text files are not direct executable codes.	Binary codes are direct executable code. Execution is fast
7	e.g. .txt, .docx, pptx, .c	e.g. .bin, .exe,.obj , image,audio,video files

12.3 Operations on Files:

'C' provides various functions to perform on files. These operations include:

1. Opening a file.

2. Reading data from a file.
3. Writing data to a file.
4. Closing a file.
5. Detecting end of a file.

To perform any operation using files in 'C', a pointer must be used.

The File Pointer

A file pointer is a pointer variable of type FILE which is defined in stdio.h. The type FILE defines various properties about the file including its name, status and current position.

Basically, a file pointer identifies a specific disk file. This pointer is used by the stream associated with it to specify where to perform the operations in the file.

Syntax:

FILE *file pointer

Example:

FILE *fp;

Here, fp is declared as a pointer to a file. All operations on the file (which it points to) will be performed using the pointer fp.

Opening a File:

Before performing any file, I/O operation, the file must be opened. The process of opening a stream for use and linking a disk to it, is called *opening a file*.

The fopen () function is used to open a file.

Syntax:

FILE * fopen (const char *filename, const char *mode) ;

fopen() requires two arguments:

1. First argument is the name of the File.
2. Second argument is the mode in which the file is to be opened.

e.g. FILE *fp;

fp= fopen("L1.txt","w")

If the open operation is successful, fopen returns a pointer to type FILE. If it fails, it returns NULL.

The different values of the mode are as below in the Table 12.2

Mode	Meaning
r	Opens a file for reading. If file does not exist fopen() returns NULL.
w	Creates a file for writing. If specified file does not exist, a new one is created. If it exists, existing contents will be overwritten.

a	Opens a file for appending. If specified file does not exist, it is created.
r+	Opens the file for reading and writing. If specified file does not exist, it is created. If it exists, contents can be read and new contents can be added.
w+	Opens the file for reading and writing. If specified file does not exist, it is created. If it exists, it is overwritten by new data which can be read.
a+	Opens a file for reading and appending. If the specified file does not exist, it is created. If it exists, new data is appended to the end of the file.

Table 12.2

The default type of file is text, if one wants to open in binary mode just append b after the mode, e.g. rb, wb

Closing a File: Once the operations on the file are over, one must close the file using `fclose()` function. This will ensure that all the information related to the file is flushed out of the buffers.

Syntax:

```
int fclose(FILE * file pointer)
```

e.g. `fclose(fp);`

To close all the opened files except the predefined we can use:

```
int fcloseall(void);
```

Detecting end of a file: It is necessary to detect the end of a file. In text file a special constant EOF defined in `stdio.h` as -1 denotes the end of a file. In binary files we use the library function **`feof()`** which returns TRUE if end of file is reached, it can also be used with text files.

Syntax:

e.g.

```
if(feof(fp)==1)
```

```
printf("End of File encountered");
```

Reading and Writing File Data:

File I/O operations can be performed in three ways

1. Character/string input output to read /write characters or line of characters. These operations are commonly used with text files only.
Functions: `fgetc`, `fputc`, `fgets`, `fputs`, `getw`, `putw`
2. Formatted I/O to read/write formatted data. This can only be used with text mode files.
Functions: `fprintf`, `fscanf`
3. Direct input/output to read or write blocks of data directly. This method is used only for Binary files.
Functions: `fread`, `write`

12.4 Standard Library and Input Output Function.

Reading and Writing characters:

1. **getc () and fgetc ()** : Both are used to read a single character from a file or stream. The difference is that `getc` is a macro while `fgetc` is a function.

Syntax: `int fgetc(FILE *fp);`

They return a single character. When used with stream `stdin`, they input a character from the Keyboard.

e.g.

```
ch=fgetc (fptr);  
ch=getc (stdin);
```

2. **putc() and fputc():** Both are used to write a single character to the specified stream. If the stream is `stdout`, they display it on the standard output device.

Syntax: `int fputc(int h, FILE *fp);`

e.g.

```
char ch= 'A' ;  
putc ( ch,fptr);  
fputc ( ch,stdin);
```

3. **fgets()** It reads a line of characters from a file.

Syntax: `char *fgets (char *str, int n, FILE *fp)`

- `str` points to the string where the read string has to be stored.
- `n` is the maximum number of characters to be read. Characters are read until a new line is encountered or `(n-1)` characters have been read, whichever occurs first.

e.g.

```
char name [80] ;  
fgets (name,80,fptr);
```

4. **fputs()** writes a line of characters to a stream. It does not add a new-line to the end of the string. If it is required, then the new line has to be explicitly put.

Syntax : `char fputs(char *str, FILE *fp);`

e.g.

```
char city [ ] = " Nashik" ;  
fputs ( city, fptr ) ;
```

Write a program to enter a line of text till you press enter and save it into a text file 'bca.txt'.

```
#include<stdio.h>  
#include<conio.h>  
#include<stdlib.h>  
FILE *fp;  
main()  
{  
char ch;  
clrscr();  
fp=fopen("bca.txt","w");  
if(fp==NULL)  
{
```

```

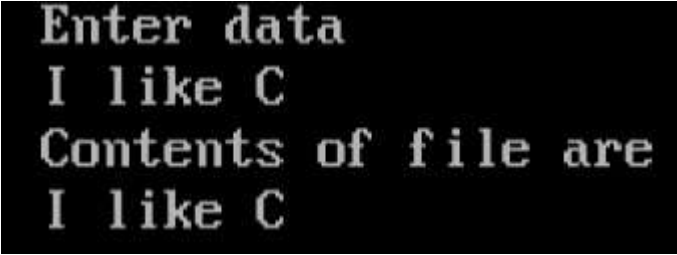
    printf("File creation error");
    getch();
    exit(1);
}
else
{
    printf("Enter data\n");
    while((ch=getchar())!='\n')
        fputc(ch,fp);
    fclose(fp);
}

```

```

fp=fopen("bca.txt","r");
if(fp==NULL)
{
    printf("File reading error");
    getch();
    exit(1);
}
else
{
    printf("Contents of file are\n");
    while((ch=fgetc(fp))!=EOF)
        printf("%c",ch);
    fclose(fp);
}
getch();
}

```



```

Enter data
I like C
Contents of file are
I like C

```

Formatted File Input / Output Functions:

The functions seen above can handle only single data types. In order to deal with multiple data types, formatted file I/O functions are used, `fprintf( )` is used for output and `fscanf()` is used for input.

1. `fprintf()`

This is similar to `printf` except that a pointer to a file must be specified. The data is written to the file associated with the pointer.

Syntax : `int fprintf (FILE *fp, char *format, argument list);`

The format string is the same as used for `printf`. The argument list are the names of variables to be output to the specified stream.

e.g.

```
char name []="Tom" ;  
int age=20 ;  
float amount=1005.75;  
fprintf (fptr, "%s %d %f", name, age, amount) ;
```

2. `fscanf()`

This is similar to `scanf` except that input comes from a specified stream instead of `stdin`.

Syntax: `int fscanf (FILE *fp, char *format , argument list) ;`

The format string is the same as used for `scanf`. The argument list contains the variables in which the data is to be read into.

e.g.

```
fscanf (fp, "%s%d%f", name, &age, &amount);
```

Write a program to create a file having following structure acno,name,bal .

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<ctype.h>
struct emp
{
int acno;
char name[20];
int bal;
}e;
FILE *fp;
main()
{
char resp;
clrscr();
fp=fopen("bank.dat","w+");
if(fp==NULL)
{
printf("File creation error");
getch();
exit(1);
}
else
{
printf("Enter data\n");
do
{
printf("\nEnter Acno :");
scanf("%d",&e.acno);
printf("Enter Name :");
scanf("%s",e.name);
printf("Enter balance :");
scanf("%d",&e.bal);
fprintf(fp,"%d %s %d\n",e.acno,e.name,e.bal);
printf("Any more additions (Y/N)");
resp=getche();
}while(toupper(resp)=='Y');
rewind(fp);
printf("\nData is \n");
while((fscanf(fp,"%d%s%d",&e.acno,e.name,&e.bal))!=EOF)
printf("%d %s %d\n",e.acno,e.name,e.bal);
fclose(fp);
getch();
}
```

}

Enter data

Enter Acno :1

Enter Name :Raj

Enter balance :3000

Any more additions (Y/N)y

Enter Acno :2

Enter Name :Tom

Enter balance :4000

Any more additions (Y/N)n

Data is

1 Raj 3000

2 Tom 4000

Direct File Input /Output :

This is used only with binary-mode files. It is used to read or write block of data.

1. fwrite() :

This function writes a block of data from memory to a binary file. It returns the number of elements written.

Syntax : int fwrite (void *buf, int size, int count ,FILE *fp);

- buf is a pointer to the region of memory which holds the data to be written to the file.
- Size specifies the size in bytes of individual data items.
- Count specifies the number of items to be written.

e.g. To write to a single float variable x to a file, the statement will be

```
fwrite (&x, sizeof (float) , 1 , fptr ) ;
```

To write a 100 element integer array a to the file,

```
fwrite (a,sizeof (int) , 100, fptr );
```

Apr.2015-5M

Explain the following functions with example.

- a. fprintf()
- b. fscanf()
- c. fread()
- d. fwrite()

2. fread() :

It reads a block of data from binary file and assigns it to the specified memory address. It returns the number of values read.

Syntax: int fread(void *buf, int size, int count , FILE *fp) ;

- Buf is the pointer to which memory receives data read from the file(i.e., it is the address of the variable).
- Size specifies the size in bytes of individual data items being read.
- Count specifies the number of items to be read.

e.g. fread (&num,sizeof (int) ,1 , fp) ;

This reads an integer from the file and assigns it to num. If we have a structure variable emp and its members have to be read from the file, fread can be used.

```
fread ( &emp, sizeof (emp) , 1 , fp) ;
```

Write a program to write and read employee information from a binary file (name and salary).Also display the employee details having the maximum salary.

```
#include<stdio.h>
#include<conio.h>
struct employee
{
    char name [20] ;
    float sal ;
};
void main()
{
    FILE *fp;
    struct employee emp[20],e;
    float maxSalary=0 ;
    int i,n;
    clrscr();
    if((fp=fopen("employee.in", "wb"))==NULL)
    {
        printf("Error opening file") ;
        exit(1);
    }
    printf("\n How many employees? : ") ;
    scanf("%d",&n);
    for(i=1;i<=n;i++)
    {
        printf("\nEnter the name and salary") ;
        scanf("%s%f",e.name,&e.sal) ;
        fwrite(&e,sizeof(e),1,fp) ;
    }
    fclose(fp) ;
    fp=fopen ("employee.in" ,"rb") ;    /* reopen file */
    if(fp==NULL)
    {
        fprintf(stderr,"Error opening file") ;
        exit(1);
    }
    /*Read data of n employees from file into array emp */
    if((fread(emp,sizeof(struct employee),n ,fp))!=n)
    {
        fprintf(stderr,"Error reading file" ) ;
        exit(1) ;
    }
    fclose ( fp ) ;
    /*Display array emp */
    for ( i=0;i<n;i++)
    {   printf( "\n Name=%s salary =%f" ,emp[i].name,emp[i].sal) ;
        if(emp[i].sal>maxSalary )
        {
            maxSalary = emp[i].sal ;
        }
    }
}
```

```

        e = emp[i] ; // save employee in e
    }
}
printf( "\nThe employee details having maximum salary is : " );
printf( "\n Name=%s Salary=%.2f" , e.name , e.sal );
getch();}

```

The screenshot shows a terminal window with the following text:

```

How many employees? : 2
Enter the name and salaryRaj 3000
Enter the name and salaryTom 4000
Name=Raj salary =3000.000000
Name=Tom salary =4000.000000
The employee details having maximum salary is :
Name=Tom Salary=4000.00

```

Additional File handling Functions :

1. **fflush()**

This causes the contents of an open output stream to be written to the specified stream, its buffer is cleared.

Other Functions

- i. fflush()
- ii. remove()
- iii. rename()

Syntax : fflush (FILE * fp);

e.g. fflush (stdin);

2. **remove():**

This deletes the file specified. If it is open, be sure to close it before removing it.

Syntax : int remove (const char * filename);

e.g. remove ("stud.dat");

3. **rename():**

This function changes the name of an existing disk file.

Syntax : int rename(const char * old name, const char *newname);

Both files must be in the same disk drive.

e.g. rename ("L1.txt" , "newL1.txt");

12.5 Random Access to Files:

Every open file has a position pointer or a position indicator associated with it. This indicates the position where read and write operation takes place.

In all earlier programs, we read the file sequentially. However, C provides functions to control the position pointer by means of which data can be read from or written to any position in the file.

These functions are:

1. **ftell()** :

This function is used to determine the current location of the file pointer.

Syntax: long ftell (FILE * fp) ;

It returns a long integer that gives the current pointer position in bytes from the start of the file (i.e., it gives the offset in bytes the beginning of the file). The beginning of the file is considered at position 0.

e.g.

```
fp=fopen ( "in. txt" , " r" ) ;
```

```
printf ( " %ld", ftell (fp) ) ;
```

Here the output will be 0, because when a file is opened, the pointer points to the beginning of the file.

```
fp=fopen ( "in. txt" , "a" ) ;
```

```
printf ( "%ld", ftell (fp)) ;
```

Here the output will be the number of bytes in the file because when a file is opened in the append mode, the pointer points to the end of the file.

2. **rewind () :**

This function sets the pointer to the beginning of the file.

Syntax : void rewind (FILE * fp);

This function can be used if we have read some data from a file and want to start reading from the beginning of the file, without closing and reopening the file.

e.g.

```
rewind ( fp ) ;
```

```
printf ( “%ld” , ftell (fp) ) ;
```

This will yield 0 since rewind positions the pointer at the start of the file.

3. **fseek () :**

The function fseek allows the pointer to be set to any position in the file.

Syntax: fseek (FILE *fp, long Offset , int origin) ;

- Offset indicates the distance in bytes that the position pointer has to be moved by.
- Origin indicates the reference point in the file with respect to which the pointer is moved offset number of bytes.

These can be three values for origin with symbolic constants which are defined in stdio.h.

Constant	Value	Meaning
SEEK_SET	0	Moves the pointer offset bytes from beginning of file.
SEEK_CUR	1	Moves the pointer offset bytes from its current position.
SEEK_END	2	Moves the pointer offset bytes from the end of the file.

e.g.

1. fseek (fptr, 0 , SEEK_END) ;

This positions the pointer to the end of the file.

2. struct emprec

```
{ _  
_  
} rec ;
```

If a file contains four records of struct emprec as shown.

→

rec1	rec2	rec3	rec4
------	------	------	------

- i. fseek (fp, 2* sizeof (struct emprec) , SEEK_SET) ;

This positions the pointer to the beginning of rec3.

fread (& rec , sizeof (struct emprec) , fp) ;

will read the third record.

- ii. fseek (fp, sizeof (struct emprec) , SEEK_END) ;

will position the pointer to the beginning of the 4th record.

Oct.2011 – 1M

State the use of rewind() function.

```
fseek ( fp, -2*sizeof ( struct emprec ) , SEEK_CUR ) ;  
will now position it to the start of the second record.
```

3. **ftell** can be used with **fseek** to find the size of the file.

```
fseek ( fp, 0 ,SEEK_END ) ;  
printf ( “%ld” , ftell ( fp ) ) ;
```

This will display the size of the file in bytes. We can use 0,1,2 in place of SEEK_SET,SEEK_CUR and SEEK_END.

SUMMARY:

In this Chapter we studied the concept of File handling . Files allow us to store the persistent data permanently. Two type of files were discussed , they were text files and binary files. Various functions to open and close a file were discussed. How to write and read a character for text files was studied. Further we saw fprintf and fscanf a category of formatted input and output functions. fwrite and fread functions were discussed to write and read binary files. Additional file handling functions such as fflush(), remove(), rename() , ftell(), rewind(), fseek() were also seen.

SOLVED PROBLEMS AND SELF CHECK QUESTIONS

SOLVED PROBLEMS

- 1) If the mode includes b after the initial letter, what does it indicates?
 - a) text file
 - b) big text file
 - c) binary file**
 - d) blueprint text
- 2) Which one of the following is correct syntax for opening a file.
 - a) FILE *fopen(const *filename, const char *mode)**
 - b) FILE *fopen(const *filename)
 - c) FILE *open(const *filename, const char *mode)
 - d) FILE open(const*filename)
- 3) EOF is an integer type defined in stdio.h and has a value ___
 - a) 1
 - b) 0
 - c) NULL
 - d) - 1.**
- 4) Which of the following true about FILE *fp.
 - a) FILE is a keyword in C for representing files and fp is a variable of FILE type.
 - b) FILE is a stream
 - c) FILE is a buffered stream
 - d) FILE is a structure and fp is a pointer to the structure of FILE type**
- 5) File manipulation functions in C are available in which header file?
 - a) streams.h
 - b) stdio.h**
 - c) stdlib.h
 - d) files.h

SELF-TEST 01

MCQ 01: The first and second arguments of fopen() are:

- a. A character string containing the name of the file & the second argument is the mode.
- b. A character string containing the name of the user & the second argument is the mode.
- c. A character string containing file pointer & the second argument is the mode.
- d. A character string containing file pointer & the second argument is the mode.

MCQ 02: What is the function of the mode 'w+'?

- a. Create text file for writing, discard previous contents if any

- b. Create text file for update, discard previous contents if any
- c. Create text file for writing, do not discard previous contents if any
- d. Create text file for update, do not discard previous contents if any

MCQ 03: FILE is of type _____

- a. int type
- b. char * type
- c. struct type
- d. None of the mentioned

MCQ 04: fseek() should be preferred over rewind() mainly because

- a. rewind() doesn't work for empty files
- b. rewind() may fail for large files
- c. In rewind, there is no way to check if the operations completed successfully
- d. All of the above

MCQ 05: fwrite() can be used only with files that are opened in binary mode.

- a. true
- b. false

EXERCISE

- Q1. Discuss the concept of streams? Enlist them.
- Q2. Differentiate between binary and text files.
- Q3. Explain the following functions with their usages.
 - 1 fopen 2. fclose 3. ftell() 4. fflush
 - 5 fprintf 6. fscanf 7.fread 8. fwrite
- Q4. Explain getc(), fgetc(), putc(), fputc() functions.
- Q5. What is the use of rewind() function.
- Q6. Discuss the various modes of opening a file.

REFERENCE MATERIAL

REFERENCES: [geeksforgeeks.com](https://www.geeksforgeeks.com/), [unstop.com](https://www.unstop.com/)

<https://www.w3schools.com/c/>, [javatpoint.com](https://www.javatpoint.com/), [idiabix.com](https://www.idiabix.com/)

<https://www.examveda.com/>, <https://www.sankalandtech.com/>, <https://gtu-mcq.com/>, <https://www.sanfoundry.com/>

<https://testbook.com/>, <https://letsfindcourse.com/>

[https://www.lkouniv.ac.in/site/writereaddata/siteContent/202005082120331143zeeshan_Difference Table_4.pdf](https://www.lkouniv.ac.in/site/writereaddata/siteContent/202005082120331143zeeshan_Difference_Table_4.pdf)

MOOCs

YOUTUBE VIDEOS

WIKIPEDIA

BLOGS : UNSTOP.COM

OER (SLIDES, PDFs, TUTORIALS)

REFERENCE BOOKS

ANSWERS TO SELF-CHECK QUESTIONS

1.(a) 2. (b) 3. (c) 4. (c) 5. (a)

UNIT 13: STORAGE CLASSES

UNIT OBJECTIVES

After successful completion of this unit, you will be able to understand.

- ❖ Scope and Extent.
- ❖ Storage Classes in a single source file.
- ❖ auto, extern, static , register.

12.1 Introduction to Storage Classes: Scope of Variables.

There are two different ways to characterize variables i.e. by **data type** and **by storage class**. Data type refers to the type of the information represented by a variable. e.g. integer number, floating point number character etc. **Storage class** refers to the permanence of a variable and its scope within the program, that is the portion of the program over which the variable is recognized.

We have four storage classes in 'C': automatic, external, static and register. These storage classes are identified by the keywords auto, extern, static and register.

The storage class of a variable determines :

1. where a variable is stored.
2. Its default initial value.
3. Scope of the variable.
4. Lifetime of the variable.

Automatic storage class:

This is the default storage class of variables that are declared within function and are local to the function in which they are declared. In order to explicitly declare a variable which belongs to this class the keyword **auto** is used.

e.g. auto int i;

This variable comes into existence only within the function where it is declared and ceases to exist after the function is exited hence termed as automatic.

Features:

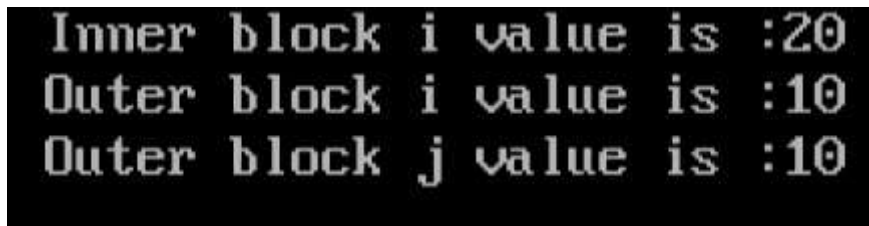
1. **Storage** : Main memory.
2. **Scope** : Local to the block where it is declared.(Block Scope)
3. **Lifetime** : It exists as long as control remains in the block where it is declared.
4. **Default initial value** : Garbage.

Write a program to demonstrate the Concept of auto storage class.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    auto int i = 10,j;
    {
        auto int i = 20;
        clrscr();
        printf("Inner block i value is :%d\n",i);
    }
    printf("Outer block i value is :%d\n",i);
}
```

```
printf("Outer block j value is :%d\n",i);  
getch();}
```

In the above variables 'i' variable since blocks.
any garbage



```
Inner block i value is :20  
Outer block i value is :10  
Outer block j value is :10
```

program the two are different they are in different Variable j contains value.

Static Storage Class

Local variables are automatic by default, which means that every time the function in which they are declared is called, they are created and destroyed when the function ends. They do not retain their value between function calls.

However, in many cases it is required that a variable retains its value between function calls. This is possible if the variable is declared belonging to the **static storage class**.

Syntax : static data_type variable;

e.g. static int x ;
static long factorial ;

Types of static variables

1. **Local static variables:** These variables have block or function scope and they retain their value between calls to the function.
2. **Global static variables:** They are global to the file in which they are defined. Unlike an ordinary external variable which is visible to all functions in the file and functions in other files, a static external is visible only to functions in its own file.

Features

1. **Storage :** Main Memory.
2. **Scope :** Block or file scope depending upon where it is declared.
3. **Lifetime :** Persists between function calls if scope is block Scope.
4. **Default Initial value :-** Zero.

Write a program to demonstrate the Concept of static storage class.

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

int fibo(int);
int getdata(void);

main()
{
    int n,term,tno;
    clrscr();
    n=getdata();
    for(tno=1;tno<=n;tno++)
    {
        term =fibo(tno);
        printf("%d \t",term);
    }
    getch();
}

int fibo(int tno)
{
    static int f1=0,f2=1,f3;

    if(tno==1)
        f3=f1;
    else
        if(tno==2)
            f3=f2;
        else
        {
            f3=f1+f2;
            f1=f2;
            f2=f3;
        }
    return(f3);
}
```

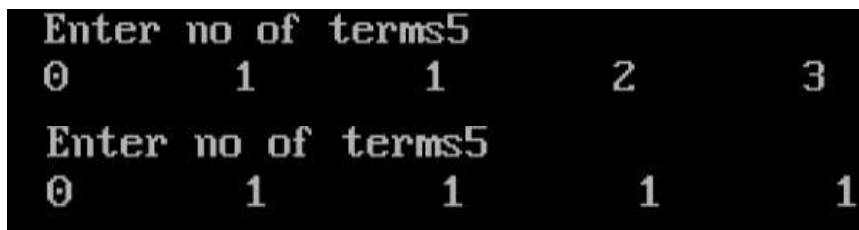
```
int getdata(void)
{
    int num;

    printf("Enter no of terms");

    scanf("%d",&num);

    return(num);
}
```

The below static keyword function fibo.



```
Enter no of terms5
0      1      1      2      3
Enter no of terms5
0      1      1      1      1
```

output is when the is removed from the

The result shows that every time function fibo is called f1, f2 are initialized to 0 and 1 respectively only once and its value persist between function calls.

Extern Storage Class

Variables belonging to this class are also called as global variables or external variables. They are declared outside all functions and are accessible to all the functions in that source code file.

If the program code extends over two or more files, the extern keyword can be used to declare the variable within a function that uses the global variable.

Syntax : extern data_type variable;

e.g. extern int k;

Features :

1. **Storage** : Main memory.
2. **Scope** : File scope.
3. **Lifetime** : It exists as long as the program which uses the variable is running. It retains its value between functions.
4. **Default initial value** :- Zero.

```

#include<stdio.h>
#include<stdio.h>
void display(void);

int n=5 ;           // initialized.
main ( )
{
    extern int n ;  // declared
    clrscr();
    printf("\n%d", n) ;
    display() ;
    getch();
}
void display (void)
{
    extern int n ;  //declared
    printf ("\n%d",n) ;
}

```



The declaration within the functions indicates that the function uses an external variable which is defined elsewhere. If both these functions are in the same source code file, the declarations are not required. If the variable n is to be used in functions written in separate source code files, the declaration using the extern keyword is required.

Advantages

- i. Use of global variable simplifies communication, i.e., they need not be passed to functions,(thereby making argument lists shorter) and any function can use them whenever required.
- ii. Symbolic constants are often declared globally.

Disadvantages

- i. By using external variables, the principles of modular programming, i.e., data isolation is violated.
- ii. Even when not required, external variables persist in memory.

Register Storage Class

The register keyword is used to tell the compiler to store the variable in a CPU register rather than in main memory. The register variables have similar features as the automatic storage class except for the storage location.

Syntax : register data_type variable;

e.g.

```
register int i ;
```

```
register char ch ;
```

Features

- 1. Storage** : CPU registers.
- 2. Scope** : Block scope.
- 3. Lifetime** : Exists as long as control is within the Block where it is defined.
- 4. Default initial value** : Garbage.

Advantages of register variables

The CPU has its own limited storage locations, which it uses for actual data operations. These locations are called registers.

To manipulate data and perform operations, the CPU moves data back and forth between the memory and registers, which takes a finite amount of time. Thus, if a particular variable is kept in the register itself, the CPU can access it faster. Hence, variables, which are heavily used, may be declared of this type so that execution is faster.

Limitations

- i. There are only a limited number of registers in the CPU. So, a register may not be available for the variable. In such a case, the variable is treated as an Ordinary automatic variable.
- ii. Most compilers allow this storage class to be used only with integer data type (int or char).
- iii. The unary & operator (address of) cannot be used with these variables either explicitly or implicitly.
- iv. It cannot be used with either static or external storage classes.
- v. It cannot be used for structures, arrays or unions.

The following table summarizes the storage classes, scope and initializations.

Storage Class	Variable is declared	Visibility	Remarks
static	Outside a function.	Anywhere within the file.	Are initialized only once, values are retained through function calls, default initial value is zero.
	Inside a function block.	Function/block scope.	
extern	Outside a function.	Anywhere within the file.	If they are to be used in multiple files, they have to be declared in each function using the extern keyword. Initialization can be done only once-outside the functions.
register	Inside a Function/block.	Function/block scope.	Limited number of registers, restriction on the type of variables, cannot use pointers for register variables, no default value.
auto	Inside a Function/block.	Function / block scope, i.e., local to the function/block.	Variable is initialized each time the function / block is entered, no default value. Does not exists outside function block where declared.

SUMMARY:

In this Chapter we studied the concept of Storage classes , Scope and extent of variables . Different storage classes auto, extern, static , register were discussed.

SOLVED PROBLEMS AND SELF CHECK QUESTIONS

SOLVED PROBLEMS

- 1) In 'C' language, local variables ___ can retain their value between function calls.
 - a) register
 - b) extern
 - c) **static**
 - d) auto
- 2) Which of the following operators cannot be applied if the storage class of a variable is specified as 'register' in 'C' language.
 - a) !
 - b) --
 - c) **&**
 - d) ++
- 3) Which of the following is not a valid storage class.
 - a) static
 - b) **dynamic.**
 - c) auto
 - d) register
- 4) Which if the following is the default class for local variable in C
 - a) **auto**
 - b) register
 - c) extern
 - d) static
- 5) What is the default storage class for global variable,
 - a) auto
 - b) static
 - c) extern
 - d) register

SELF-TEST 01

MCQ 01: Which storage class is used for local variables.

- a) register
- b) auto
- c) static
- d) extern

MCQ 02 :What is the primary purpose of the static storage class.

- a) extend scope
- b) preserve value
- c) allocate memory
- d) register access

MCQ 03 :When does a register storage class make sense to use.

- a) for global variables
- b) for function arguments
- c) for local variables
- d) for static variables

MCQ 04 : What is the significance of the extern storage class.

- a) local scope
- b) external linkage
- c) automatic allocation
- d) static duration

register access

MCQ 05 : in which storage class is the variables storage duration throughout the programs execution.

- a) extern
- b) static
- c) auto
- d) register

EXERCISE

- Q1. Explain the concept of auto storage classes. Explain with an example.
- Q2. Explain the concept of static storage classes. Explain with an example.
- Q3. Explain the concept of extern storage classes. Explain with an example.
- Q4. Explain the concept of register storage classes. Explain with an example.
- Q5. When is the register storage class most useful.
- Q6. What are the advantages and limitations of the register storage class.
- Q6. Differentiate between automatic and static storage class.

REFERENCE MATERIAL

REFERENCES: [geeksforgeeks.com](https://www.geeksforgeeks.com/), [unstop.com](https://www.unstop.com/)

<https://www.w3schools.com/c/>, [javatpoint.com](https://www.javatpoint.com/), [idiabix.com](https://www.idiabix.com/)

<https://www.examveda.com/>, <https://www.sankalandtech.com/>, <https://gtu-mcq.com/>, <https://www.sanfoundry.com/>

<https://testbook.com/>, <https://letsfindcourse.com/>

[https://www.lkouniv.ac.in/site/writereaddata/siteContent/202005082120331143zeeshan_Difference Table_4.pdf](https://www.lkouniv.ac.in/site/writereaddata/siteContent/202005082120331143zeeshan_Difference_Table_4.pdf)

MOOCs

YOUTUBE VIDEOS

WIKIPEDIA

BLOGS : UNSTOP.COM

OER (SLIDES, PDFs, TUTORIALS)

REFERENCE BOOKS

ANSWERS TO SELF-CHECK QUESTIONS

1.(b) 2. (b) 3. (b) 4. (b) 5. (b)

UNIT 14: THE C PREPROCESSOR

UNIT OBJECTIVES

After successful completion of this unit, you will be able to understand.

- ❖ Introduction and Features of C Preprocessor.
- ❖ Macro Expansion.
- ❖ Macro with Arguments.
- ❖ Macros Versus Functions.
- ❖ File Inclusion.
- ❖ Conditional Compilation.

14.1 Introduction to Preprocessors.

A Preprocessor is a program that processes or analyzes the source code file before it is given to the compiler. It takes action according to special instructions called Preprocessor directives which begin with #.

Following tasks are performed by the Preprocessor.

- Joins any lines that end with a backslash character into a single line.
- Divides the program into a stream of tokens.
- Remove comments, replacing them by a single space.
- Processes preprocessor directives and expands macros.
- Replaces escape sequences by their equivalent internal representation.
- Concatenates adjacent constant character strings.
- Replaces trigraph sequences by their equivalents. Trigraph sequences are used to handle non-ASCII character sets.

14.2 Preprocessor Directives.

Preprocessor directives are special instructions for the Preprocessor.

1. They begin with a '#' which must be the first non-space character on the line.
2. They do not end with a semicolon.

Preprocessor directives come under **three** categories:

1. File inclusion directive
2. Macro substitution directive
3. Conditional compilation directive

14.2.1 File Inclusion Directive:

The file inclusion directive is **#include**. This directive instructs the compiler to include contents of the specified file, i.e., it inserts the entire contents of the file at that position.

Syntax : #include <filename> OR #include "filename"

- In the first format, the filename is searched in standard directories only.

- In the second, the file is first searched in the current directory. If it is not found there, the search continues in the standard directories.
- Any external file-containing user defined functions, macro definitions, etc. can be included.
- An included file can include other files.

e.g. `#include<stdio.h>`

`#include<myhead.h>`

There are many preprocessor Directives available. Some of them are:

- `#include`: It inserts a header file from another file.
- `#define`: It is a preprocessor used as a macro.
- `#undef`: It undefines a preprocessor macro.
- `#if`: Tests if a compile-time condition is true.
- `#ifdef`: Return true if the macro is defined.
- `#ifndef`: Return true if the macro is not defined.
- `#else`: The alternative for `#if`.
- `#elif`: else and `#if` in one statement.
- `#endif`: Ends preprocessor conditional.
- `#warning`: It can be used for warnings.

`#error`: It is used to stop compilation.

14.2.2 Macro Substitution Directive :

- The macro substitution directive is **`#define`**. It defines a symbol which represents a value. Wherever the macro name occurs in a program the preprocessor substitutes the code of the macro at that position.
- It is used to define symbolic constants which occurs frequently in a program.

Syntax : `#define macro_id value`

e.g. `#define PI 3.14`

`#define TRUE 1`

`#define FALSE 0`

`#define pf printf`

`#define sf scanf`

`#define message "Welcome to C"`

State the use of “#” define.

Every occurrence of the macro-id in the program will be replaced by its corresponding value.

e.g. `area = PI * pow(r,2);`

14.3 Argumented Macros :

A macro can be defined with arguments. Such a macro is also called a function macro because it looks like a function.

It represents a small executable code. Each time the macro name is encountered, its arguments are replaced by the actual arguments in the program.

e.g.

```
#define HALFOF (x)    x/2
#define SQR (x)      x*x
#define MAX (x,y)     ( ( x ) > (y) ? (x) : (y) )
```

Wherever the macro occurs in the program, its argument (example:x) is replaced by the variable or value used in the program.

e.g.

```
result = HALFOF (10) ;    // replaced by result = 10/2 ;
ans = SQR (a) ;           // replaced by ans = a*a ;
```

It is advisable to enclose the arguments in (). If the arguments are not enclosed in (), it may yield wrong results.

e.g.

```
result=HALFOF (10+2) ;
```

This will be evaluated as result=(10+2/2); and will give incorrect results. The correct way to define the macro would be:

```
#define HALFOF (x)     ( (x) /2)
```

Now, result=((10+2)/2); which will give the correct result.

Nested Macros: It is possible to call one macro inside another macro

```
#define SQR(x) ( (x) * (x))
#define CUBE(x) (SQR(x) *(x))
#define SIXTH(x) (CUBE(x)*CUBE(x))
```

14.4 Macros Versus Functions:

No.	Macro	Function
1.	A macro is small in size.	A function can contain several lines of code.
2.	A macro is preprocessed.	A function is compiled.
3.	A macro is expanded in the code, i.e., its code replaces the macro name.	Function code is not inserted into the program when a function is called.
4.	The program size increases.	The program size does not increase.
5.	Execution of macros is faster since the code is replaced.	Functions execute slower since control has to be passed to the function and then returned back to the program.
6.	No type checking is done for macros. Hence there can be side-effects or unforeseen errors.	Strict type checking is done for functions.
7.	Usually used for small code which appears many times.	Usually used for large code which needs to be executed many times.

14.5 Conditional Compilation in C Programming

Conditional compilation in C programming is a powerful feature that allows developers to selectively compile specific sections of code based on certain conditions. This feature is particularly useful when dealing with code that needs to work on different platforms, with variations in operating systems, hardware configurations, or compiler-specific directives

What are the types of preprocessors used in conditional compilation?

These are the preprocessor directives, commonly used for conditional compilation in C programming.

```
#if  
#ifdef  
#ifndef  
#else  
#elif  
#if defined  
#endif
```

Conditional compilation can be extremely useful in many scenarios. For example, you can use it to include platform-specific code, define feature flags, or enable debugging statements in development builds while excluding them in production builds. Conditional compilation allows developers to write flexible and customized code for different scenarios without cluttering the codebase with unnecessary statements. It enhances code readability, maintainability, and portability.

Write a macro named area which will calculate are of a rectangle. Also create a macro name design to print a pattern.

[Use the concept of multiline Macros]

```
#include<stdio.h>

#define area length*breadth

#define design for(i=1;i<=5;i++){ \
                for(j=1;j<=i;j++) { \
                    printf("*"); \
                } \
                printf("\n"); \
            }

main()
{

    int length, breadth, i,j;
    clrscr();
    printf("Enter Length ");
    scanf("%d",&length);
    printf("Enter Width ");
    scanf("%d",&breadth);
    printf("Area is  %d \n",area);
    design;
    getch();
}
```



```
Enter Length 4
Enter Width 5
Area is 20

*
**
***
****
*****
```

SUMMARY:

In this Chapter we studied the concept of C Preprocessor which is collection of special statements called directives that are executed at the beginning of the Compilation process. The `#include` and `#define` statement are preprocessor directives. Additional preprocessor directives studied were `#if`, `#elif`, `#else`, `#endif`, `#ifdef`, `#ifndef`, `#line`, `#undef`. Differences between Macros and functions were discussed.

SOLVED PROBLEMS AND SELF CHECK QUESTIONS

SOLVED PROBLEMS

1) Which one of the following is invalid macro in C programming?

- a) #pragma
- b) #error
- c) #ifndef
- d) **#elseif**

2) which keyword is used to define the macros in 'C'?

- a) macro
- b) define
- c) **#define**
- d) #macro

3) Which symbol is used to declare the preprocessor directives?

- a) #
- b) \$
- c) *
- d) ^

4. How many types of macros are there in 'C'?

- a) 1
- b) **2**
- c) 3
- d) 4

5. In which stage the following code

```
#include<stdio.h>
```

gets replaced by the contents of the file stdio.h

- a) During editing
- b) During linking
- c) During execution
- d) **During preprocessing**

SELF-TEST 01

MCQ 01: In C , which directive is used to conditionally include code based on preprocessor macros.

- a) #ifdef

- b)** `#ifndef`
- c)** `#if`
- d)** `#else`

MCQ 02 : Which of the following are correct preprocessor directives in C?

- 1: `#ifdef`
- 2: `#if`
- 3: `#elif`
- 4: `#undef`

- a)** 1, 2
- b)** 4
- c)** 1, 2, 4
- d)** 1, 2, 3, 4

MCQ 03 : If the file to be included doesn't exist, the preprocessor flashes an error message.

- a)** True
- b)** False

MCQ 04 : Preprocessor directive `#undef` can be used only on a macro that has been `#define` earlier

- a)** True
- b)** False

MCQ 05 : What is the purpose of the C preprocessor in C Programming

- a)** Compile Code
- b)** Optimize Code
- c)** Preprocess code before compilation ans
- d)** Execute code

EXERCISE

- Q1. Explain the concept of auto storage classes. Explain with an example.
- Q2. Explain the concept of static storage classes. Explain with an example.
- Q3. Explain the concept of extern storage classes. Explain with an example.
- Q4. Explain the concept of register storage classes. Explain with an example.
- Q5. When is the register storage class most useful.
- Q6. What are the advantages and limitations of the register storage class.
- Q6. Differentiate between automatic and static storage class.

REFERENCE MATERIAL

REFERENCES: [geeksforgeeks.com](https://www.geeksforgeeks.com/), [unstop.com](https://www.unstop.com/)

<https://www.w3schools.com/c/>, [javatpoint.com](https://www.javatpoint.com/), [idiabix.com](https://www.idiabix.com/)

<https://www.examveda.com/>, <https://www.sankalandtech.com/>, <https://gtu-mcq.com/>, <https://www.sanfoundry.com/>

<https://testbook.com/>, <https://letsfindcourse.com/>

[https://www.lkouniv.ac.in/site/writereaddata/siteContent/202005082120331143zeeshan Difference Table 4.pdf](https://www.lkouniv.ac.in/site/writereaddata/siteContent/202005082120331143zeeshan%20Difference%20Table%204.pdf)

https://embetronicx.com/tutorials/p_language/c/preprocessor-in-c/

MOOCs

YOUTUBE VIDEOS

WIKIPEDIA

BLOGS : UNSTOP.COM

OER (SLIDES, PDFs, TUTORIALS)

REFERENCE BOOKS

ANSWERS TO SELF-CHECK QUESTIONS

- 1.(c) 2. (d) 3. (a) 4. (a) 5. (c)