

A Complete Notes on

*Especially
For
B.E. First semester
Students*

Programming in C

Based on syllabus of Pokhara University

Prepared By:

Er.Pradip Paudel

Lecturer, Everest Engineering college, sanepa-2 Lalitpur

Email: pradippaudel89@gmail.com

CMP 103.3 Programming in C (3-0-3)

Evaluation:

	Theory	Practical	Total
Sessional	30	20	50
Final	50	-	50
Total	80	20	100

Course Objectives:

The object of this course is to acquaint the students with the basic principles of programming and development of software systems. It encompasses the use of programming systems to achieve specified goals, identification of useful programming abstractions or paradigms, the development of formal models of programs, the formalization of programming language semantics, the specification of program, the verification of programs, etc. the thrust is to identify and clarify concepts that apply in many programming contexts:

Chapter	Content	Hrs.
1 Introduction	History of computing and computers, programming, block diagram of computer, generation of computer, types of computer, software, Programming Languages, Traditional and structured programming concept	3
2 Programming logic	Problems solving(understanding of problems, feasibility and requirement analysis) Design (flow Chart & Algorithm), program coding (execution, translator), testing and debugging, Implementation, evaluation and Maintenance of programs, documentation	5
3 Variables and data types	Constants and variables, Variable declaration, Variable Types, Simple input/output function, Operators	3
4 Control Structures	Introduction, types of control statements- sequential, branching- if, else, else-if and switch statements, case, break and continue statements; looping- for loop, while loop, do—while loop, nested loop, goto statement	6
5 Arrays and Strings	Introduction to arrays, initialization of arrays, multidimensional arrays, String, function related to the strings	6
6 Functions	Introduction, returning a value from a function, sending a value to a function, Arguments, parsing arrays and structure, External variables, storage classes, pre-processor directives, C libraries, macros, header files and prototyping	6

7	Pointers	7
----------	-----------------	----------

Definition pointers for arrays, returning multiple values from functions using pointers. Pointer arithmetic, pointer for strings, double indirection, pointer to arrays, Memory allocation-malloc and calloc

8	Structure and Unions	5
----------	-----------------------------	----------

Definition of Structure, Nested type Structure, Arrays of Structure, Structure and Pointers, Unions, self-referential structure

9	Files and File Handling	4
----------	--------------------------------	----------

Operating a file in different modes (Real, Write, Append), Creating a file in different modes (Read, Write, Append)

Laboratory:

Laboratory work at an initial stage will emphasize on the verification of programming concepts learned in class and use of loops, functions, pointers, structures and unions. Final project of 10 hours will be assigned to the students which will help students to put together most of the programming concepts developed in earlier exercises.

Textbooks:

1. Programming with C, Byran Gottfried
2. C Programming, Balagurusami

References

1. A book on C by A L Kely and Ira Pohl
2. The C Programming Language by Kerighan, Brain and Dennis Ritchie
3. Depth in C, Shreevastav

CHAPTER-1

INTRODUCTION

Generations of computer

- Explain briefly the various generations of computers.[PU:2012 fall]
- Why second generations of computer are better than first generation? Differentiate between third generation and forth generation of computer.
- Write a short notes on:
Generations of computers [PU: 2018 spring]

Generations of computer refers to the different state of advancement of computer technology.

When generations goes up the following characteristics are in effect.

Characteristics	Effects
Size	Decrease
Speed	Increase
Power consumption	Decrease
Processing capability	Increase
Storage capacity	Increase
Reliability and accuracy	Increase

Based on the period of development and the features incorporated, the computers are classified into different generations- First generation to Fifth generation. This is called the computer generation.

1) First Generation (1940-1956)

- **Technology:** Vacuum tubes were used
- **Input/output device:** Punch card was used as input/output devices.
- **Processing speed:** Slow processing speed and measured in millisecond.
- **Programming language:** Machine level language was used for programming.
- **Reliability and accuracy:** The computers were not fully reliable and accurate.
- **Size and cost:** The size of computer was very large and its cost was also expensive.
- **Power consumption and heat emission:** Computer consumed a lot of electricity power and emitted a lot of heat.
- **Examples.** UNIVAC, EDVAC , ENIAC etc.

2) Second Generation (1956-1963)

- **Technology:** Transistors were used
- **Input/output device:** Punch card was used as input/output devices.
- **Processing speed:** Faster processing speed and measured in microsecond.
- **Programming language:** Assembly and High level programming language were used such as COBOL,FORTAN and ALGOL
- **Reliability and accuracy:** More Reliable and accurate than first generations of computers.
- **Size and cost:** Smaller in size and less expensive than first generations of computer.
- **Power consumption and heat emission:** Computer consumed a lot of electricity power and emitted a lot of heat.
- **Examples.** PDP-5,PDP-8,IBM 7090,IBM 1620

3) Third Generation (1964-1971)

- **Technology:** IC(Integrated Circuits)were used
- **Input/output device:** Keyboards and monitors were used as input/output device
- **Processing speed:** Faster than previous generation of computers and processing speed measured in nanosecond.
- **Programming language:** Further development of high level programming language for computer programming.
- **Reliability and accuracy:** Computer became fully reliable and accurate.
- **Size and cost:** Smaller in size and less expensive than Previous generations of computer.
- **Power consumption and heat emission :** Less than previous generations of computers
- **Examples.** CDC11 7600,PDP 11 and IBM 370

4) Fourth Generation (1972 onwards)

- **Technology:** VLSI(Very Large Integration Circuits)
- **Input/output device:** All different types of input/output devices available such as microphone, speaker, scanner ,printer
- **Processing speed:** Faster than previous generation of computers and processing speed increased up to picoseconds and (MIPS) Millions of instructions per second.
- **Programming Language:** Problem-oriented fourth generation language (4GL) is used to develop the program. As well database programming language were used.
- **Operating system** like MS DOS ,MS windows with Graphical User Interface(GUI) and User friendly applications

- **Size and cost:** Smaller in size and less expensive than Previous generations of computer.
- **Power consumption and heat emission :** Less than previous generations of computers
- Examples. Pentium II, Pentium III, Pentium IV, DEC's Alpha

5) Fifth Generation – (present and future)

Fifth generation computing devices, based on artificial intelligence and still are development. Computer of this generation offers Ultra Large Scale Integration (ULSI) technology.

Characteristics:

- Aims to solve highly complex problems, which require reasoning, intelligence and expert knowledge.
- Based on Parallel processing architecture.
- Will use natural language rather than high level language. Processing with quantum computation and molecular technology will radically change the computing in future.

Computer Software

- What is a software? Differentiate tailored and packaged software.
- What is computer software? Explain the different type of software used in today's life.[PU:2018 fall]
- What do you mean by Instructions and software? Explain various types of software.[PU:2019 spring]

Software is a computer program which is a sequence of instructions designed to direct a computer to perform certain task. The software enables a computer to receive input, store information, make decisions, manipulate and output data in the correct format. A program consists of instruction that tell the computer what to do and how to behave.

System software

The purpose of system software is to help run the computer system by controlling, Integrating and managing the individual hardware components of a computer system.eg. Operating system, device drivers etc. Operating systems like linux , windows and DOS (Disk operating system) control all parts of the computer system by handling I/O devices, coordinating and managing resources like memory, disk ,CPU etc. and provide a environment over which other programs(software) can run.

Application software

Software which is used for user's specific application known as application software. Application software are designed to process data and support users in an organization such as solving equations or producing bills, result processing of campuses, data processing of accounts in banks. E.g. Word, Excel, PowerPoint, Photoshop, etc.

Application software can be classified into following categories.

- **Tailored software:** These kind of software are developed for solving particular problem. eg. A software for payroll of an organization, attendance system for student, software for ticket reservation etc.
- **Packaged software:** These software that is often used together, performs similar functions, or includes similar features, and is bundled together as a set of software programs. For example, Microsoft Office is packaged software, including multiple software programs used in a home or office, such as Microsoft Excel, Microsoft Word, and Microsoft PowerPoint. Video and audio editing software may be available as packaged software as well, as they may be used together for editing music and video files used in a movie.
- **Utility software:** These are special types of application software which help us to fine tune the performance of a computer, prevent unwanted actions or perform system related tasks such as checking for virus and removing virus, system utilities which provide information about current state of the use of files, memory, users and peripherals eg disk info, check disk, debuggers for removing "bugs" from program.

Types of computers

1) On the basis of working mode

How does digital computer differ from analog computer? Define Hybrid computer.

i) Analog computer

Analog computer is one which operates on continuous data, usually of a physical nature such as length, voltage or current, etc. An analog machine is usually a special purpose device dedicated to a single task. Analog computers are based on analog signals that are continuous signals. These types of computers were widely used in scientific and industrial applications.

Presley, thermometer, speedometer, barometer, lactometer, etc are the example of an analog computer. An example of analog devices is a thermometer, barometer, speedometer and ammeter.

ii) Digital computer

The computer which works on discrete data or discontinuous data is known as a digital computer. It works on a binary system where 0 represent off and 1 represents on. It is based on digital signals i.e discrete signals.

So, the basic principle of these computers are either present or absence of an electric pulse in the signals. It is a multipurpose and programmable computer. It is fast processing, more accurate and has large memory capacity. It is usually general purpose computer.

Digital watch, digital speedometer etc are the example of digital devices Some examples of a digital computer are IBM PC, Apple/Macintosh computer, etc.

Analog computer vs Digital computer

Analog computer	Digital computer
1. Uses the continuous signal for its operation.	1. Use the discrete signal for their processes which are in the binary form (0 and 1).
2. Usually contains either no any or limited memory capacity.	2. It usually contains larger storage capacity.
3. Accuracy is less.	3. They are more reliable, accurate and less affected by noise.
4. Slower in speed.	4. Faster in speed.
5. Used for specific purpose.	5. Used for general purpose.
6. These computers use a network of resistors and capacitors.	6. Here a large number of logic gates, microprocessors and on-off switches are used.
7. Eg. Speedometer, Thermometer, Analog watch etc. are the analog devices.	7. Eg. Digital watch, Digital speedometer etc are the example of digital devices.

iii) Hybrid Computer

A computer, which has a combination best feature of both analog and digital computers is called a hybrid computer. It helps the user to process discrete and continuous data. The hybrid computer can convert the analog signal into digital signals and digital signal into analog signal. Hybrid computers are used mainly in specialized applications where both kinds of data need to be processed. Therefore, they help the user to process both continuous and discrete data.

They have usually the speed of digital computers and the accuracy of analog computers. They can perform the task of both analog and digital computer. They are usually used for special problems, as it is a special purpose computer in which input data are derived and measurement are converted into digits and processed by a computer.

It is mostly used in radar communication, rocket launching, weather forecasting and in other fields.

In weather forecasting, analog devices measure wind speed, humidity, temperature, wind direction and fed to the digital devices that compare with the past information to predict the climate changes.

2) On the basis of size and capabilities

Explain the classification of computer based on size and capabilities.

- i) **Super computers:** Supercomputers are very expensive, very fast, and the most powerful computers we have in the world. It has multiuser, multi -processing, very high efficiency and large amount of capacity. They are used in scientific research centers like NASA. These computer types are also very large in size due to the numerous parts and components involved in their design.

A good example of a Supercomputer is Tianhe-2.

- ii) **Mainframe computers:** These are large and expensive computer types capable of supporting hundreds, or even thousands, of users simultaneously. Thus, they are mostly used by governments and large organizations for bulk data processing, critical applications, transaction processing, census, industry and consumer statistics among others. They are ranked below supercomputers

Examples. IBM 4300 series, ICL 39 series, IBM-1401.

- iii) **Mini Computer**

Minicomputers are used by small businesses & firms. Minicomputers are also called as "Midrange Computers". These are small machines and can be accommodated on a disk with not as processing and data storage capabilities as super-computers & Mainframes. These computers are not designed for a single user. Individual departments of a large company or organizations use Mini-computers for specific purposes. For example, a production department can use Mini-computers for monitoring certain production process.

Popular Minicomputers

- K-202
- Texas Instrument TI-990
- SDS-92

- iv) **Micro Computer**

Desktop computers, laptops, personal digital assistant (PDA), tablets & smartphones are all types of microcomputers. The micro-computers are widely used & the fastest growing computers. These computers are the cheapest among the other three types of computers. The Micro-computers are specially designed for general usage like entertainment, education and work purposes. Well known manufacturers of Micro-computer are Dell, Apple, Samsung, Sony & Toshiba.

Block Diagram of computer

- With the help of block diagram of digital computer explain the function of control unit and memory unit.[PU:2014 spring]
- Draw a block diagram of digital computer. Explain Each Component in brief. [PU:2018 spring]
- What is the functional difference between primary memory and secondary memory? Explain the function of control unit with the help of block diagram of digital computer.[PU:2016 fall]

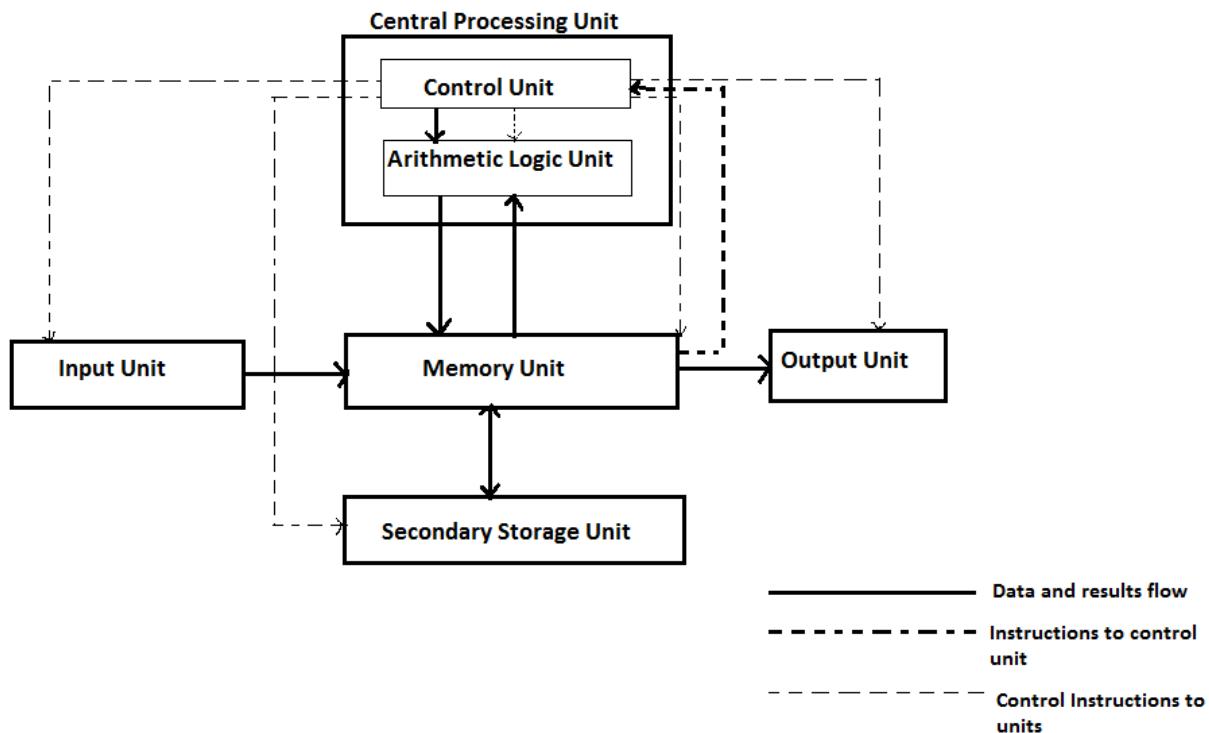


Fig. Block Diagram of Computer

Input units: They are input devices from which data and instructions entered into memory unit. The commonly used input devices are keyboard, mouse, trackball, joystick, touch panel etc.

Memory unit: It holds data and instructions that are entered through the input unit, before they processed. It preserves the intermediate and final results before they are sent to the output devices. It also saves data for later use. They are categorized as:

- Primary storage:** They stores data and programs while the program is being executed. They are temporary in nature. The data is lost when computer is switched off. They are faster, expensive and occupy less space as compared to secondary storage unit. eg RAM
- Secondary storage:** They stores the data and program permanently. The program that run on a computer are first transferred to primary memory before it actually run.

Whenever the results are saved they get stored in secondary memory. They are cheaper and slower than primary memory. eg. Hard disk, CD-ROM etc.

Processing Unit:

It is also known as central processing unit, used to perform computation and information processing on those data that is entered through input devices. Processing unit consist of Control Unit (CU) and Arithmetic Logic Unit (ALU).

- **Control Unit:** The control coordinates or controls all the activities performed in a computer system. It transfers data and instructions to the ALU for arithmetic operations. It tells the computer's memory, arithmetic and logic unit and input and output devices how to respond to the instructions that have been sent to the processor. As well, It directs the operation of the other units by providing timing and control signals.
- **Arithmetic and Logic unit:** They performs Arithmetic and logical operations on data. Arithmetic operations performed by them are Addition, Subtraction, Multiplication, Division etc. Similarly logical operations perform by them are AND, OR, NOT etc.
Whenever calculations are required, the control unit transfers the data from memory unit to ALU, once computations are done, the results are transferred to the Memory unit by the control unit and then it is send to the output unit for displaying data.

Output Unit

They are set of output devices used for providing the output of a program that is obtained after performing the operations specified in a program. Commonly used output devices are monitor, printer etc.

Programming Language

- What do you mean by programming language? Explain all types of programming language with examples.[PU:2013 spring][PU:2014 fall]
- What is programming language? Describe Low level programming language(LLL) and High level Programming language (HLL) with examples.[PU:2012 fall]
- Define programming language .Differentiate between high level and low level programming language.[PU:2013 fall][PU:2015 spring]
- What is programming Language? Why High level programming (HLL) is preferred to low level programming Language(LLL)?[PU:2016 spring]
- What do you mean by programming language? Discuss on machine language, assembly language and high level language.[PU:2017 fall]
- What is programming language? Explain about high level programming language and low level programming languge.[PU:2019 fall]
- Compare and Contrast High level programming language and Low level language.[PU:2017 spring]

A programming language is a standardized communication technique for describing instructions for a computer. Each programming language has a set of syntactic and semantic rules used to define computer programs.

A programming language enables a programmer to precisely specify what data computer is act upon, how these data are to be stored/transmitted and what actions are to be taken under circumstances.

Programming language are classified mainly in two categories:

- Low level programming language
 - High level programming language
- i) **Low level programming language**
- Low level language are specific to hardware. Before creating a program, it is required to have through knowledge of that hardware. low level programming language are divided into two types.
- Machine language
 - Assembly language

Machine language

Machine language are lowest- level programming language. A computer understands program written only in the machine language. It is directly executable by computer without the need for translation by a compiler or an assembler.

Machine code consist entirely of the 0's and 1's of the binary system. Early computers were programmed using machine language. Programs written in machine language are faster and efficient.

Writing program in machine language is very tedious, time consuming, difficult to find bugs in longer programs.

Assembly language

In assembly language, instead of using numeric opcodes (i.e. pattern of 0 & 1), mnemonics are used eg. ADD, SUB etc. Program written in assembly language must be converted into machine language which could be done by assembler. Assembly language program written for one type of CPU won't run on another. So that assembly language is time consuming and machine dependent.

ii) High Level programming language

The syntax of high level is closer to human language. High level language were developed to make programming easier. Most of the high level language are English like languages. They use familiar English words, Special symbols (! & etc.) in their syntax .Therefore high level language are easier to read, write, understand and maintain.

Each high level language has their own set of grammar and rules to represent set of instructions. Eg. C ,C++,Java, FORTAN etc.

Program written in high level language also need to translated into machine language. This can be done either by compiler or interpreter.

Language Translator

- What do you mean by language translation? Differentiate compiler and interpreter.
- Explain the significance of compiler and interpreter.

A programmer write a program in high level language that is to be translated into machine language equivalent code. This task is achieved by using language translator.

The common language translator are:

- Compiler
- Interpreter
- Assembler

Difference between compiler and interpreter

Compiler	Interpreter
1. A compiler translates the entire source code to object code and then only object code is executed.	1. An interpreter translates one statement at a time, executes it and continues for another statement.
2. Compiler is faster than interpreter.	2. Interpreter is slower than compiler.
3. It generates the error message only after scanning the whole program. Hence debugging is comparatively hard.	3. It continuously translates the program until the first error is met, in which case it stops. Hence debugging is easy.
4. A compiler is a complex program.	4. Interpreter is less complex program than compiler.
5. As compared to an interpreter developing a compiler is difficult.	5. As compared to a compiler developing interpreter is easier.
6. Programming language like C,C++,FORTAN use compiler.	6. Python use interpreter.

Assembler

An assembler is a program (software) which translates the program written in assembly language to machine language. It takes the basic commands and operations from assembly code and converts them into binary code that can be recognized by a specific type of processor.

Assemblers are similar to compilers in that they produce executable code. However, assemblers are more simplistic since they only convert low-level code (assembly language) to machine code. Since each assembly language is designed for a specific processor, assembling a program is performed using a simple one-to-one mapping from assembly code to machine code in that they produce executable code.

Differences between high level and low level programming language

High level programming language	Low level programming language
1. It is programmer friendly language.	1. It is machine friendly language.
2. Compiler or interpreter is required for translation.	2. Assembler is required for translation.
3. They execute slower.	3. They execute faster.
4. For writing program hardware knowledge is not required.	4. For writing program hardware knowledge is required.
5. They are easier to learn and understand by human.	5. It is difficult to learn and understand by human.

6. It can run on any platform.	6. It is machine dependent.
7. It is simple to debug and maintain.	7. It is difficult to debug and maintain as compared to high level language.
8. Example: C, C++, Java etc.	8. Example: Assembly language

Advantages and Disadvantages of High level and low level programming language

Advantages of High level language

- High level languages are programmer friendly.
- They are easy to write, debug and maintain.
- They are portable, which means same code can run on different hardware.
- It is machine independent language.
- Easy to learn.

Disadvantages of High level language

- It takes additional translation times to translate the source to machine code.
- Execution of high level programs are comparatively slower than low level programs.
- Compared to low level programs, they are generally less memory efficient.
- Cannot communicate directly with the hardware.

Advantages of low level languages

- Programs developed using low level languages are fast and memory efficient.
- There is no need of any compiler or interpreters to translate the source to machine code. Thus, cuts the compilation and interpretation time.
- It can directly communicate with hardware devices.

Disadvantages of low level languages

- Programs developed using low level languages are machine dependent and are not portable.
- It is difficult to develop, debug and maintain.
- Programmer must have additional knowledge of the computer architecture of particular machine, for programming in low level language.

Traditional and structured programming

Differentiate structured programming concept from traditional programming concept.

In traditional Programming we start with a problem to solve. We figure out how to break the problem into smaller parts, and then each part into smaller parts. At each stage we think about how to do things, then another, then yet another. So divide and conquer with an emphasize on doing things.

Structured programming is a subset of procedural programming that enforces a logical structure on the program being written to make it more efficient and easier to understand and modify.

Structured programming frequently employs a top down top-down design model, in which developers map out the overall program structure into separate subsections. A defined function or set of similar function is coded in a separate module or sub-module, which means that code can be loaded into memory more efficiently and that module can be reused in another programs. After a module has been tested individually, it is then integrated with other modules into the overall program structure.

Pokhara University Previous Board Exam Questions

1) Why higher generations of computer are better?

When generations goes up the following characteristics are in effect.

Characteristics	Effects
Size	Decrease
Speed	Increase
Power consumption	Decrease
Processing capability	Increase
Storage capacity	Increase
Reliability and accuracy	Increase

Due to this reason ,higher generations computer are better.

2) Discuss briefly about the history of computing and computers. [Assignment]

3) Why C is called Structured Programming language?[PU:2017 spring]

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 function

4) What do you mean by instructions and software?[PU:2019 spring]

An instruction is an order given to a computer processor by a computer program. At the lowest level, each instruction is a sequence of 0s and 1s that describes a physical operation the computer is to perform (such as add decimal, move, load register)

Software is a computer program which is a sequence of instructions designed to direct a computer to perform certain task. The software enables a computer to receive input, store information, make decisions, manipulate and output data in the correct format. A program consists of “instructions” that tell the computer what to do and how to behave.

CHAPTER 2

Programming Logic

What is a problem?

- Problem is defined as the difference between an existing situation and desired situation, that is, in accordance with calculation; problem is numerical situation and has complex form. Solution is a desired situation and has simplest form.
- If a problem is solved by computing using machine called computer, then such process is called problem solving using computer.

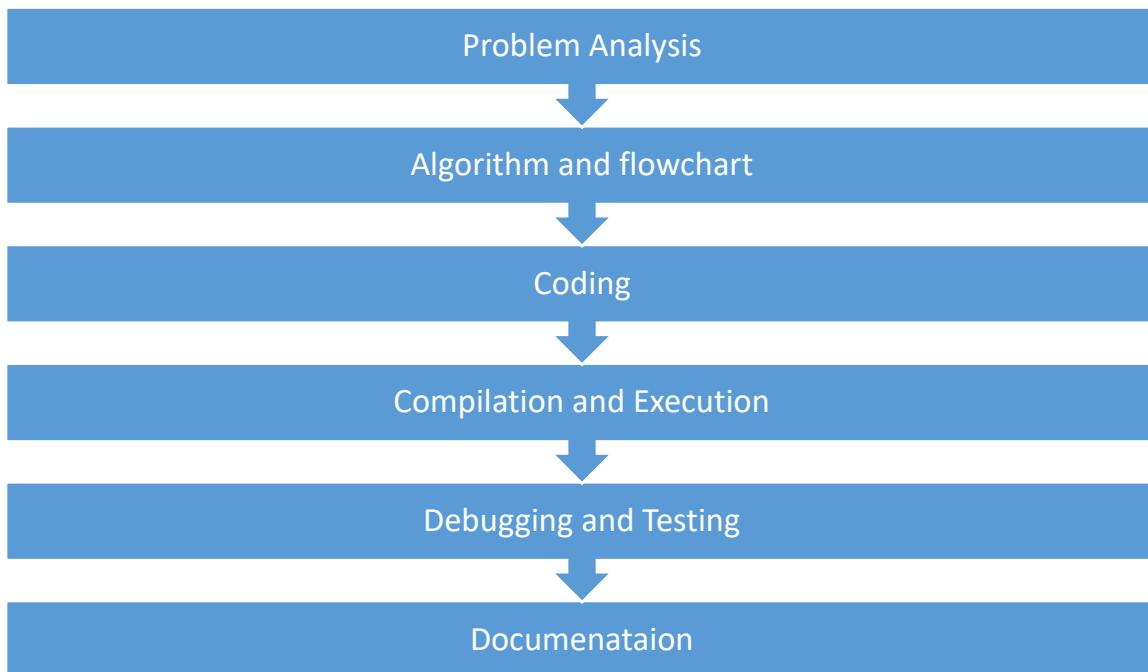


Fig. Steps in problem solving by a computer

- 1) **Problem Analysis:** It is important to give a clear and precise problem Statement .It is also called Problem definition. In this step, we should specify the following things.
 - i. **Objectives:** The problem should be stated clearly.
 - ii. **Input requirements:** To get desired output it is required to define the input data and source of input data.e.g. In student information system, input data may be students records and source may be college administration.
 - iii. **Output Requirements:** We must know what exactly we are expecting out of the system.

- iv. **Processing Requirements:** It is required to clearly defined processing requirements to convert the given input data to the required output. In processing requirements, there may be hardware platform, software platform, manpower etc.
- v. **Evaluating feasibility:** It is one of the most important phases, where we mainly decide whether the purposed software development task is technically and economically feasible.

2) Algorithm and flowchart

An Algorithm is defined as a sequence of steps or procedures necessary to solve problem. To be an algorithm set of step must be unambiguous. Each step tells what task to be performed.

An Excellent Algorithm should have the following properties.

- i. **Finiteness:** Each algorithm must have finite number of steps to solve a problem.
- ii. **Definiteness:** The action of each step should be defined clearly without any ambiguity.
- iii. **Inputs:** Inputs of the algorithm should be define precisely, which can be given initially or while the algorithm runs.
- iv. **Output:** An algorithm can give one or more result. After an algorithm terminates, algorithm must give desired result.
- v. **Effectiveness:** An algorithm should be more effective among different ways of solving the problem.

EXAMPLES

Algorithm to find the sum of any two numbers

- Step 1: Start
- Step 2: Declare variables a ,b and sum
- Step 3: Read value of a and b
- Step 4: calculate sum=a+b
- Step 5: Display sum
- Step 6: Stop

Algorithm for calculating the simple interest using the formula $i=p*t*r/100$

- Step 1: Start
- Step 2: Declare variables p, t, r and i
- Step 3: Read value of p, t, r
- Step 4: $i=p*t*r/100$
- Step 5: Display i
- Step 6: Stop

An algorithm to convert Fahrenheit when temp in Centigrade is given.

Step 1: Start
Step 2: Declare variables f and c
Step 3: Read value of c
Step 4: calculate $f=1.8*c+32$
Step 5: Display f
Step 6: Stop

An algorithm to calculate the area of rectangle

Step 1: Start
Step 2: Declare variables l, b and area
Step 3: Read value of l and b
Step 4: calculate area= $l*b$
Step 5: Display area
Step 6: Stop

Advantages of Algorithms:

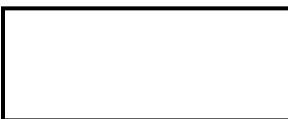
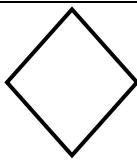
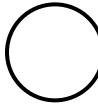
- It is a step-wise representation of a solution to a given problem, which makes it easy to understand.
- An algorithm uses a definite procedure.
- It is not dependent on any programming language, so it is easy to understand for anyone even without programming knowledge.
- Every step in an algorithm has its own logical sequence so it is easy to debug.
- By using algorithm, the problem is broken down into smaller pieces or steps hence, it is easier for programmer to convert it into an actual program.

Disadvantages of Algorithms:

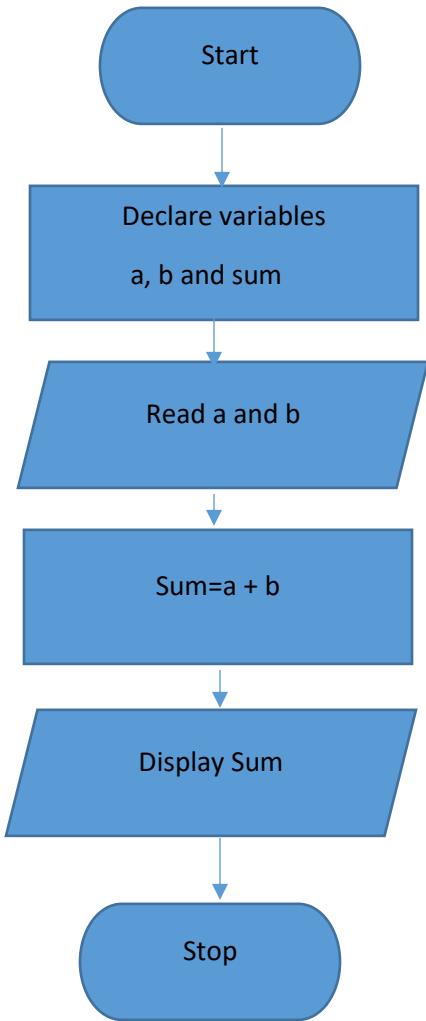
- Algorithm is Time consuming.
- Difficult to show Branching and Looping in Algorithms.
- Big tasks are difficult to put in Algorithms.

Flowchart:

A flowchart is an pictorial representation of an algorithm. It uses boxes of different shapes to denote different types of instructions. The actual instructions are written within these boxes use clear and concise statements. These boxes are connected by solid lines having arrow marks to indicate the flow of operation, that is exact sequence in which instructions are to be executed.

Symbol	Purpose	Description
	Flow Line	Used to connect symbols to indicate the flow of logic
	Terminal (Start/Stop)	Used to indicate the start and end of flowchart. One flow line exists from Start and one enter to Stop
	Processing	Uses for arithmetic and data manipulation operations
	Input/ Output	Used whenever information is entered into the flowchart or displayed from the flowchart.
	Decision	Used to represent operation in which two possible alternatives ie. Decision making and branching
	On-page Connectors	Used to connect remote flowchart portions of the same page.

Flowchart to add two numbers



Advantages of using flowcharts

- i. **Communication:** Flowcharts are better way of communicating the logic of the system to all concerned.
- ii. **Effective analysis:** With the help of flowchart, problem can be analyzed in more efficient way.
- iii. **Proper Documentation:** Program flowcharts serve as a good program documentation, which is needed for various purposes
- iv. **Efficient coding :** The flowcharts act as a guide or blueprint during the system analysis and program development phase
- v. **Proper Debugging:** The flowchart helps in debugging process.
- vi. **Efficient program maintenance:** Maintenance of operating program becomes easy with the help of flowchart.

Limitations of using flowchart

- i. **Complex logic:** A flowchart becomes complex and clumsy when the program logic is quite complicated.
- ii. **Difficulty in alternations and modifications:** If alternations are required, the flowchart may need to be drawn completely.

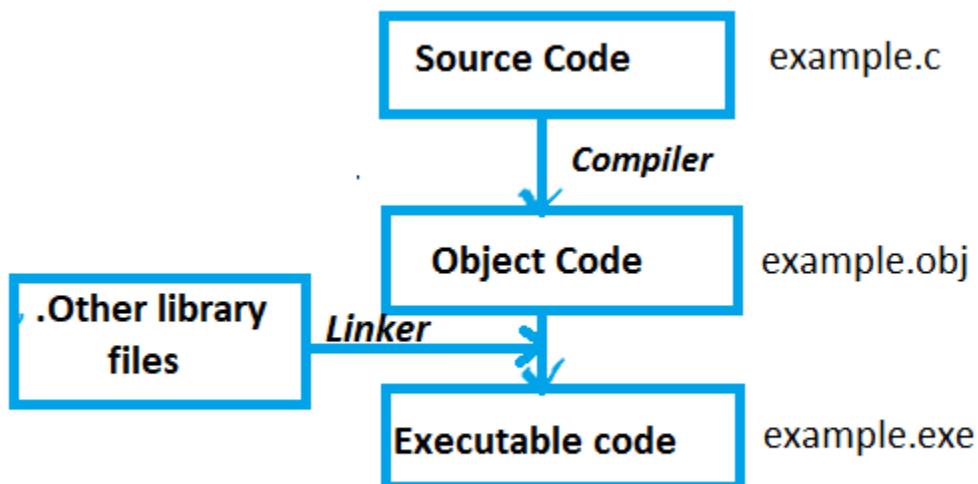
3) Coding

The coding is a process of transforming program logic or ideas into computer language format. This stage translates program design into computer instructions using some programming language, such as c, c++ etc. so, we should properly follow the rule and syntax while coding. The code writing using programming language is also known as source code.

4) Compilation and execution

Program written in high level language must be converted into low level language because computer only understand low level language. This process is known as compilation. So, compiler does job of translating high level language into low level language. Compilation process tests the program whether it contains error or not. If syntax errors are present, compiler cannot compile the code until it is corrected.

Once, compilation is completed then the program is linked with another library files needed for execution. Thereby resulting in binary program and then program is loaded into memory for the purpose of execution. During execution, program may ask user for inputs and generates output after processing inputs.



5) Debugging and testing

Debugging

Debugging is the process of isolating and correcting any type of errors. Different type of debugging techniques are as follows.

- Error Isolation
- Tracing
- Watch Values
- Break points
- Stepping

Testing

Testing ensures the program performs correctly the required task. Testing is very important phase before delivering software to customer. Testing is usually performed for the following purposes.

- To improve quality
- For verification and validation
- For reliability estimation

Testing process may include the following two stages.

i) **Human testing**

- Done before computer based testing begins
- Test is carried out statement by statement by a programmer/test group and analyzed with respect to checklist of common programming errors.

ii) **Computer based testing**

Computer based testing involves two stages namely compiler testing and runtime testing.

Compiler testing

- Debugging techniques used to find syntax error.

Runtime testing

- If there is no syntax error, we need to find logical and runtime error. After removing these errors, it is required to run program with test data to check whether program is producing correct result or not.

Types of Errors:

Generally errors are classified into following types:

Syntax Error: Any violation of rules of the language results in syntax error. Most frequent syntax error are:

- Missing parenthesis
- Missing semicolons.
- Printing the value of variable without declaring it.

Semantic Error:

This error occurs when the statements written in the program are not meaningful to the compilers.

eg. $a+b=c$

Runtime Error:

Error which occur during program execution after successful compilation are called runtime error.one of the most common run-time error is

- Division by zero
- Null pointer assignment
- Trying to open file that was not created

Logical Error

Logical errors are the errors in the output of the program. The presence of logical errors to undesired or incorrect output and are caused due to error in logic applied in the program to produce desired output.

Latent Error:

Latent Errors are the hidden errors that occur only when a particular set of data is used.
Consider below example.

`result=(a+b)/(c-d);`

An error occurs only when c and d are equal because that make remainder zero (divide by zero error).Such error can be detected only by using all possible combinations of data.

6) Documentation

Program Documentation is the description of program and its logic written to support understanding of program. It keeps information of all phases described above while developing project.

- Documentation of program helps to those who use, maintain and extend the program in future.
- Properly documented program is necessary which will be useful and efficient in debugging, testing, maintenance and redesign process.

There are two types of documentation

1) Programmer documentation(Technical documentation)

Programmer's Documentation is prepared for future reference to the programmers who maintain, redesign, and upgrade the system.

It may contain

- Detail requirements of program
- Dataflow diagram, E-R diagram and flowchart
- Selection of meaningful variables name and use of comments
- Parameter and definition list to explain the meaning and purpose of each parameter and definition of each function.
- Algorithm and flowchart of each component
- Input and output data used for testing and other references

2) User documentation

- Provides the support to user who use the software.
- It provides guidelines for installation of program and use it efficiently.

Implementation

After a program has been written and tested it need to be implemented to target environment to solve problem. Various needs of physical hardware and accessories required by the program to solve the intended problem needs to be present upon implementation.

Evaluation and Maintenance

The Evaluation of the performance of program is need to be done at frequent interval once the program software is implemented. Evaluation can be done by various parameters such as quality assurance, user friendliness, flexibility and functionality.

Maintenance is the process of modifying a software for the following reason.

- To remove the a bug from the program
- To improve the efficiency of code
- Add new function to fulfill new user requirements.
- To improve user interface.

Previous Board Exam Questions of Pokhara University

- 1) Define Algorithm and flowchart. Draw a flowchart to read 3 numbers from the user and find the smallest one.**[PU:2012 fall]**
- 2) What do you mean by Algorithm and Flowchart? Explain the C compilation process in brief.**[PU:2013 spring]**
- 3) What is the importance of documentation in programming? Write an algorithm and draw a flowchart to find and output all the roots of a quadratic equation, for non zero coefficients .In case of errors program should report suitable error message.**[PU:2014 spring]**
- 4) Discuss the significance of Algorithm and Flowchart in programming. Draw a flowchart for finding greatest digit for the supplied number by user.**[PU:2014 fall]**
- 5) Write the significance of algorithm and flowchart in programming. Draw a neat flowchart to input a number and check it is prime number or not.**[PU:2015 spring]**
- 6) What is flowchart ?Write an algorithm and draw a flowchart to display whether the given number is prime or not.**[PU:2015 fall]**
- 7) What is the significance of algorithm and flowchart in programming? Write an algorithm and draw a neat flowchart to input a number and check it is palindrome number or not.[Note: Palindrome number remains same even after reverse such as 989]
[PU:2016 spring]
- 8) What do you mean by algorithm and flowchart? Write an algorithm and flowchart to find palindrome of a given number.**[PU:2017 spring]**
- 9) Define a role of flowchart in efficient program maintenance with its character. Also develop a flowchart to print the even numbers between 150 to 500.**[PU:2016 fall]**
- 10) Define algorithm and flowchart. Draw a flowchart to read three numbers from the user and find the smallest one.**[PU:2017 fall]**
- 11) Define the role of flowchart in efficient program maintenance with its character. Also develop a flowchart to print the Armstrong numbers between 150 to 500.**[PU:2018 spring]**

- 12) Write an algorithm and draw a flowchart to generate the Fibonacci sequence of eight terms.[PU:2018 fall]
- 13) What is the role of algorithm in programming? Write down an algorithm to check if the given number is prime or not.[PU:2019 fall]
- 14) Differentiate between algorithm and flowchart. Write an algorithm and flowchart to find the number given by user is divisible by 2 ,3 or 6 or not.[PU:2019 spring]
- 15) **Write the short notes on:**
- Debugging and testing[PU:2016 spring][PU:2018 fall]
 - Compilation and Execution process
 - Software development life cycle(SDLC)[PU:2017 spring,PU:2018 fall]**
 - Types of Error in programming
 - Importance of Documentation in programming
 - Compile time error and Runtime error
 - Documentation[PU:2018 spring]
- 16) Define coding, compilation, debugging, execution, testing and Implementation and maintenance.
- 17) What are the steps involved in problem solving by using computer.

CHAPTER 3

Variable and data types

Character set of C

Character set is a package of valid characters recognized by compiler/interpreter.

C uses character as building blocks to form basic program elements.(Eg. Constants, variables, expressions etc.

The characters available in C are categorized into following types.

1) Letters/Alphabets

- Uppercase(A-Z)
- Lowercase(a-z)

2) Digits (0-9)

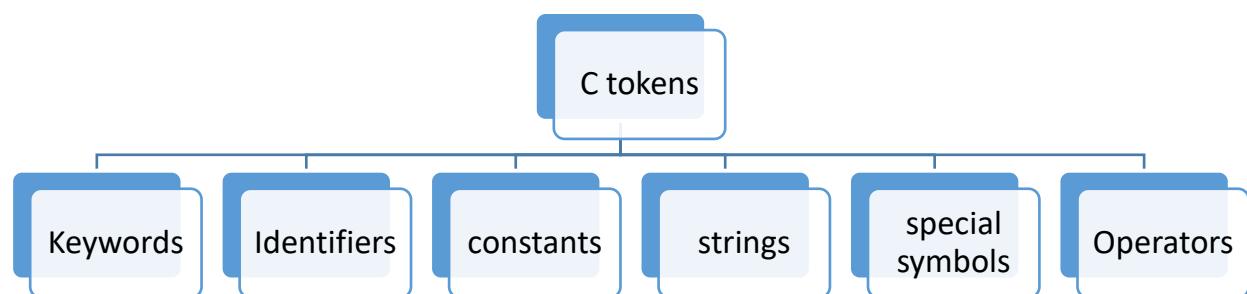
3) Special symbols (+,-,* , ! ,%,_,<,>, :)

4) Whitespace characters

- Blank Space
- Newline
- Horizontal tab
- Vertical tab

Tokens

Tokens are the smallest individual units of program. C has six types of tokens .They are as follows:



Keywords

Keywords are predefined words whose meaning has already been defined by compilers. These keywords are used for pre-defined purposes and cannot be used as identifiers.

The standard keywords are:

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

Identifiers

Identifiers are the names given by user to various program elements such as variables, functions and arrays.

What are the rules for naming identifier? [PU: 2014 spring]

Rules of naming Identifiers

- First character must be alphabet (or underscore).
- Must consist of only letters, digits or underscore
- Keyword cannot be used.
- The length of identifiers should not be greater than 31 characters.
- Must not contain whitespace.

Data type

Important Questions from this topic:

- What are the different data types available in C? Explain their types qualifier, conversion character, range of value and storage size in memory occupied by each type. [PU:2018 fall]
- Mention the appropriate data type for storing following data. Also justify your answer in brief. [PU:2013 fall]
 - Distance jumped by frog
 - A prime number between 5 and 555
 - Weight of your body
 - The examination symbol number for student
- How can you declare following variables using suitable data types? Mobile phone numbers, address, body, temperature, salary. Also explain each memory occupancy size and range. [PU:2018 spring]
- Why we need different data types in programming? [PU:2017 spring]
- Why it is necessary to have knowledge of data type in C programming. Explain all types of data types of data type available in C. [PU: 2017 spring]
- Describe the four basic data-types along with their size and range. [PU:2019 fall]

Definition:

Data type is a classification of type of data that a variable can hold in programming. The data type specifies the range of values that can be used and size of memory needed for that value. Generally data type can be categorized as:

1) Primary data type

The size and range of data types on 16 bit machine are:

Data type	Description	Required Memory	Range	Format specifier
char	Used to store character value Eg. 'a', 'c', 'x'	1 byte	-128 to 127	%c
int	Used to store whole numbers/non fractional numbers. Eg. 101, 205, -908	2 bytes	-32768 to +32768	%d
float	Used to store fractional number and has precision of 6 digits. Eg. 5.674, 304.67, 67.8903	4 bytes	3.4×10^{-38} to 3.4×10^{38}	%f
double	Used when precision of float is insufficient and it has precision of 14 digits.	8 bytes	1.7×10^{-308} to 1.7×10^{308}	%lf
void	Has no values and used as return type for function that do not return a value. Eg. void main()			

2) Derived data type

They are derived from existing primary data types.

Eg. Array, Union, Structure

3) User defined data type

The data type that are defined by user is known as user defined data types. Examples **enum** (Enumerated data type) and **typedef** (type definition datatype)

General form of typedef

typedef existing_data_type new_name_for_existing_data_type ;

fundamental data type

new identifier

Example

typedef int integer ;

integer symbolizes int data type Now, we can declare int variable "a" as **integer a** rather than **int a** .

Variable

Variable is defined as a meaningful name given to the data storage location in computer memory. It is a medium to store and manipulate data. The value of variable can change during execution of program.

Variable declaration

Naming a variable along with its data type for programming is known as variable declaration.

The declaration deals with two things:

- It tells the compiler what the variable name is
- It specifies what type of data the variable can hold

Syntax:

data_type variable_name ;

eg. int roll;

roll is a variable of type integer. roll can only store integer variable only.

- Explain the necessary rules to define the variable name in C programming.[PU:2012 fall]
- Which of the following are invalid variable name and why?[PU:2019 fall]

Minimum	First.name	Row total	&name
Doubles	3 rd _row	column_total	float

Rules for declaring variables

Variable name must begin with a letter, some system permit underscore as first character.

- ANSI standard recognizes a length of 31 characters.
- It should not be keyword.
- Whitespace is not allowed.
- Variable name are case sensitive .ie. uppercase and lowercase are different. Thus variable temp is not same as TEMP.
- No two variables of the same name are allowed to be declared in the same scope.

Variable initialization

Initializing variable means specifying an initial value to assign it.

1) Compile time initialization

In compile time initialization value is assigned to variable using assignment operator '='.

Eg: int a; //variable declaration
a=10; //variable initialization

It is also possible to assign a value at the time of declaration of variable.

Eg. int a=10,b=5;
Float x=10.5;

2) Runtime initialization

In runtime initialization the value is assigned by using scanf() function.

Syntax:

scanf("format specifier",list of address of variable);

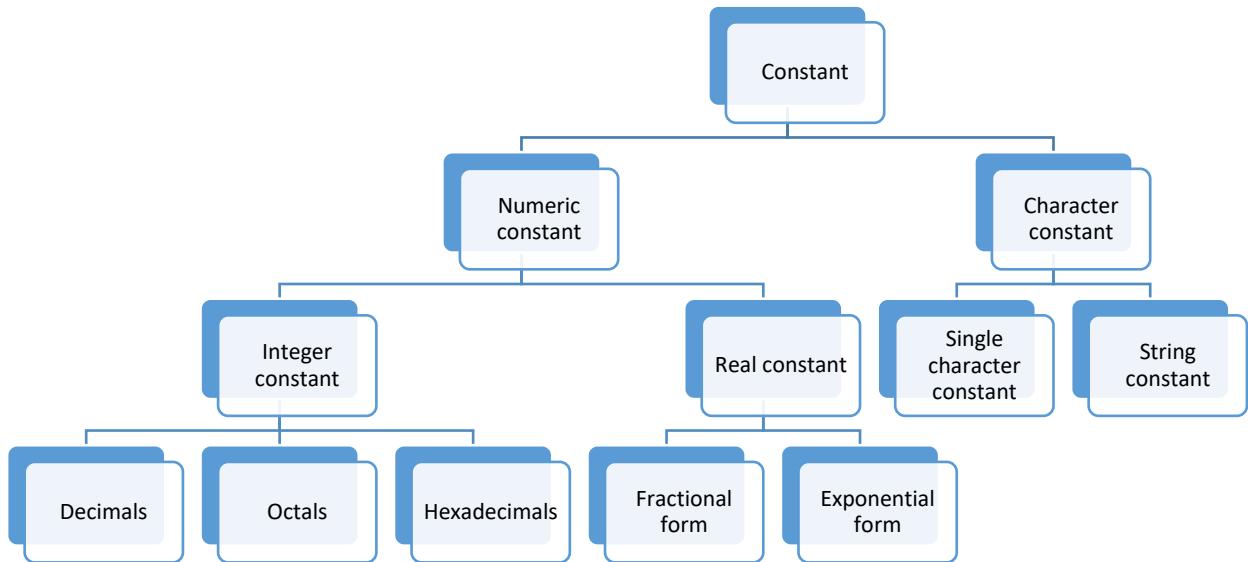
eg. scanf("%d%d",&a,&b);

Program to illustrate runtime initialization:

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int a,b;
    printf("Enter two numbers\n");
    scanf("%d%d",&a,&b);
    printf("Entered numbers are %d and %d",a,b);
    getch();
    return 0;
}
```

Constants

Constant refers to fixed values that do not change during the execution of program. These fixed values are also called literals. The C constants are as shown in figure. Number associated constant are numeric constant and character associated constant are character constants.



1) Numeric Constant

It consist of:

1.1 Integer constant

It refers to sequence of digits.

There are three types of integer constants.

Decimals: Decimal Integer consist of set of digits, 0 through 9, preceded by an optional +ve or -ve sign. eg. 124, -321,678, 654343 etc.

Octal: An octal Integer constant consist of any combination of digits from the set 0 to 7, with a leading 0. eg 037, 075, 0664, 0551

Hexadecimal: Hexadecimal are sequence of digits 0-9 and alphabet A-F with preceding 0x or 0X. eg. 0x5567, 0X53A, 0xFF etc.

1.2. Real or floating point constant

They are numeric constant with decimal points. The real constant are further categorized as

Fractional real constant

They are the set of digits from 0 to 9 with decimal points. eg. 394.7867, -0.78676

Exponential real constant

In the exponential form the constants are represented in following form:

mantissa e exponent

where, the mantissa is either a real number expressed in decimal notation or an integer but exponent is always in integer form. eg.

$21565.32 = 2.156532E4$, Where $E4=10^4$

Similarly, -0.000000368 is equivalent to -3.68E-7

2) Character Constant

Single Character Constant

A single character constant (or simply character constant) contains a single character enclosed within a pair of single quote marks. Eg.

'5' 'X' 'A' '4' etc.

String constant

A string constant is a sequence of characters enclosed in double quotes. The characters may be numbers, special characters and blank space.eg. "Hello" "green" "305" etc.

Statements

A statement is a command given to the computer that instructs the computer to take a specific action, such as display to the screen, or collect input. A computer program is made up of a series of statements. Such as:

- 1) Selection statements (if statements)
- 2) Iteration Statements (for ,while, etc.)
- 3) Jump statements (goto etc.)

Preprocessor directive

Preprocessor directives is a collection of special statement that are executed at the beginning of compilation process. They are placed in a source program before the main function and begins with #(Hash).

Some examples of preprocessor directives are

```
#include<stdio.h>      //used for standard input and output  
#define PI 3.1416        //used for defining symbolic constant PI as 3.1416  
#define TRUE 1           //used for defining True as 1
```

We use preprocessor directive for

- File inclusion
- Conditional compilation
- Macro expansion

Type casting

Converting an expression of a given type into another type is known as type casting.

Here, it is best practice to convert lower data type to higher data type to avoid data loss.

1. Implicit Conversion

Implicit conversions do not require any operator for conversion. They are automatically performed when a value is copied to a compatible data type in the program.

Here, the value of **i** has been promoted from **int** to **float** and we have not had to specify any type-casting operator

Program

```
#include<stdio.h>
int main()
{
    int i=20;
    float result;
    result=i; // implicit conversion
    printf("value=%f", result);
    return 0;
}
```

Output

```
Result=20.000000
```

2) Explicit conversion

In this type of conversion one data type is converted forcefully to another data type by the user.

The general rule for cast is

(type-name) expression

Where type-name is one of the standard c data types. The expression may be constant , variable or an expression.

Example	Action
x=(int)7.5	7.5 is converted to integer by truncation
a=(int)21.3/(int)4.5	Evaluated as 21/4 and result would be 5
b=(double)sum/n	Division is done in floating point mode
z=(int)a+b	a is converted into integer and then added to b

Program

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int a=5,b=2;
    float result;
    result=(float)a/b; // Explicit conversion
    printf("value=%f",result);
    getch();
    return 0;
}
```

Output

Result=2.50000

Escape sequences

Write a short notes on: Escape sequences [PU: 2016 fall]

Escape sequences are non-printable characters used for formatting the output only. It is a character combination of backslash (\) followed by a letter or by combination of digits. Each sequences are typically used to specify actions such as carriage return, backspace, line feed or move cursors to next line.

The commonly used Escape sequences are as follows

Escape Sequence	Purpose
\a	Alert sound .A beep is generated by a computer on execution.
\b	Backspace
\n	Newline
\r	carriage return
\t	Horizontal tab
\v	Vertical tab
\\"	Backslash
\"	Display double quote character
\'	Display single quote character
\0	Null character. Marks the end of string

Symbolic constant

Symbolic constant is simply a identifier used in place of constants. They are usually defined at the start of the program. When the program is compiled each occurrence of symbolic constant is replaced by corresponding character sequence. Preprocessor directive like #define allow an identifier to have constant value throughout the program.

The syntax for defining symbolic constant is

```
#define symbolic_constant_name value
```

eg .#define PI 3.14159
preprocessor symbolic name Constant
directive

Advantages:

- Modifiability
- Understandability

Program to illustrate use of symbolic constant

```
#include<stdio.h>
#include<conio.h>
#define PI 3.1416
int main()
{
    float r,area;
    printf("\nEnter the value of r\n");
    scanf("%f",&r);
    area=PI*r*r;
    printf("Area=%f",area);
    getch();
    return 0;
}
```

Delimiters

A Delimiter is a unique character or series of character that indicates the beginning or end of a specific statement, string or function body set.

Some examples of delimiters are

1. **Comma , :**It is used to separate two or more variables ,constants
2. **Semicolon ; :**It is used to indicate the end of statement
3. **Apostrophes ' :**It is used to indicate the character constant
4. **Double quotes “ ” :**It is used to indicate the end of string.
5. **White space:** Space, tab, newline etc.
6. **Curly brackets {}**
7. **Round brackets or parentheses ()**

Pseudo code

Write a short notes on: Pseudo code [PU: 2015 spring]

The code which is exactly not the real code in a particular programming language but looks like a programming language instructions are known as pseudo code. Pseudo code are phrases written ordinary natural language .The pseudo code is not recognized by computer because they are not written in programming language.

Example:

If student's grade is greater than or equal to 60

```
Print "passed"  
else  
    Print "failed"  
endif
```

Advantages:

- A beginner can develop the logic to used to solve particular problem without knowing the syntax of computer language.
- Easy to understand and modify.
- Can be easily converted into program.
- A non-technical professional can easily understand

Disadvantages:

- Not recognized by computer
- No any fix and standard rules to write pseudocode.so one can misunderstand the pseudo-code written by other programmer.

ASCII

ASCII is the American Standard Code for Information Interchange. It defines an encoding standard for the set of characters and unique code for each character in the ASCII chart.

A character variable holds ASCII value (an integer number between 0 and 127) rather than that character itself in c programming. That value is known as ASCII value.

for example ASCII value of 'A' is 65

what this means is that, if you assign 'A' to a character variable,65 is stored in that variable rather than 'A' itself.

Characters	ASCII values
A-Z	65-90
a-z	97-122
0-9	48-57

Program to print ASCII value of the given character

```
#include<stdio.h>
#include<conio.h>
int main()
{
char ch;
printf("Enter the character\n");
scanf("%c",&ch);
printf("ASCII value of %c is %d",ch,ch);
getch();
return 0;
}
```

Input/ Output functions

C programming language provides many built-in functions to read any given input and to display data on screen when there is need to output the result. The header file for input/output functions is <stdio.h>

The input/output function is classified into two types.

- 1) Formatted I/O functions
- 2) Unformatted I/O functions

1) Formatted I/O functions

Formatted Input

Formatted Input refers to an input data that can be arranged in a particular format according to user requirements. The built-in function scanf() can be used to input data into the computer from standard Input device.

The general form of scanf is

```
scanf("control string",arg1,arg2 ....argn);
```

- The control string refers to field in which data is to be entered.
- The arguments arg1,arg2 ,...argn specify the address of locations where data is stored.

eg. `scanf("%d%d",&num1,&num2);`

Formatted output

Formatted output functions are used to display data in a particular specified format.

The printf() is an example of formatted output function.The general form of printf() statement is **`printf("control string",arg1,arg2 argn);`**

The control string may consist of the following data items.

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 and \b

The arguments arg1,arg2,.....argn are the variables whose value are formatted and printed according to specifications of the control string.

Eg. printf("First number=%d\tSecond number=%d",num1,num2);

Program to read multiple values of different data types using single scanf() function.

```
#include<stdio.h>
#include<conio.h>
int main()
{
    char name[20];
    int age;
    char gender;
    float weight;
    printf("Enter your Name,Age,Gender and Weight\n");
    scanf("%s %d %c %f",name,&age,&gender,&weight);
    printf("Your Name=%s\n",name);
    printf("Your Age=%d\n",age);
    printf("Your Gender=%c\n",gender);
    printf("Your Weight=%0.2f",weight);
    getch();
    return 0;
}
```

Format specifier

Format specifier is used during input and output. It is the way to tell the compiler what type of data is in variable during taking input using scanf() or printing using printf(). In format specifier the percentage(%) is followed by conversion character. This indicates the corresponding data item.

Example:

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

Here %d is format specifier and it tells the compiler that we want to print an integer value that is present in variable a.

In this way there are several format specifiers in c. Like %c for character, %f for float, etc.

List of different format specifiers

Format specifier	Purpose
%c	Character
%d	Signed Integer
%f	Floating point
%u	Unsigned Integer
%o	Octal
%X or %x	Hexadecimal representation of unsigned integer
%e or %E	Scientific Notation of float values
%s	String
%lf	Floating point (double)
%Lf	Floating point (long double)
%%	Prints % character
%n	Prints nothing

2) Unformatted I/O Functions

Unformatted I/O function do not allow the user to read or display data in a desired format. These type of library functions basically deal with a single character or string of characters. The functions `getchar()`, `putchar()`, `gets()`, `puts()`, `getch()`, `getche()`, `putch()` are considered as unformatted functions.

i) `getchar()` and `putchar()`

The `getchar()` reads a character from a standard input device. It buffers the input until the 'ENTER' key is pressed and then assigns this character to character variable.

Syntax is :

`character_variable=getchar();`

where `character_variable` refers to some previously declared character variable.

Eg.

`char c;`

`c=getchar();`

`putchar()` function displays a character to standard output device.

Syntax is: `putchar(character_variable);`

Program

```
#include<stdio.h>
#include<conio.h>
int main()
{
    char ch;
    printf("Enter the character");
    ch=getchar();
    printf("Entered characer is :");
    putchar(ch);
    getch();
    return 0;
}
```

ii) getch(), getche() and putch()

The function getch() and getche() both reads a single character in a instant. It is typed without pressing the ENTER key. The difference between them is that getch() reads a character typed without echoing it on a screen, while getche() reads the character and echoes (displays) it onto the screen.

Syntax:

```
character_variable=getch();
characher_varibale=getche();
```

The function putch() prints a character onto the screen.

Syntax:

```
putch(character_variable);
```

Program

```
#include<stdio.h>
#include<conio.h>
int main()
{
    char ch1,ch2;
    printf("Enter First character:");
    ch1=getch();
    printf("\nEnter Second character:");
    ch2=getche();
    printf("\n Firstcharacter is:");
    putch(ch1);
    printf("\nSecond character is:");
    putch(ch2);
    getch();
    return 0;
}
```

Output

```
Enter First character:  
Enter Second character: b  
First character is: a  
Second character is: b
```

At first input getch() function read the character input from the user but that the entered character was not displayed. But in second input getche() function read the a character from the user input and entered character was echoed to the user without pressing any key.

iii) gets() and puts()

The gets() function is used to read a string of text containing whitespaces, until newline character is encountered. It can be used as an alternative function for reading strings. Unlike scanf() functions, it does not skip whitespaces(ie.It can be used to read multiwords string)

Syntax: gets(string_variable);

The puts() function is used to display the string on the screen.

Syntax: puts(string_variable);

Program

```
#include<stdio.h>  
#include<conio.h>  
int main()  
{  
    char name[20];  
    printf("Enter your name:\n");  
    gets(name);  
    printf("Your name is:");  
    puts(name);  
    getch();  
    return 0;  
}
```

OPERATOR

Important questions from this topic

1. Define operator in C. List out the different types of operators used in C. Explain three of them with example.[PU:2015 fall]
2. Define operator and operand. List the types of operators and explain any five of them. [PU: 2019 spring]
3. What is an operator ?Explain the arithmetic ,logical, relational and assignment operators in C language.[PU:2016 fall]
4. What is an operator? Explain conditional operator with suitable example.[PU:2017 fall]
5. Describe about the unary operator, binary operator and ternary operator with example. [PU:2016 spring]
6. Write a short notes on:
 - Unary operations
 - Binary and unary operators [PU:2015 fall][PU:2014 spring]

Operator

- Operator is a symbol that tells the computer to perform certain mathematical or logical manipulations.
- Operators are used in program to manipulate data and variables.

Operand

The data items on which operators are act upon are called operands.

eg. $a + b$

- Here, symbol $+$ is known as operator
- a and b are operands

Expression

The combination of variables, constants and operators written according to syntax of programming language is known as Expression.

eg. $a+b*c-5$

Types of operators

Types of Operators are classified as:

1. On the basis of no of operands
2. On the basis of function/utility

1. On the basis of number of operands

i) Unary operator

The operator which operates on single operand known as unary operator.

- (++) increment Operator
- (--) Decrement operator
- + Unary plus
- Unary minus
- eg . ++x, -5, +8 , y- - etc.

ii) Binary operator

The operator which operates on two operands are known as binary operator.

eg. +(addition), -(subtraction) , /(division) , * (multiplication) , <(less than) >(greater than)
etc.

eg 3+3, 5>2 , 6*7 , 15-4

iii) Ternary operators

The Operators that operates on three operands are known as ternary operator.

Ternary operator pair “?:” is available in c .

Syntax:

expression1? expression2:expression3

The expression1 is evaluated first,

- if it is true then expression2 is evaluated and its value becomes value of the expression
- If it is false expression3 is evaluated and its value becomes value of the expression.

Eg .Let us consider the program statement

```
int a=15;b=10,max;
```

```
max = (a>b)?a: b;
```

Here, The value of a will assigned to max.

2. On the basics of utility /functions

1) Arithmetic Operators

They are used to perform Arithmetic/Mathematical operations on operands.

Operator	Meaning	Example
		If $a=20$ and $b=10$
+	Addition	$a+b=30$
-	Subtraction	$a-b=10$
*	Multiplication	$a*b=200$
/	Division	$a/b=2$
%	Modulo Division	$a\%b=0$

2) Relational Operators

These are used to compare two quantities and depending on their relation take certain decision. The value of relational operator is either 1(if the condition is true) and 0 (if the condition is false).

Operator	Meaning	Example
		If $a=20$ and $b=10$
==	Equal to	$(a==b)$ is not True
!=	Not Equal to	$(a!=b)$ is True
>	Greater than	$(a>b)$ is True
<	Less than	$(a<b)$ is not True
>=	Greater than or Equal to	$(a>=b)$ is True
<=	Less than or Equal to	$(a<=b)$ is not True

3) Logical Operators

Logical Operators are used to compare or evaluate logical and relational expressions and returns either 0 or 1 depending upon whether expressions results true or false.

Operator	Meaning	Example
		If $a=20$ and $b=10$
$\&\&$	Logical AND	Expression $((a==b)\&\&(a>15))$ equals to 0
$ $	Logical OR	Expression $((a==b) (a>15))$ equals to 1
!	Logical NOT	Expression $!(a==b)$ equals to 1

4) Assignment Operators

Assignment Operators are used to assign the result of an expression to a variable. The mostly used assignment operator is “=”.

Operator	Name	Example
=	Assignment	$c=a+b$ will assign value of $a+b$ into c
$+=$	Add AND assignment	$c+=a$ is equivalent to $c=c+a$
$-=$	Subtract AND assignment	$c-=a$ is equivalent to $c=c-a$
$*=$	Multiply AND assignment	$c*=a$ is equivalent to $c=c*a$
$/=$	Divide AND assignment	$c/=a$ is equivalent to $c=c/a$
$\%=$	Modulus AND assignment	$c\%=a$ is equivalent to $c=c\%a$

5) Increment and Decrement Operator

The increment operators (++) causes its operand to be increased by 1 whereas decrement operator (--) causes its operand to be decreased by 1.

They are unary operators (ie. They operate on single operand). It can be written as

- $x++$ or $++x$ (post and pre increment)
- $x--$ or $--x$ (post and pre decrement)

Note: The increment and decrement operators can be used as postfix and prefix notations.

Prefix notation: The variable is incremented (or decremented) first and then expression is evaluated using the new value of the variable.

Eg. int m=5;
y=++m;
In this case the value of y and m would be 6.

Postfix notation: The expression is evaluated first using the original value of the variable and then variable is incremented (or decremented by one).

Eg. int m=5;

y=m++;

The value of y would be 5 and m would be 6

6) Conditional Operator

The ternary Operator pair “?:” is available in c to construct conditional expressions of the form.

expression1? expression2: expression3

The expression1 is evaluated first,

- if it is true then the expression 2 is evaluated and its value becomes the value of the expression.
- If expression1 is false expression3 is evaluated and its value becomes the value of the expression.

Example:

Consider the following statements.

```
int a, b;
a=10;
b=15;
max= (a>b)? a: b;
```

In this example value of b will be assigned to max.

Program to find the maximum number between two numbers using conditional operator.

```
#include<stdio.h>
#include<conio.h>
int main( )
{
    int num1,num2,max;
    printf("\nEnter the two numbers:\n");
    scanf("%d%d",&num1,&num2);
    max=(num1>num2)?num1:num2;
    printf("\nThe largest number is %d",max);
    getch();
    return 0;
}
```

7) Bitwise operators

The operators which are used for the manipulation of data at bit level. These operators are used for testing the bits, shifting them right or left.

Operator	Meaning
&	Bitwise AND
	Bitwise OR
^	Bitwise exclusive OR
<<	Shift left
>>	Shift right

The truth tables for &, |, and ^ is as follows:

p	Q	p & q	p q	p ^ q
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Assume a = 60 and b = 13 in binary format, they will be as follows –

a = 0011 1100
b = 0000 1101

a&b = 0000 1100 =>12
a b = 0011 1101 =>61
a^b = 0011 0001 => 49

Bitwise shift operators

i. Left shift <<

This causes the operand to be shifted left by some bit positions.

General form:

Operand<<n

Eg.

n=60, n1=n<<3

N	0000 0000	0011 1100
First shift	0000 0000	0111 1000
Second shift	0000 0000	1111 0000
Third shift	0000 0001	1110 0000

After left shifting value of n1 becomes 480

ii. Right shift >>

This causes operand to be shifted to the right by some bit positions.

Operand >>n

Eg. n=60, n2=n>>3

N	0000 0000 0011 1100
First shift	0000 0000 0001 1110
Second shift	0000 0000 0000 1111
Third shift	0000 0000 0000 0111

After Right shifting value of n2 becomes 7

8) Special operators

C supports some special operators. Some of them are

i. Sizeof operator

The sizeof is a compile time operator and when used with an operand, it returns the number of bytes the operand occupies. The operand may be variable, constant or data type qualifier.

Examples:

```
m=sizeof(sum);  
n=sizeof(long int);  
k=sizeof(235L);
```

Program

```
#include<stdio.h>  
#include<conio.h>  
int main()  
{  
    float num;  
    printf("Number of bytes allocated=%d",sizeof(num));  
    getch();  
    return 0;  
}
```

ii. Comma operator

The comma operator can be used to link related expressions together .A comma-linked list are evaluated left to right and the value of right most expression is the value of combined expression.

For example the statement

```
value (x=10, y=5, x+y);
```

First assigns the value 10 to x then assigns 5 to y and finally assigns 15 (ie 10+5) to value.

Operator precedence and associativity

Write a short notes on: Operator precedence and associativity [PU:2018 fall]

- Operator precedence is a predefined rule of priority of operators. If more than one operators are involved in an expression the operator of higher precedence is evaluated first.
- Associativity indicates the order in which multiple operators of same precedence executes.

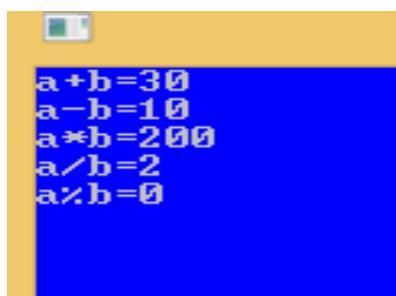
Precedence	Operator	Associativity
1	(), { }	Left to right
2	++, --, !	Right to left
3	*, /, %	Left to right
4	+, -	Left to right
5	<, <=, >, >=	Left to right
6	== , !=	Left to right
7	&&	Left to right
8		Left to right
9	?:	Right to left
10	=, *=, /=, %=, -=	Right to left

Some operators Examples:

Program to illustrate Arithmetic Operators

```
1 #include <stdio.h>
2 #include <conio.h>
3 int main()
4 {
5     int a=20, b=10, c;
6     c = a + b;
7     printf("a+b=%d\n", c);
8     c = a - b;
9     printf("a-b=%d\n", c);
10    c = a * b;
11    printf("a*b=%d\n", c);
12    c = a / b;
13    printf("a/b=%d\n", c);
14    c = a % b;
15    printf("a%b=%d", c);
16    getch();
17    return 0;
18 }
```

Output

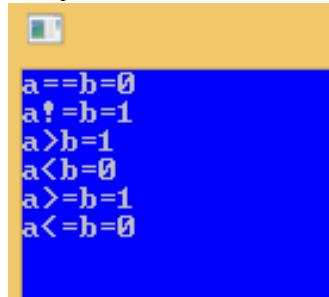


```
a+b=30
a-b=10
a*b=200
a/b=2
a%b=0
```

Program to illustrate Relational Operators

```
1 #include <stdio.h>
2 #include <conio.h>
3 int main()
4 {
5     int a=20, b=10;
6     printf("a==b=%d\n", a==b);
7     printf("a!=b=%d\n", a!=b);
8     printf("a>b=%d\n", a>b);
9     printf("a<b=%d\n", a<b);
10    printf("a>=b=%d\n", a>=b);
11    printf("a<=b=%d\n", a<=b);
12    getch();
13    return 0;
14 }
```

Output

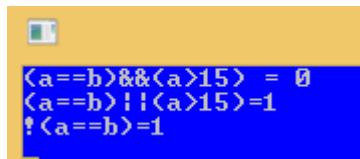


```
a==b=0
a!=b=1
a>b=1
a<b=0
a>=b=1
a<=b=0
```

Program to illustrate logical operators

```
1 #include<stdio.h>
2 #include<conio.h>
3 int main()
4{
5     int a=20, b=15, result;
6     result=( a==b ) &&( a>15 );
7     printf( "( a==b ) &&( a>15 ) = %d \n ", result );
8     result=( a==b ) || ( a>15 );
9     printf( "( a==b ) || ( a>15 ) = %d \n ", result );
10    result=! ( a==b );
11    printf( " ! ( a==b ) = %d \n ", result );
12    getch();
13    return 0;
14}
15
```

Output



```
c:\> a==b>&&(a>15) = 0
c:\> a==b>!<(a>15>=1
!<a==b>=1
```

Program to illustrate assignment operators

```
1 #include<stdio.h>
2 #include<conio.h>
3 int main()
4{
5     int a=5, c;
6     c=a;
7     printf( " c=%d \n ", c );
8     c+=a;
9     printf( " c=%d \n ", c );
10    c-=a;
11    printf( " c=%d \n ", c );
12    c*=a;
13    printf( " c=%d \n ", c );
14    c/=a;
15    printf( " c=%d \n ", c );
16    c%=a;
17    printf( " c=%d \n ", c );
18    getch();
19    return 0;
20}
21
```

output



```
c:\> c=5
c=10
c=5
c=25
c=5
c=0
```

Program to illustrate Increment/Decrement Operators

```
1 #include<stdio.h>
2 #include<conio.h>
3 int main()
4 {
5     int a=5, b=10, c=15, d=20, result;
6     result=++a;
7     printf("%d\n", result);
8     result=b++;
9     printf("%d\n", result);
10    result=-c;
11    printf("%d\n", result);
12    result=d--;
13    printf("%d\n", result);
14    getch();
15    return 0;
16 }
17
```

Output



```
6
10
14
20
```

Program to illustrate Bitwise operators

```
#include<stdio.h>
#include<conio.h>
int main()
{
int n1=60,n2=13,x,y,z,left,right;
x=n1&n2;
y=n1|n2;
z=n1^n2;
left=n1<<3;
right=n1>>3;
printf("AND=%d\n",x);
printf("OR=%d\n",y);
printf("XOR=%d\n",z);
printf("Value after left shift=%d\n",left);
printf("Value after right
shift=%d\n",right);
getch();
return 0;
}
```

output



```
C:\User
AND=12
OR=61
XOR=49
Value after left shift=480
Value after right shift=7
```

1) WAP to display your name.

```
#include<stdio.h>
#include<conio.h>
int main()
{
char name[20];
printf("Enter your name:");
gets(name);
printf("\nYour name is:");
puts(name);
getch();
return 0;
}
```

2) WAP to calculate area and circumference of circle having the radius r should be taken from user.

```
#include<stdio.h>
#include<conio.h>
int main()
{
float r,area,circum;
printf("Enter the radious of circle");
scanf("%f",&r);
area=3.1416*r*r;
circum=2*3.1416*r;
printf("\nArea of circle=%f",area);
printf("\nCircumference of circle is %f",circum);
getch();
return 0;
}
```

3) WAP to swap two numbers

```
#include<stdio.h>
#include<conio.h>
int main()
{
int a,b,temp;
a=5;
b=10;
printf("Number before swapping a=%d
and b=%d",a,b);
temp=a;
a=b;
b=temp;
printf("\nNumber after swapping a=%d
and b=%d",a,b);
getch();
return 0;}
```

4) WAP to swap two numbers without using third variable.

```
#include<stdio.h>
#include<conio.h>
int main()
{
int a,b;
a=5;
b=10;
printf("\nNumber before swappig a=%d and
b=%d",a,b);
a=b+a;
b=a-b;
a=a-b;
printf("\nNumber after swappig a=%d and b=%d",a,b);
getch();
return 0;
}
```

5) WAP that will convert temperature in centigrade into Fahrenheit.

```
#include<stdio.h>
#include<conio.h>
int main()
{
float centi,fah;
printf("\nEnter the temperature in
centigrade");
scanf("%f",&centi);
fah=1.8*centi+32;
printf("\nThe equivalent temperature in
fahrenheit is %f",fah);
getch();
return 0;
}
```

6) WAP to check the given 3 digit number is armstrong or not.

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
int main()
{
int num,a,b,c,arm;
printf("\nEnter the number");
scanf("%d",&num);
c=num%10;
b=(num/10)%10;
a=num/100;
arm=pow(a,3)+pow(b,3)+pow(c,3);
if(num==arm)
{
    printf("Entered number is armstrong number");
}
else
{
    printf("Entered number is not armstrong
number");
}
getch();
return 0;
}
```

7) WAP to find the reverse of a given (3 digit) number.

```
#include<stdio.h>
#include<conio.h>
int main()
{
int num,a,b,c,rev;
printf("Enter the number");
scanf("%d",&num);
c=num%10;
b=(num/10)%10;
a=num/100;
rev=c*100+b*10+a;
printf("Reverse of given number is %d",rev);
getch();
return 0;
}
```

8) WAP to find the sum of digit of given (3 digit) number.

```
#include<stdio.h>
#include<conio.h>
int main()
{
int num,a,b,c,sum;
printf("Enter the number");
scanf("%d",&num);
c=num%10;
b=(num/10)%10;
a=num/100;
sum=a+b+c;
printf("Sum of digit of given number is %d",sum);
getch();
return 0;
}
```

9) WAP to check the given year is leap year or not.

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int year;
    printf("Enter the year");
    scanf("%d",&year);
    if((year%4==0&&year%100!=0)||year%400==0)
    {
        printf("%d is leap year",year);
    }
    else
    {
        printf("%d is not leap year",year);
    }
    getch();
    return 0;
}
```

What is the purpose of qualifiers register and volatile? [PU: 2013 spring]

A volatile keyword is a qualifier which prevents the objects, from the compiler optimization and tells the compiler that the value of the object can be change at any time without any action being taken by the code. It prevents from the cache a variable into a register and ensures that on every access variable is fetched from the memory.

The volatile keyword is mainly used where we directly deal with GPIO, interrupt or flag Register. It is also used where a global variable or buffer is shared between the threads

Example of declaration: volatile int status;

Similary, register is a storage class specifier. The variables which are most frequently used in a C program can be put in registers using register keyword. The keyword register hints to compiler that a given variable can be put in a register because register are faster than memory to access.

Example of declaration: register int i;

Other important questions:

1. Differentiate Keywords and Identifier.
2. Define the following terms with suitable examples. Statements, Tokens and format specifier.
3. **Write the short notes on**
 - Escape sequence
 - Pseudo code
 - Preprocessor directives
 - Symbolic Constant
 - Delimiter
 - ASCII
4. Define the following terms with suitable example[pu:2014 fall]
 - i) Statement
 - ii) Token
 - iii) Format specifier
5. Define the term operators, variable, and constant with example. **[PU:2012 fall]**
6. Describe four basic data types. How could you extend the range of values they represent? [PU:2013 spring]

CHAPTER-4

Control Structures

Control structure enables us to specify the order in which the various instructions in a program are to be executed by the computer. In other words control instructions determine the “flow of control” in a program. There are 3 types of control instructions in C.

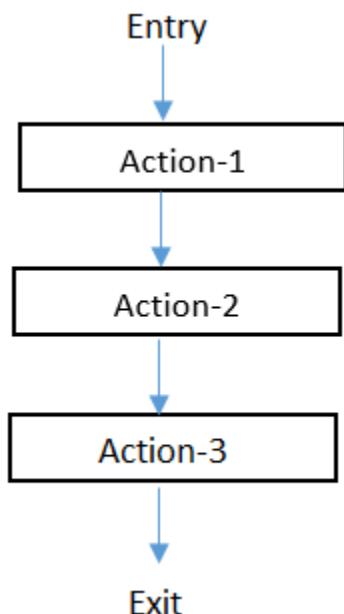
They are:

- 1 Sequential Control instruction
- 2 Selection or decision control instruction
- 3 Repetition or looping control instruction

1. Sequential Control Instruction

- Instructions are executed in the same order in which they appear
- Each instruction is executed exactly once.
- No condition is evaluated.

Flowchart



Example

Program to find the sum of two numbers.

```
#include<stdio.h>
#include<conio.h>
int main()
{
int num1,num2,sum;
printf("Enter the two numbers");
scanf("%d%d",&num1,&num2);
sum=num1+num2;
printf("Sum of two numbers is %d",sum);
getch();
return 0
}
```

2. Selection/Branching/Decision Control instructions

They are used when we have a number of situations where we may need to change the order of execution of statements based on certain conditions.

- i. simple if statement
- ii. if else statement
- iii. nested if else statement
- iv. else if ladder

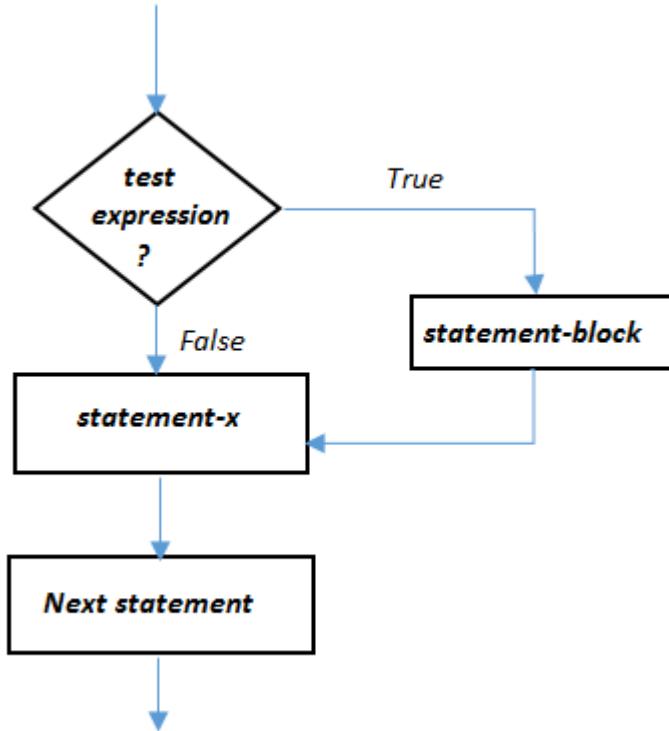
i) Simple if statement

Evaluates the expression first then:

- If the value of expression is true , it executes the statement within the block
- Otherwise it skips the statements within its block and continues from the first statement outside the block.

Syntax

```
if(test expression)
{
statement-block;
}
statement-x;
```

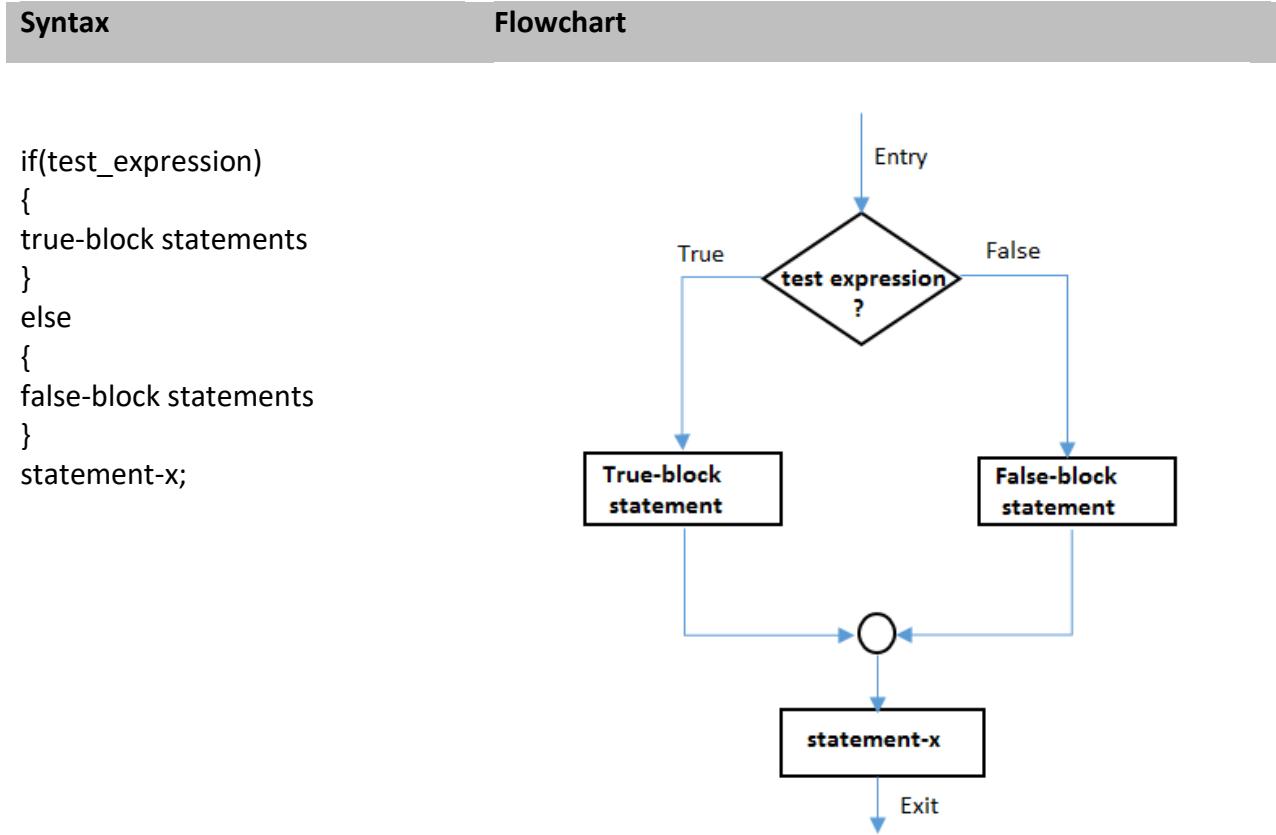
Flowchart

WAP to input the average marks of a student and add 10% bonus marks if his/her average marks is greater than or equal to 65.

```
#include<stdio.h>
#include<conio.h>
int main()
{
float marks;
printf("Enter the marks\n");
scanf("%f",&marks);
if(marks>=65)
{
    marks=marks+marks*0.1;
}
printf("Final marks=%f",marks);
getch();
return 0;
}
```

ii) If-else statement

- Extension of simple if statement.
- If test expression is true,
 - then true block statement(s) (*immediately following the if statements are executed*)
 - otherwise false block statements are executed.



WAP to find the maximum number between two numbers

```
#include<stdio.h>
#include<conio.h>
int main()
{
int num1,num2;
printf("Enter the first number\n");
scanf("%d",&num1);
printf("Enter the second number\n");
scanf("%d",&num2);
```

```

if(num1>num2)
{
printf("Maximum number=%d",num1);
}

else
{
printf("Maximum number=%d",num2);
}
getch();
return 0;
}

```

iii) Nested if-else statement

When a series of decisions are involved, we may have to use more than one if. . . . else statement in nested form as shown below.

Syntax	Flowchart
<pre> If(test condition-1) { If(test condition-2) { Statement -1; } else { Statement-2; } } else { Statement-3; } } Statement x; </pre>	<pre> graph TD Entry((Entry)) --> D1{test condition-1 ?} D1 -- False --> S3[Statement-3] D1 -- True --> D2{test condition-2 ?} D2 -- False --> S2[Statement-2] D2 -- True --> S1[Statement-1] S2 --> Join(()) S1 --> Join Join --> Sx[Statement-x] Sx --> NS[Next Statement] NS --> End(()) </pre>

Program to find the largest number among three numbers.

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

int main()
{
    int a,b,c;
    printf("Enter three number\n");
    scanf("%d%d%d",&a,&b,&c);
    if(a>b)
    {
        if(a>c)
        {
            printf("The largest number is %d",a);
        }
        else
        {
            printf("The largest number is %d",c);
        }
    }
    else
    {
        if(b>c)
        {
            printf("Largest number is %d",b);
        }
        else
        {
            printf("largest number is %d",c);
        }
    }
}
```

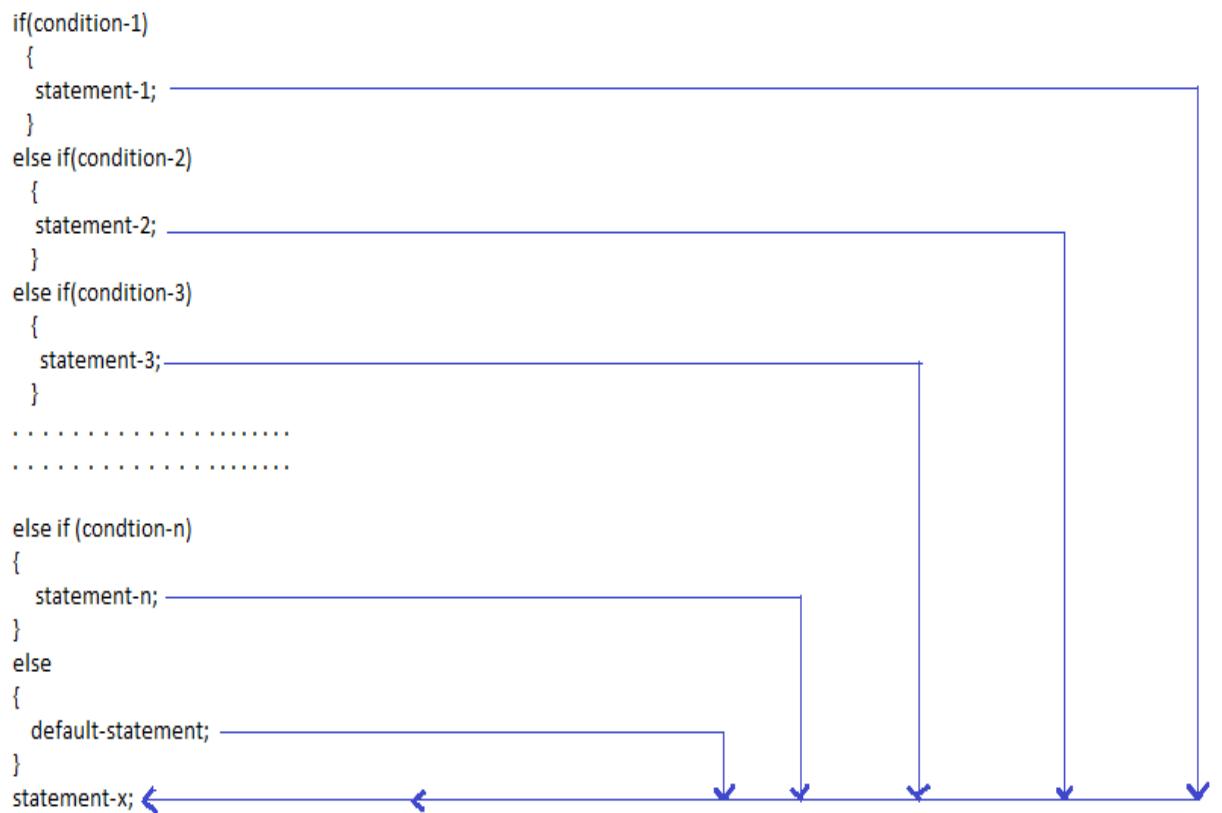
iv) Else-if ladder

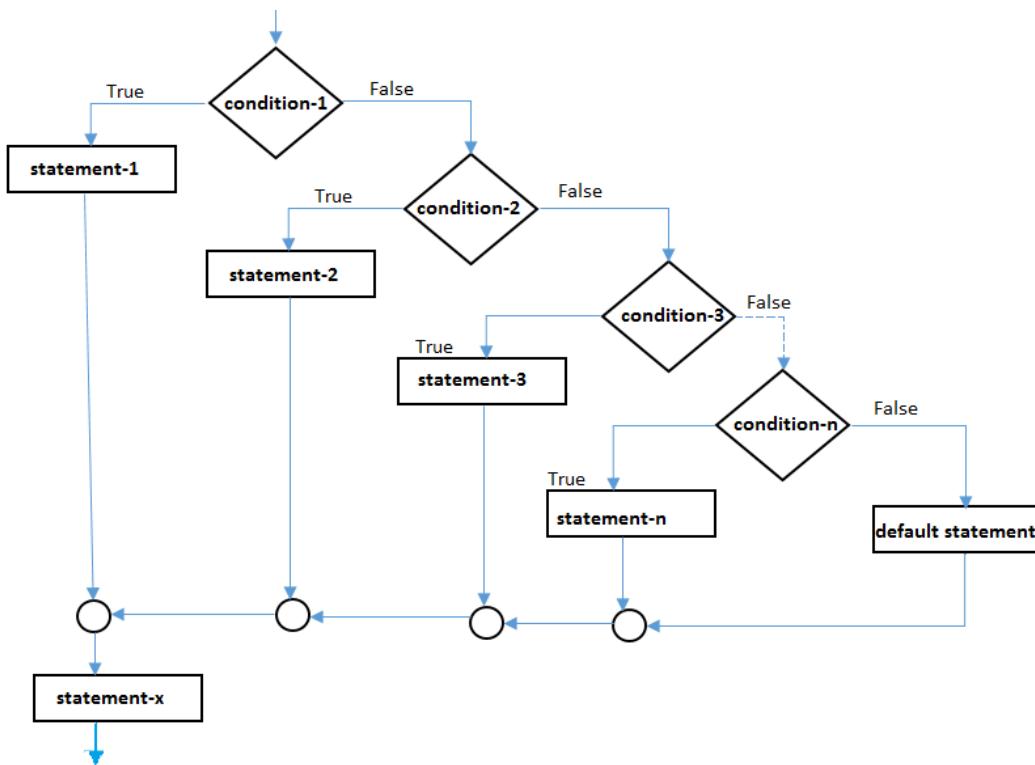
It is used when multiple decisions are involved. Here, the condition expression is evaluated in order.

- If any of these expression is true, the statement associated with it is executed and this terminates the whole chain.
- If none of the expression is true then statement associated with final else is executed.

Syntax:

```
if(condition-1)
{
    statement-1; _____
}
else if(condition-2)
{
    statement-2; _____
}
else if(condition-3)
{
    statement-3; _____
}
.....
.....
else if (condition-n)
{
    statement-n; _____
}
else
{
    default-statement; _____
}
statement-x; <-----<
```

**Flowchart**



An electricity board charges according to the following rates.

For the first 100 units Rs 40 Per Unit

For the next 200 units.....Rs. 50 Per Unit

For the beyond 300 Units.....Rs.60 Per unit

All users are also charge meter charge. Which is equal to Rs.50. Write a program to to read number of units consumed and print out total charges.[PU:2015 spring,PU:2015 fall]

```

#include<stdio.h>
#include<conio.h>
#define meter_charge 50
int main()
{
int units,charge,totalcharge;
printf("Enter the number of units\n");
scanf("%d",&units);
if(units<=100)
{
charge=units*40;
}
else if(units<=300)
{
charge=100*40+(units-100)*50;
}
else
{
charge=100*40+200*50+(units-300)*60;
}
  
```

```

totalcharge=charge+meter_charge;
printf("Total charge=%d",totalcharge);
getch();
return 0;
}

```

WAP to read the marks of 5 subjects and based on percentage printout the following.

Percentage >=80	Distinction
Percentage 60 - 79	First division
Percentage 45 - 59	Second division
Percentage 32 - 44	Third division
Percentage less than or equals to 31	Failed

```

#include<stdio.h>
#include<conio.h>
int main()
{
float sub1,sub2,sub3,sub4,sub5,per;
printf("Enter the marks of 5 subjects\n");
scanf("%f%f%f%f%f",&sub1,&sub2,&sub3,&sub4,&sub5);
per=(sub1+sub2+sub3+sub4+sub5)/5;
if(per>=80)
{
printf("Distinction");
}
else if(per>=60)
{
printf("First division");
}
else if(per>=45)
{
printf("Second division");
}
else if(per>=32)
{
printf("Third division");
}
else
{
printf("Failed");
}
getch();
return 0;
}

```

3. Repetition/Iteration/Loop Control Instructions

- Describe the working of loop and while loop with flowcharts and examples.[PU:2013 fall]
- Differentiate between while loop and do while loop. [PU: 2016 spring]
- What do you mean by selective and repetitive statement? [PU: 2017 spring]
- *[Hint for solution: if,if-else,else-if ladder are selective statement and loop statement such as for,while,do-while are repeatative statement]*
- Explain entry controlled and exit controlled loops with examples.[PU:2016 fall]
- Differentiate between while and do while loops with suitable examples.[PU:2017 fall]
- Differentiate pre-test and post-test loop.[PU:2018 spring]
- *[Hint for solution: while and for loop are pre-test loop and do while is post-test loop]*
- Why do you mean by entry controlled loop and exit controlled loop ?Explain the different types of looping constructs available in C with suitable examples.[PU:2014 spring]
- What is entry controlled and exit controlled loop?[PU:2019 fall]

Loop control instructions causes a program to execute the certain block of code until some conditions for termination of loop are satisfied.

Basically there are three types of loop control instructions.

- i) while Loop (**Entry Controlled loop**)
- ii) do while loop (**Exit Controlled loop**)
- iii) for loop

i) While loop

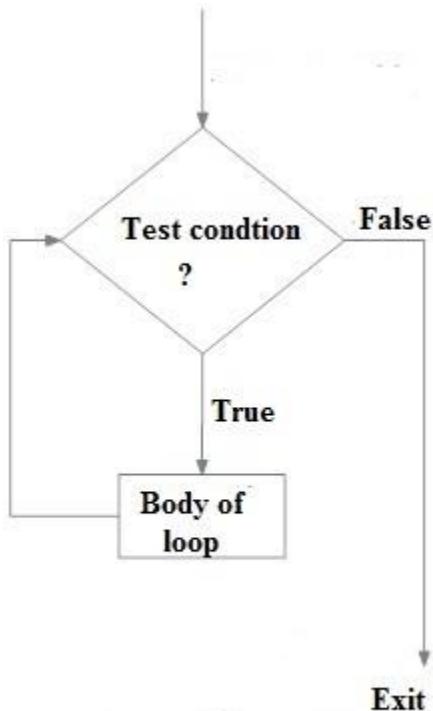
Test condition is evaluated first

- If the condition is true the body of the loop is executed.
After the execution of the body, test condition is again evaluated and if it is true body is executed once again. This process goes on until test condition is false.
- If test condition is false, the body of the loop will not be executed and control is transferred out of the loop.

Syntax:

```
while (test condition)
{
    body of the loop
}
```

Flowchart



Flowchart of while loop(Entry controlled loop)

Program to print 10 natural numbers using while loop.

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int i;
    i=1;
    while(i<=10)
    {
        printf("%d\n",i);
        i++;
    }
    getch();
    return 0;
}
```

ii) do while loop /exit controlled loop

The test is performed at end of the body of the loop and therefore the body of the loop is executed unconditionally for the first time.

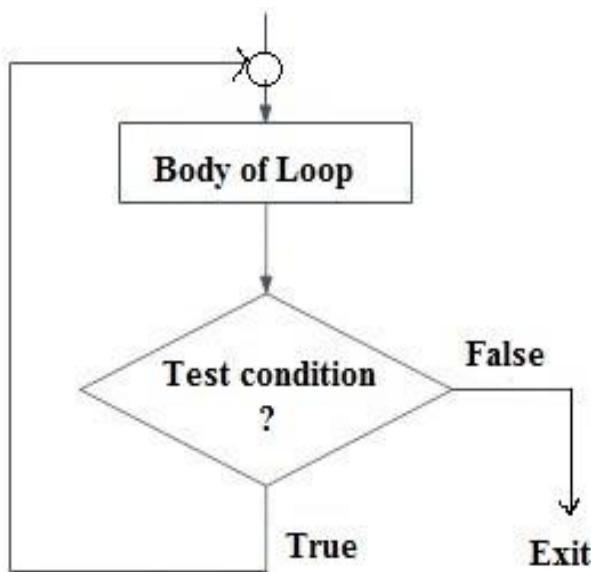
Then, test condition is evaluated.

- If test condition is true, the body of the loop is executed again. This process goes on until the test condition is false.
- When test condition is false, loop is terminated and control goes to the statement that appears immediately after the while statement.

Syntax

```
do  
{  
    Body of the loop  
} while(test condition);
```

Flowchart



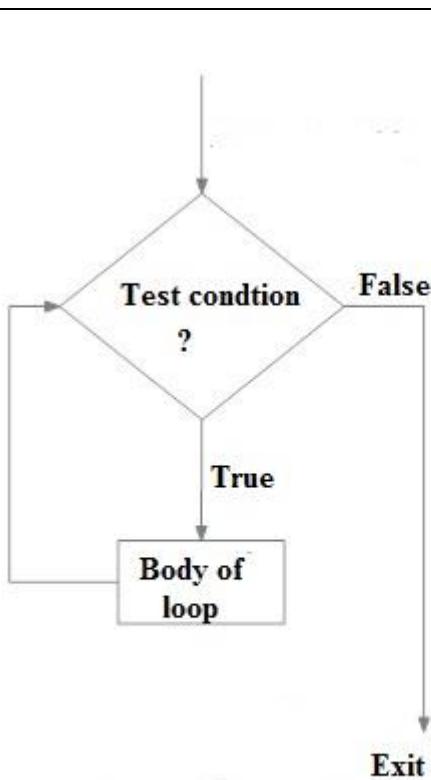
Flowchart of do-while(Exit controlled) loop

Program to print 10 natural numbers using do-while loop.

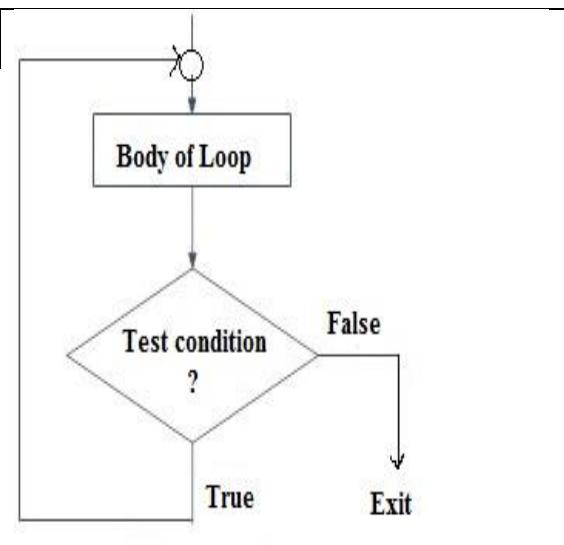
```
#include<stdio.h>
#include<conio.h>
int main()
{
int i;
i=1;
do
{
    printf("%d\n",i);
    i++;
}while(i<=10);
getch();
return 0;
}
```

Differentiate Entry controlled loop (While loop) and Exit controlled loop (Do- While loop) with examples.

While loop(Entry controlled loop)	Do-while loop(Exit controlled loop)
Test condition is evaluated at beginning of the loop execution.	Test condition is evaluated at the end of the body of the loop.
The body of the loop will execute only if the test condition is true.	The body of the loop will execute at least once without depending on the test condition.
Syntax: while(test condition) { Body of the loop }	Syntax: do { Body of the loop }while(test condition);



Flowchart of while loop(Entry controlled loop)



Flowchart of do-while(Exit controlled) loop

Program to illustrate concept of while loop

```

int main()
{
int i=0;
while(i>=4)
{
printf("Hello\n");
i++;
}
return 0;
}

```

Output: Nothing will display.

Program to illustrate concept of do- while loop

```

int main()
{
int i=0;
do
{
printf("Hello\n");
i++;
}while(i>=4);
return 0;
}

```

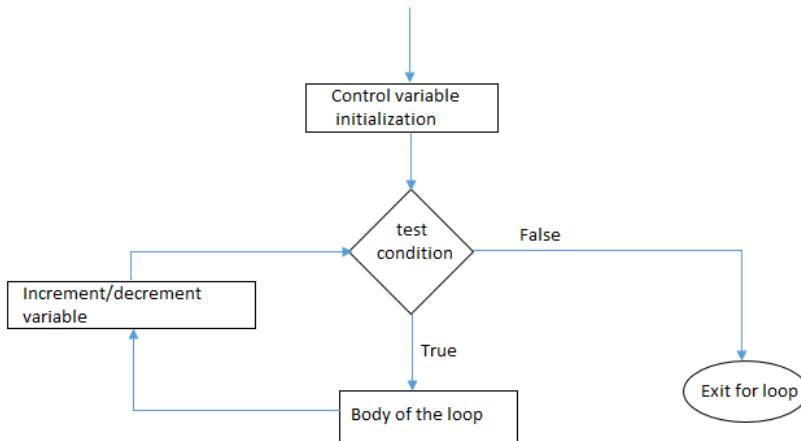
Output: Hello

iii) For loop

Syntax:

```
for(initialization; test condition; increment/decrement)
{
    body of the loop
}
```

Flowchart:



Program to print 10 natural number using for loop.

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int i;
    for(i=1;i<=10;i++)
    {
        printf("%d\n",i);
    }
    getch();
    return 0;
}
```

WAP to print numbers from 200 to 500 which are divisible by 5 and find their sum.

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int i,sum=0;
    for(i=200;i<=500;i++)
    {
        if(i%5==0)
        {
            printf("%d\n",i);
            sum=sum+i;
        }
    }
    printf("The sum is %d",sum);
    getch();
    return 0;
}
```

For Loop vs while Loop

For Loop	While Loop
It have definite number of iterations.	It may or may not have definite number of iterations.
Keyword for is used.	Keyword while is used.
In 'for' loop the initialization once done is never repeated.	In while loop if initialization is done during condition checking, then initialization is done each time the loop iterate.
In 'for' loop iteration statement is written at top, hence, executes only after all statements in loop are executed.	In 'while' loop, the iteration statement can be written anywhere in the loop
Syntax: for(initialization; test-condition;increment/decrement) { body of the loop }	Syntax: while(test condition) { Body of loop }

For example:

```
int main()
{
int n;
for(n=0;n<10;n++)
{
printf("Nepal\n");
}
return 0;
}
```

For example:

```
int main()
{
int n=10;
while(n!=0)
{
printf("Nepal\n");
n--;
}
return 0;
}
```

Nested Loop**Write short notes on: Nested loop[PU:2013 fall]**

A loop inside another loop is called nesting of loops. There can be any number of loops inside one another with any combinations depending on the complexity of program.eg. A for loop inside a while loop or while loop inside a for loop.

Nested while loop

```
while(test-condition1)
{
Statement(s);
while(test-condition2)
{
statement(s);
.....
}
.....
}
```

Nested do-while loop

```
do
{
Statement(s);
do
{
Statement(s);
.....
}while(test-condition2);
.....
}while(test-condition1);
```

Nested for loop

```
for(initialization;testcondition;increment/decrement)
{
Statement(s);
for(initialization;testcondition;increment/decrement)
{
Statement(s);
}
}
```

Example: Nested for loop

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int i,j;
    for(i=0;i<=5;i++)
    {
        for(j=1;j<=i;j++)
        {
            printf("*\t");
        }
        printf("\n");
    }
    getch();
    return 0;
}
```

Jump statements

Important questions from this topic

- Why do we need break and continue statement. [PU:2017 spring]
- Compare continue and break statements. [PU:2016 fall]
- Differentiate between break and continue statements with suitable example program. [PU:2018 fall]
- Why do you use “continue” and “break” statement in your program? Explain with suitable example program. [PU:2018 spring]
- Explain break and continue statement with example.[PU:2014 fall]
- Write short notes on:
- Break and continue statement.[PU:2013 spring][PU:2019 spring]

Jump Statement makes the control jump to another section of the program unconditionally when encountered. It is usually used to terminate the loop or switch-case instantly. It is also used to escape the execution of a section of the program. There are basically three types of jump statements.

- Break
- Continue
- Goto

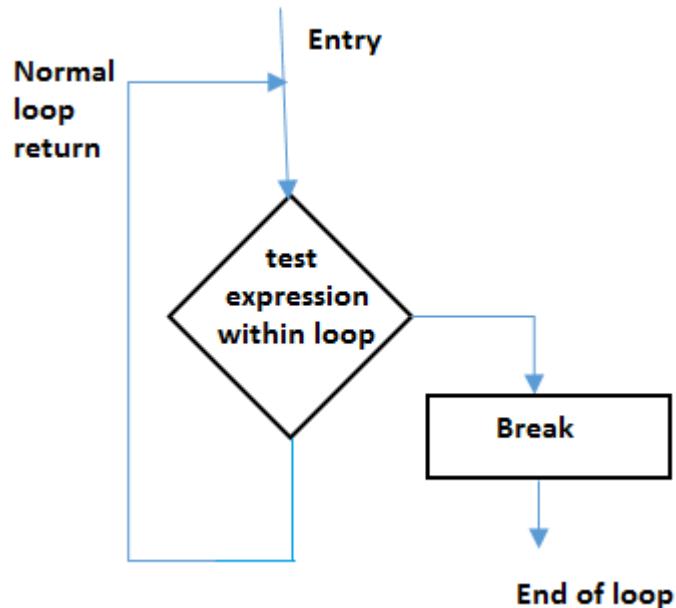
i) Break statement

The break statement is used inside the loop or switch statement.

The execution of break statement will abort the loop and continue to execute statements followed by loop.

Similarly, execution of break statement causes bypass the rest of statement in switch and takes control out of the switch statement.

Flowchart:



Example

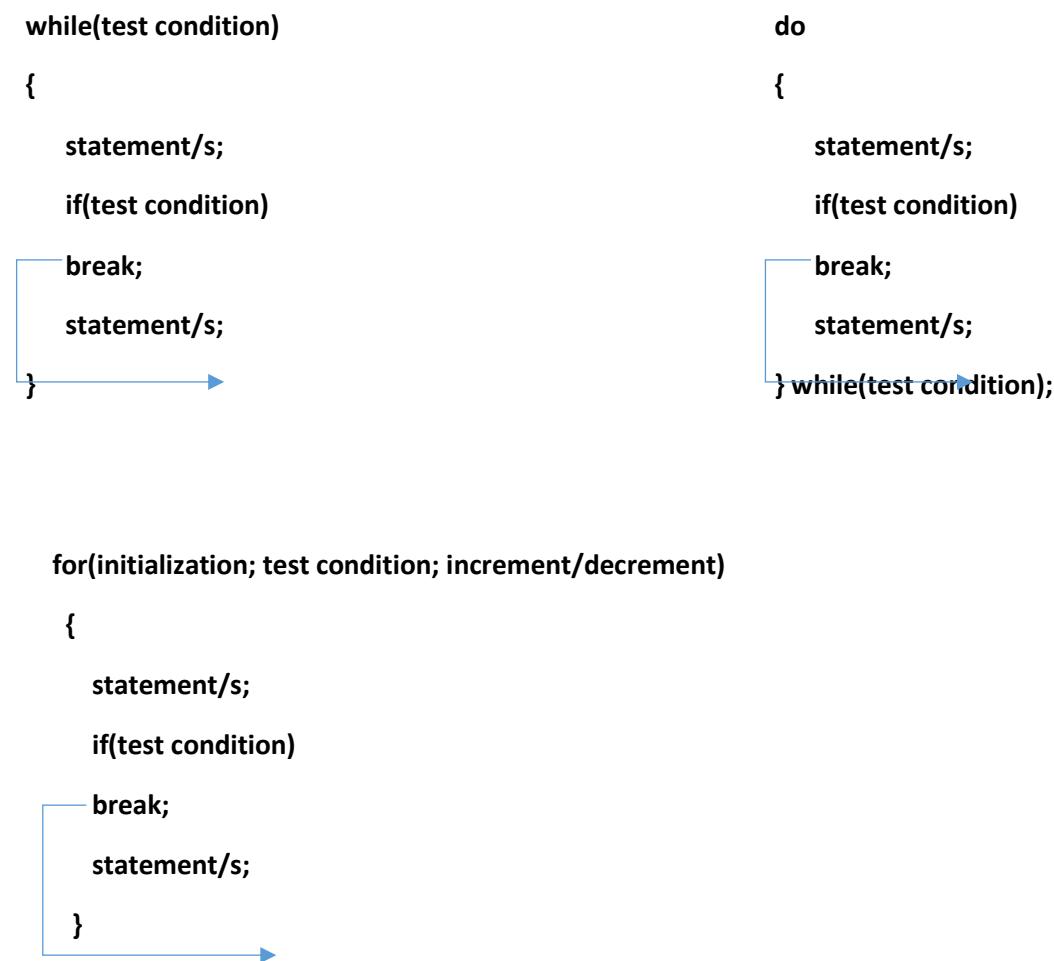
```
int main()
{
int i;
for(i=1;i<=5;i++)
{
printf("%d\t",i);
if(i==3)
break;
}
printf("\n loop terminates here.");
return 0;
}
```

Output:

1 2 3

Loop terminates here.

How break statement works?



ii) Continue statement

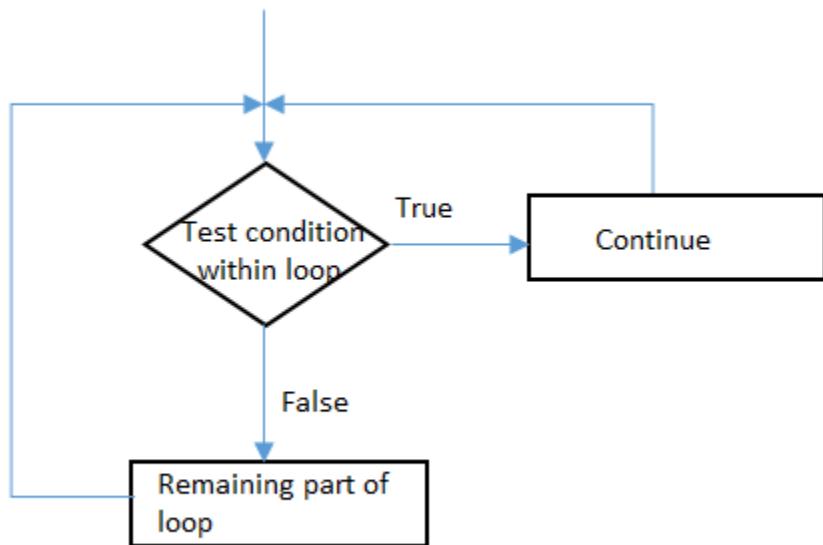
When continue statement is encountered inside the loop, it skips the execution of statements specified within the body of the loop and control automatically passes to the next iteration of the loop.

Example

```
int main()
{
    int i;
    for(i=1;i<=5;i++)
    {
        if(i==3)
            continue;
        printf("%d\t",i);
    }
    return 0;
}
```

Output:

1 2 4 5

**How continue statement works?**

► **while(test condition)**
{
 statement/s;
 if(test condition)
 continue;
 statement/s;
}

► **do**
{
 statement/s;
 if(test condition)
 continue;
 statement/s;
} **while(test condition);**

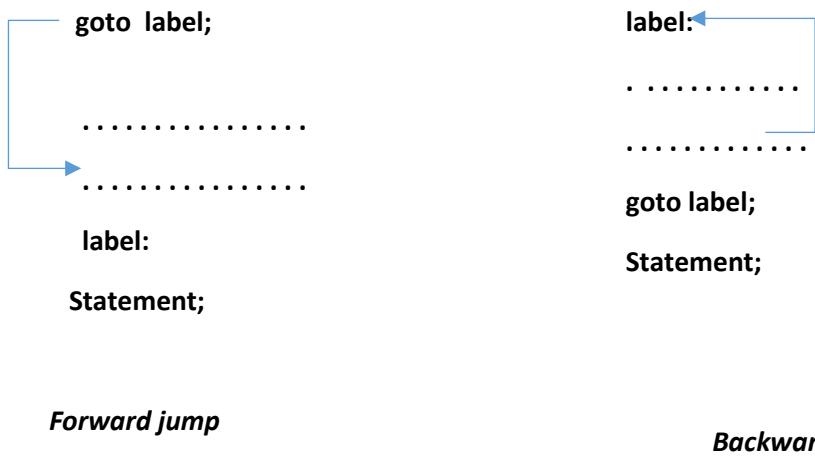
► **for(initialization; test condition; increment/decrement)**
{
 statement/s;
 if(test condition)
 continue;
 statement/s;
}

iii) goto statement

Write a short notes on: Goto statement [PU:2014 spring][PU:2014 fall][PU:2016 fall]

The goto statement is used to alter the program execution sequence by transferring the control to some other parts of the program.

Syntax:



where ,label is an identifier that is used to label the target statement to which control will be transferred.

Note: Use of goto statement is highly discouraged in any programming language because it makes difficult to trace the control flow of program, making the program hard to modify and understand.

Program to illustrate the working of goto statement

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int a,b;
    printf("Enter two numbers\n");
    scanf("%d%d",&a,&b);
    if(a>b)
        goto label1;
    else
        goto label2;
    label1:
        printf("Greatest number=%d",a);
    label2:
        printf("Greatest number=%d",b);
    getch();
    return 0;
}
```

Case-control instruction (switch-case statement)

Write a short notes on: Switch case statement.[PU:2015 fall]

The control statement that allows us to make a decision from number of choices is called switch case statement.

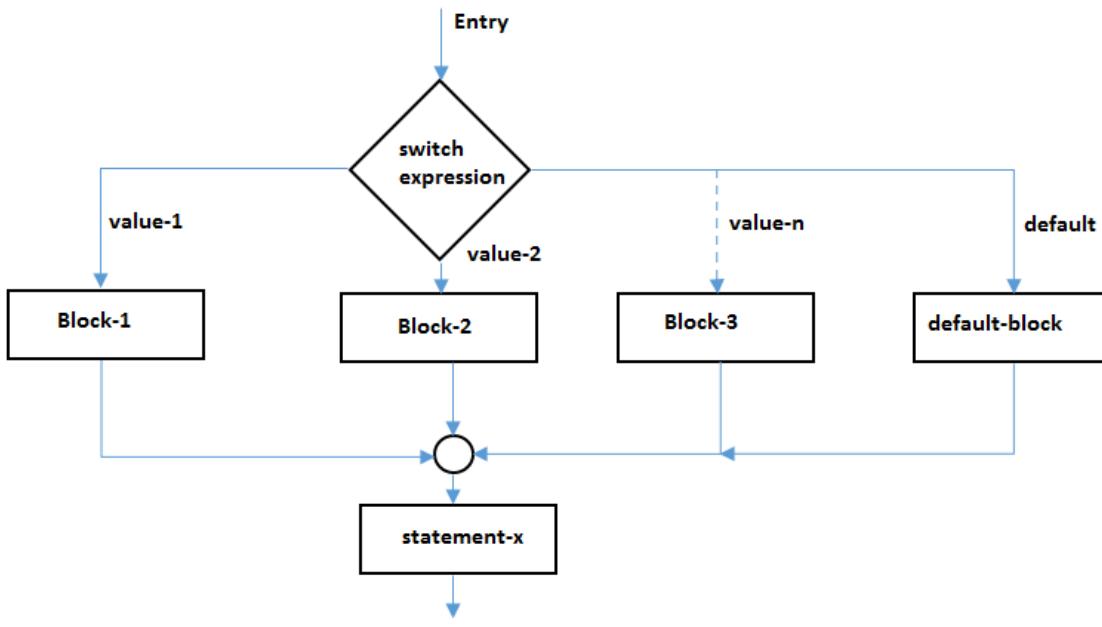
The switch case statement successively test the value of a given variables (an expression) with a list of case value (integer or character constants)

- When match is found, the statement associated with that case is executed.
- If none of the case value matches the expression then default statement is executed.

General form:

```
switch(expression)
{
    case value-1:
        block-1;
        break;
    Case value-2:
        block-2;
        break;
    .....
    default:
        default-block;
        break;
}
statement-x;
```

Flowchart:



for example, consider the program to display the corresponding days of a week according to the numbers entered.

```
#include<stdio.h>
#include<conio.h>
int main()
{
int day;
printf("Enter the numeric day of week\n");
scanf("%d",&day);
switch (day)
{
case 1:
printf("Day is Sunday")      ;
break;
case 2:
printf("Day is Monday");
break;
case 3:
printf("Day is Tuesday");
break;
case 4:
printf("Day is Wednesday");
break;
case 5:
printf("Day is Thursday");
break;
case 6:
printf("Day is Friday");
break;
```

```

case 7:
printf("Day is Saturday");
break;
default:
printf("Invalid choice!");
}
getch();
return 0;
}

```

WAP to display the following menu and perform the following operations.

1. Find the simple interest
2. Convert degree Celsius to Fahrenheit
3. Convert character into ASCII code
4. Find the area of circle
5. Exit from the program

and perform above operation until user want to exit.

```

#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int main()
{
int choice;
float p,t,r,i,c,f,area;
char ch;
while(1)
{
printf("\nMenu");
printf("\n1.Find simple intrest");
printf("\n2.convert celcius of fahreheit");
printf("\n3.convert character to ASCII code");
printf("\n4.Find area of circle");
printf("\n5.Exit from program");
printf("\nEnter your choice\n");
scanf("%d",&choice);
switch(choice)
{
case 1:
printf("Enter Principal,Time and Rate\n");
scanf("%f%f%f",&p,&t,&r);
i=(p*t*r)/100;
printf("simple intrest is %f",i);
break;
}
}

```

```

case 2:
printf("Enter the temperature in celcius\n");
scanf("%f",&c);
f=1.8*c+32;
printf("Converted temp is %f",f);
break;
case 3:
printf("Enter a character\n");
fflush(stdin);
scanf("%c",&ch);
printf("The corresponding ASCII code is %d",ch);
break;
case 4:
printf("Enter the radius of circle");
scanf("%f",&r);
area=3.14*r*r;
printf("Area=%f",area);
break;
case 5:
exit(0);
default:
printf("Wrong choice!");
}
}
getch();
return 0;

```

Difference between if-else and switch

If-else	Switch
1. Execution of statement will be depend upon the output of the expression inside if statement.	1. Which statement will be executed is decided by user.
2. if-else statement uses multiple statement for multiple choices.	2. switch statement uses single expression for multiple choices.
3. if statement evaluates integer, floating-point type as well boolean type.	3. switch statement evaluates only character or integer value.
4. Either if statement will be executed or else statement is executed.	4. switch statement execute one case after another till a break statement is appeared or the end of switch statement is reached.
5. It does not require break statement because only one of the blocks of code is executed.	5. It needs the involvement of break statement to avoid execution of block just below the current executing block.

- | | |
|--|--|
| <p>6. If the condition inside if statements is false, then by default the else statement is executed if created.</p> | <p>6. If the condition inside switch statements does not match with any of cases, for that instance the default statements is executed if created.</p> |
|--|--|

Differentiate between switch and nested if else statements with a suitable example.[PU:2017 spring]

Switch	nested if-else
It is easy to understand for multiple selections.	It becomes complicated for multiple selections.
It uses a single expression for all cases, but each case must have a constant value of integer type or character type.	It uses an independent expression for each case.
Only a single expression is given in the switch statement which returns a single value. The test condition cannot be given in a specified range.	The test condition can be given in a special range of value. If the given condition matches then the statements under it will be executed.
General form: <pre>switch(expression) { case value-1: block-1; break; Case value-2: block-2; break; default: default-block; break; } statement-x;</pre>	General form: <pre>If(test condition-1) { If(test condition-2) { Statement -1; } else { Statement-2; } else { Statement-3; } } Statement x;</pre>

<pre> graph TD Entry --> Switch{switch expression} Switch -- value-1 --> Block1[Block-1] Switch -- value-2 --> Block2[Block-2] Switch -- value-n --> Block3[Block-3] Switch -- default --> DefaultBlock[default-block] Block3 -.-> DefaultBlock Block1 --> Merge(()) Block2 --> Merge Block3 --> Merge DefaultBlock --> Merge Merge --> StatementX[statement-x] StatementX --> End </pre>	<pre> graph TD Entry --> Cond1{test condition-1 ?} Cond1 -- False --> Statement3[Statement-3] Cond1 -- True --> Cond2{test condition-2 ?} Cond2 -- False --> Statement2[Statement-2] Cond2 -- True --> Statement1[Statement-1] Statement2 --> Merge(()) Statement1 --> Merge Merge --> StatementX[Statement-x] StatementX --> NextStatement[Next Statement] NextStatement --> End </pre>
<p>It needs involvement of the break statement to avoid the execution of the block just below the current executing block.</p> <p>If the condition inside switch statements does not match with any of cases, for that instance the default statements is executed if created.</p>	<p>It does not require break statement because only one of the blocks of code is execute at a time.</p> <p>If the condition inside if statements is false, then by default the else statement is executed if created.</p>
Assignment: <p>What is menu driven structure explain with suitable examples.[PU:2016 spring]</p> <p>Describe the different types of decision control statements used in C programming with their syntax. [PU: 2012 fall]</p>	

Programs related to Decision control statements

1. WAP to find second largest number among three numbers.
2. WAP to find the largest number among four numbers.

```
#include<stdio.h>
#include<conio.h>
int main()
{
int a,b,c;
printf("Enter three numbers\n");
scanf("%d%d%d",&a,&b,&c);
if(a>b&&a<c| a>c&&a<b)
{
printf("second largest(middle) number=%d",a);
}
else if(b>a&&b<c| b>c&&b<a)
{
printf("second largest(middle) number=%d",b);
}
else
{
printf("second largest(middle) number=%d",c);
}
getch();
return 0;
}
```

```
#include<stdio.h>
#include<conio.h>
int main()
{
int a,b,c,d;
printf("Enter the four numbers\n");
scanf("%d%d%d%d",&a,&b,&c,&d);
if(a>b&&a>c&&a>d)
{
printf("max=%d",a);
}
else if(b>c&&b>d)
{
printf("max=%d",b);
}
else if(c>d)
{
printf("max=%d",c);
}
else
{
printf("max=%d",d);
}
getch();
return 0;
}
```

3. An electricity board charges according to following rates.

For the first 20 units..... Rs 80

For the next 80 units.....Rs.7.5 per unit

For the next 100 units ...Rs 8.5 per unit

For the beyond 200 unitsRs 9.5 per unit

And Tax 15% in total amount is charged to all users.

Write a program to read number of units consumed and print out the total charges.

[PU:2012 fall]

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int units;
    float amount,total;
    printf("Enter the number of units\n");
    scanf("%d",&units);
    if(units<=20)
    {
        amount=80;
    }
    else if(units<=100)
    {
        amount=80+(units-20)*7.5;
    }
    else if(units<=200)
    {
        amount=80+80*7.5+(units-100)*8.5;
    }
    else
    {
        amount=80+80*7.5+100*8.5+(units-200)*9.5;
    }
    total=amount+amount*0.15;
    printf("Total charges=%f",total);
    getch();
    return 0;
}
```

4. WAP to read the three sides of triangle and print area for the valid data and to print "invalid data" if either one side of the triangle is greater or equals to the sum of other two sides.

(Area= $\sqrt{s(s - a)(s - b)(s - c)}$) where a, b, c are the three sides and s=(a+b+c)/2

```
#include<stdio.h>
#include<conio.h>
int main()
{
    float a,b,c,s,area;
    printf("Enter the three sides of traingle\n");
    scanf("%f%f%f",&a,&b,&c);
    if(a>=(b+c) || b>=(a+c) || c>=(a+b))
    {
        printf("Invalid Data");
    }
    else
    {
        s=(a+b+c)/2;
        area=sqrt(s*(s-a)*(s-b)*(s-c));
        printf("Area of traingle=%f",area);
    }
    getch();
    return 0;
}
```

5. Consider the following tax table

Income	Tax
< RS.10,000	Nil
Rs. 10,000 to Rs.19,999	10%
RS 20,000 to Rs.29,999	15%
RS 30,000 to Rs.49,999	20%
> =Rs.50,000	25%

Write a program to compute tax amount when income is given.

```
#include<stdio.h>
#include<conio.h>
int main()
{
    float income,tax;
    printf("Enter the income\n");
    scanf("%f",&income);
    if(income<10000)
    {
        tax=0;
    }
    else if(income<20000)
    {
        tax=income*0.1;
    }
    else if(income<30000)
    {
        tax=income*0.15;
    }
    else if(income<50000)
    {
        tax=income*0.2;
    }
    else
    {
        tax=income*0.25;
    }
    printf("Tax amount=%f",tax);
    getch();
    return 0;
}
```

6. WAP to find whether the entered year is leap year or not.
7. WAP to read a character from the keyboard and convert it into uppercase if it is in lowercase and vice versa.

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int year;
    printf("Enter the year\n");
    scanf("%d",&year);
    if (((year % 4 == 0) && (year % 100!= 0)) || (year%400 == 0))
        printf("%d is a leap year", year);
    else
        printf("%d is not a leap year", year);
    getch();
    return 0;
}
```

```
#include<stdio.h>
#include<conio.h>
int main()
{
    char ch;
    printf("Enter the character\n");
    scanf("%c",&ch);
    if(ch>=65&& ch<=90)
    {
        printf("The character is uppercase\n");
        ch=ch+32;
        printf("The equivalent lowercase character is %c",ch);
    }
    else if(ch>=97&&ch<=122)
    {
        printf("The character is lowercase\n");
        ch=ch-32;
        printf("The equivalent uppercase character is %c",ch);
    }
    else
    {
        printf("Invalid Input\n");
    }
    getch();
    return 0;
}
```

8. WAP to read a temperature in of a day in Fahrenheit to print
 "Nice day" if temperature is greater than 60 but less than 80
 "Cold day" if temperature is 60 or lower
 "Hot day" if temperature is 80 or higher

```
#include<stdio.h>
#include<conio.h>
int main()
{
int temp;
printf("Enter the temperature in Fahrenheit\n");
scanf("%d",&temp);
if(temp<=60)
{
printf("Cold Day");
}
else if(temp<80)
{
printf("Nice day");
}
else
{
printf("Hot day");
}
getch();
return 0;
}
```

9. WAP to input a character and determine whether it is uppercase, lowercase , digits or special symbols.

```
#include<stdio.h>
#include<conio.h>
int main()
{
char ch;
printf("Enter the character\n");
scanf("%c",&ch);
if(ch>=97&&ch<=122)
{
printf("You Entered lowercase character");
}
else if(ch>=65&&ch<=90)
{
printf("You Entered Uppercase character");
}
else if(ch>=48&&ch<=57)
{
printf("You Entered Digit");
}
else
{
printf("You Entered special symbol");
}
getch();
return 0;
}
```

10. An organization is dealing with two items say A and B and provides the commission on sale of these items according to the following policies

- i) Commission rate for item A is 5% upto a sale of Rs.2000.If the sale of item is above 2000 then the commission rate is 6% on extra sale
- ii)For B 10% upto the sale of Rs.4000.If the sale is above 4000 commission rate is 12% on extra sale.

Given the sales of both items, Write a Program to compute the net commission.

```
#include<stdio.h>
#include<conio.h>
int main()
{
    float sa,sb,ca,cb,nc;
    printf("Enter the sales of a and b\n");
    scanf("%f%f",&sa,&sb);
    if(sa<=2000)
    {
        ca=sa*0.05;
    }
    else
    {
        ca=2000*0.05+(sa-2000)*0.06;
    }
    if(sb<=4000)
    {
        cb=sb*0.1;
    }
    else
    {
        cb=4000*0.1+(sa-4000)*0.12;
    }
    nc=ca+cb;
    printf("The net commission of these items is %f",nc);
    getch();
    return 0;
}
```

11. Write a program to calculate the income tax and net salary having following condition

If salary \geq 9000 income tax is 40% of salary

If salary \geq 7500 and salary \leq 8999 then income tax is 30% of salary

If salary \leq 7499 then income tax is 20 % of salary

```
#include<stdio.h>
#include<conio.h>
int main()
{
float salary,netsalary,tax;
printf("Enter the salary\n");
scanf("%f",&salary);
if(salary>=9000)
{
tax=salary*0.4;
}
else if(salary>=7500)
{
tax=salary*0.3;
}
else
{
tax=salary*0.2;
}
netsalary=salary-tax;
printf("Income tax=%f\n",tax);
printf("Net salary=%f",netsalary);
getch();
return 0;
}
```

12. A bank has introduced an incentive policy. A bonus of 2% of the balance is given to everyone, irrespective of their balance and 5% is given to female account holder if their balances and 5% is given to female account holder if their balance is more than Rs.5000/.Write a program to represent this policy and calculate balance after bonus.

```
#include<stdio.h>
#include<conio.h>
int main()
{
float balance;
char gender;
printf("Enter the balance\n");
scanf("%f",&balance);
printf("Enter the gender\n");
fflush(stdin);
scanf("%c",&gender);
if(gender=='f' | | gender=='F')
{
if(balance>5000)
{
balance=balance+0.05*balance;
}
else
{
balance=balance+0.02*balance;
}
}
else
{
balance=balance+0.02*balance;
}
printf("\nThe balance after bonus is %f",balance);
getch();
return 0;
}
```

13. Write a program and draw a flowchart to read a positive integer value and compute the following sequence .If the number is even, half it ,if it is odd ,multiply 3 and add 1 print the result. If the input value is less than 1, print a message containing word “ERROR”.

```

#include<stdio.h>
#include<conio.h>
int main()
{
    int num;
    printf("Enter the integer number\n");
    scanf("%d",&num);
    if(num<1)
    {
        printf("ERROR");
    }
    else
    {
        if(num%2==0)
        {
            num=num/2;
        }
        else
        {
            num=num*3+1;
        }
        printf("Number after result=%d",num);
    }
    getch();
    return 0;
}

```

- 14. Write a program to find the numbers and sum of all numbers greater than 100 and less than 200 that are divisible by 5.**

```

#include<stdio.h>
#include<conio.h>
int main()
{
    int i,sum;
    for(i=101;i<200;i++)
    {
        if(i%5==0)
        {
            printf("%d\t",i);
            sum=sum+i;
        }
    }
    printf("\nThe sum of numbers =%d",sum);
    getch();
    return 0;
}

```

15. Write a program which asks time in seconds and convert it into hour minutes and seconds.

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int sec, h, m, s;
    printf("Input seconds: ");
    scanf("%d", &sec);
    h = (sec/3600);
    m = (sec -(3600*h))/60;
    s = (sec -(3600*h)-(m*60));
    printf("H:M:S - %d:%d:%d\n",h,m,s);
    getch();
    return 0;
}
```

Programs related to Loop control Instructions

1. WAP to find reverse of a given number.

2. WAP find sum of digits of a given number.

[PU:2016 spring]

```
#include<stdio.h>
#include<conio.h>
int main()
{
int num,rem,rev=0;
printf("Enter the number\n");
scanf("%d",&num);
while(num!=0)
{
rem=num%10;
rev=rev*10+rem;
num=num/10;
}
printf("Reverse of number=%d",rev);
getch();
return 0;
}
```

```
#include<stdio.h>
#include<conio.h>
int main()
{
int num,rem,sum=0;
printf("Enter the number\n");
scanf("%d",&num);
while(num!=0)
{
rem=num%10;
sum=sum+rem;
num=num/10;
}
printf("sum of digits of number= %d",sum);
getch();
return 0;
}
```

3. WAP to check the given number is palindrome or not.

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int num,rem,rev=0,a;
    printf("Enter the number\n");
    scanf("%d",&num);
    a=num;
    while(num!=0)
    {
        rem=num%10;
        rev=rev*10+rem;
        num=num/10;
    }
    if(a==rev)
    {
        printf("Entered number is palindrome");
    }
    else
    {
        printf("Entered number is not palindrome");
    }
    getch();
    return 0;
}
```

4. WAP to Check the given number is Armstrong number or not

```
#include <stdio.h>
#include<conio.h>
#include <math.h>
int main()
{
    int num,orgnum,rem,sum=0,n=0;
    printf("Enter the number: ");
    scanf("%d", &orgnum);
    num=orgnum;
    while (num != 0)
    {
        num =num/10;
        ++n;
    }
    num=orgnum;
    while (num != 0)
    {
        rem = num%10;
        sum = sum+pow(rem, n);
        num=num/10;
    }
    if(sum == orgnum)
        printf("Given number is armstrong number");
    else
        printf("Given number is not an Armstrong number");
    getch();
    return 0;
}
```

5. WAP to find the sum of n natural number.

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int n,i,sum=0;
    printf("Enter the value of n\n");
    scanf("%d",&n);
    for(i=1;i<=n;i++)
    {
        sum=sum+i;
    }
    printf("The sum natural number is %d",sum);
    getch();
    return 0;
}
```

6. WAP to check the given number is prime or not

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int main()
{
    int i,num;
    printf("Enter the number\n");
    scanf("%d",&num);
    if (num <= 1)
    {
        printf("Entered number is not a prime
number\n");
        exit(1);
    }
    for(i=2;i<num;i++)
    {
        if(num%i==0)
        {
            printf("Entered number is not prime
number");
            break;
        }
        if(num==i)
        {
            printf("Entered number is a prime number");
        }
    }
    getch();
    return 0;
}
```

7. WAP to check the given number is perfect or not

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int num,i,sum=0,rem;
    printf("Enter a number: \n");
    scanf("%d",&num);
    for(i=1;i<num;i++)
    {
        if (num % i == 0)
        {
            sum=sum+i;
        }
    }
    if(sum==num)
    {
        printf("The given number is perfect
number");
    }
    else
    {
        printf("The given number is not perfect
number");
    }
    getch();
    return 0;
}
```

8. WAP to check the given number is strong or not.

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int num,i,fact,rem,sum=0,temp;
    printf("Enter a number: ");
    scanf("%d",&num);
    temp=num;
    while(num!=0)
    {
        i=1,fact=1;
        rem=num%10;
        while(i<=rem)
        {
            fact=fact*i;
            i++;
        }
        sum=sum+fact;
        num=num/10;
    }
    if(sum==temp)
        printf("%d is a strong number",temp);
    else
        printf("%d is not a strong number",temp);
    getch();
    return 0;
}
```

9. Write a stated program.

Until the user presses “Y”, read marks of student of 3 subjects and display their total percentage.

```
#include<stdio.h>
#include<conio.h>
int main()
{
char choice;
float m1,m2,m3,total,per;
do
{
printf("\nEnter the marks in 3 subjects\n");
scanf("%f%f%f",&m1,&m2,&m3);
total=m1+m2+m3;
per=(total/3);
printf("percentage=%f",per);
printf("\nDo you want to continue\n");
fflush(stdin);
scanf("%c",&choice) ;
system("cls");
}while(choice=='y' | | choice=='Y');
getch();
return 0;
}
```

10. WAP to count all the integers greater than 100 and less than 200 that are divisible by 5 . Also find the sum of these numbers.

```
#include<stdio.h>
#include<conio.h>
int main()
{
int sum=0,count=0,i;
for(i=101;i<200;i++)
{
if(i%5==0)
{
sum=sum+i;
count =count+1;
}
}
printf("The sum is %d\n",sum);
printf("Number of integers greater than 100 and less than 200 that are divisible by 5 are %d",count);
getch();
return 0;
}
```

11. WAP to find sum of even numbers from 1 to 100 .

```
#include<stdio.h>
#include<conio.h>
int main()
{
int i,sum=0;
for(i=1;i<=100;i++)
{
if(i%2==0)
{
sum=sum+i;
}
printf("%d\t",sum);
getch();
return 0;
}
```

12. WAP to find sum of even and odd numbers from n1 to n2.

```
#include<stdio.h>
#include<conio.h>
int main()
{
int n1,n2,esum=0,osum=0,i;
printf("Enter the values of n1 and n2\n");
scanf("%d%d",&n1,&n2);
for(i=n1;i<=n2;i++)
{ if(i%2==0)
{
esum=esum+i;
}
else
{
osum=osum+i;
}
}
printf("sum of even number is %d\n",esum);
printf("sum of odd number is %d",osum);
getch();
return 0;
}
```

13. WAP to find factorial of a given number

```
#include<stdio.h>
#include<conio.h>
int main()
{
int num,i,fact = 1;
printf("Enter a number \n");
scanf("%d", &num);
for (i= 1; i <= num; i++)
{
fact = fact * i;
}
printf("Factorial of given number= %d \n", fact);
getch();
return 0;
}
```

14. Write a program to enter numbers until user press zero(0) and find the sum of supplied numbers.

```
#include<stdio.h>
#include<conio.h>
int main()
{
int sum=0,num;
do
{
printf("Enter number");
scanf("%d",&num);
sum=sum+num;
}while(num!=0);
printf("Total sum of Entered number=%d",sum);
getch();
return 0;
}
```

16. WAP to generate Fibonacci series upto nth term. OR
WAP to generate Fibonacci number as per user choice[PU:2018 spring]

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int i,n, a,b,c;
    a=0;
    b=1;
    printf("\nEnter the value of n\n");
    scanf("%d",&n);
    printf("%d\t%d\t",a,b);
    for (i= 1; i <= n-2; i++)
    {
        c=a+b;
        printf("%d\t",c);
        a=b;
        b=c;
    }
    getch();
    return 0;
}
```

17. WAP to print prime number from 1 to 200. [PU:2017 spring]

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int i,j;
    for(i=1;i<=200;i++)
    {
        for(j=2;j<i;j++)
        {
            if(i% j==0)
            {
                break;
            }
            if(i==j)
            {
                printf("%d\n",i);
            }
        }
        getch();
        return 0;
    }
}
```

16. WAP to generate the Fibonacci series upto the nth term when initial values are given by user

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int i,n, a,b,c;
    printf("Enter the values of a and b\n");
    scanf("%d%d",&a,&b);
    printf("\nEnter the value of n\n");
    scanf("%d",&n);
    printf("%d\t%d\t",a,b);
    for (i= 1; i <= n-2; i++)
    {
        c=a+b;
        printf("%d\t",c);
        a=b;
        b=c;
    }
    getch();
    return 0;
}
```

18. WAP to convert decimal number to binary number

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int num,i,base=1,sum=0,rem;
    printf("Enter the number\n");
    scanf("%d",&num);
    i=num;
    while(num!=0)
    {
        rem=num%2;
        sum=sum+rem*base;
        base=base*10;
        num=num/2;
    }
    printf("Binary equivalent of %d is %d",i,sum);
}
```

19. WAP to print all 3 digits Armstrong numbers (Armstrong numbers from 100 to 999)

```
#include<stdio.h>
#include<conio.h>
int main()
{
int i,rem,sum,num;
for(i=100;i<=999;i++)
{
num=i;
sum=0;
while(num!=0)
{
rem=num%10;
sum=sum+rem*rem*rem;
num=num/10;
}
if(sum==i)
{
printf("%d\t",sum);
}
}
getch();
return 0;
}
```

20. WAP to display multiplication table of a given number.

```
#include<stdio.h>
#include<conio.h>
int main()
{
int num,i,pro;
printf("Enter the number\n");
scanf("%d",&num);
for(i=1;i<=10;i++)
{
    pro=num*i;
    printf("%d * %d = %d",num,i,pro);
    printf("\n");
}
getch();
return 0;
}
```

Programs related to case control Instructions (switch-case)

- 1) WAP to perform the Arithmetic Operation Using Switch.

```
#include<stdio.h>
#include<conio.h>
int main()
{
float a,b,result;
int choice;
printf("Menu\n");
printf("1.Addition\n");
printf("2.Substration\n");
printf("3.Multiplication\n");
printf("4.Division\n");
printf("Enter the two numbers\n");
scanf("%f%f",&a,&b);
printf("Enter your choice\n");
scanf("%d",&choice);
switch(choice)
{
case 1:
result=a+b;
printf("Addition=%f",result) ;
break;
case 2:
result=a-b;
printf("Subtraction=%f",result);
break;
case 3:
result=a*b;
printf("Multiplication=%f",result);
break;
case 4:
result=a/b;
printf("Division=%f",result);
break;
default:
printf("Invalid choice!");
break;
}
getch();
return 0;
}
```

2. WAP to perform various Arithmetic operation on the basics of operators given by the user.

3.

```
#include<stdio.h>
#include<conio.h>
int main()
{
float a,b,result;
char opr;
printf("Menu\n");
printf("1.Addition\n");
printf("2.Substration\n");
printf("3.Multiplication\n");
printf("4.Division\n");
printf("Enter the two numbers\n");
scanf("%f%f",&a,&b);
printf("Enter the operator among +,-,/and* \n");
fflush(stdin);
scanf("%c",&opr);
switch(opr)
{
case '+':
result=a+b;
printf("Addition=%f",result) ;
break;
case '-':
result=a-b;
printf("Subtraction=%f",result);
break;
case '*':
result=a*b;
printf("Multiplication=%f",result);
break;
case '/':
result=a/b;
printf("Division=%f",result);
break;
default:
printf("Invalid choice!");
}
getch();
return 0;
}
```

3) WAP to display the following menu and perform the following operations.

1. Find the simple interest
2. Convert degree Celsius to Fahrenheit
3. Convert character into ASCII code
4. Find the area of circle
5. Exit from the program

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int main()
{
    int choice;
    float p,t,r,i,c,f,area;
    char ch;
    printf("Menu\n");
    printf("1.Find simple interest\n");
    printf("2.convert Celsius of Fahrenheit\n");
    printf("3.convert character to ASCII code\n");
    printf("4.Find area of circle\n");
    printf("5.Exit from program\n");
    printf("Enter your choice\n");
    scanf("%d",&choice);
    switch(choice)
    {
        case 1:
            printf("Enter Principal,Time and Rate\n");
            scanf("%f%f%f",&p,&t,&r);
            i=(p*t*r)/100;
            printf("simple intrest is %f",i);
            break;

        case 2:
            printf("Enter the temperature in celcius\n");
            scanf("%f",&c);
            f=1.8*c+32;
            printf("Converted temp is %f",f);
            break;

        case 3:
            printf("Enter a character\n");
            scanf("%c",&ch);
            printf("The corresponding ASCII code is %d",ch);
            break;
    }
}
```

```

case 4:
printf("Enter the radius of circle\n");
scanf("%f",&r);
area=3.14*r*r;
printf("Area=%f",area);
break;
case 5:
exit(0);
default:
printf("Wrong choice!");
}
getch();
}

```

4) Write a menu program to satisfy the following function

1. Check whether the given string is palindrome or not
2. Find all the prime numbers from 100 to 200
3. Display all ASCII characters from 0 to 255.
4. Exit from the program

```

#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<stdlib.h>
int main()
{
char str1[20],str2[20];
int choice,i,j;
printf("Menu\n");
printf("1.Check whether the given string is palindrome or not\n");
printf("2.Find all prime numbers from 100 to 200\n");
printf("3.Display all ascii characters from 0 to 255\n");
printf("4.Exit from program\n");
printf("Enter your choice\n");
scanf("%d",&choice);

```

```

switch(choice)
{
case 1:
printf("Enter the string\n");
gets(str1);
strcpy(str2,str1);
strrev(str2);
if(strcmp(str1,str2)==0)
{
printf("String is palindrome");
}
else
{
printf("String is not palindrome");
}
break;
case 2:
for(i=100;i<=200;i++)
{
    for(j=2;j<i;j++)
    {
        if(i% j==0)
        {
            break;
        }
        if(i==j)
        printf("%d\t",i);
    }
}
break;
case 3:
for(i=0; i<=255; i++)
{
printf("ASCII value of character %c = %d\n", i, i);
}
break;
case 4:
exit(0);
break;
default:
printf("Wrong choice!");
}
getch();
return 0;
}

```

5) Write a menu driven program to work following cases, take appropriate input whenever required.

1. Reverse a number
2. Find sum of individual unit
3. Check for prime
4. Exit [PU: 2018 fall]

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int main()
{
int choice,i,rem,sum=0,rev=0,num;
printf("Menu\n");
printf("1.Reverse a number\n");
printf("2.Find sum of individual unit\n");
printf("3.Check for prime\n");
printf("4.Exit from program\n");
printf("Enter the number\n");
scanf("%d",&num);
printf("Enter your choice\n");
scanf("%d",&choice);
switch(choice)
{
case 1:
while(num!=0)
{
rem=num%10;
rev=rev*10+rem;
num=num/10;
}
printf("Reverse of number=%d",rev);
break;
case 2:
while(num!=0)
{
rem=num%10;
sum=sum+rem;
num=num/10;
}
printf("The sum of digits of number=%d",sum);
break;
```

```

case 3:
if (num <= 1)
{
printf("Entered number is not a prime number\n");
}
for(i=2;i<num;i++)
{
    if(num%i==0)
    {
        printf("Entered number is not prime number");
        break;
    }
}
if(num==i)
{
printf("Entered number is a prime number");
}
break;
case 4:
exit(0);
break;
default:
printf("Wrong choice!");
}
getch();
return 0;
}

```

6) Write a program to display the following menu

1. Conversion of ASCII code to char
2. To find the sum of n natural numbers
3. Exit from the program

and perform task as per users choice repeatedly until his/her choice is to exit

[PU:2013 spring]

```

#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int main()
{
int choice,n,sum=0,i,code;
while(1)
{
printf("\nMenu\n");
printf("1.Conversion of ASCII code to char\n");
printf("2.Find sum of n natural numbers\n");
printf("3.Exit from program\n");
printf("Enter your choice\n");
scanf("%d",&choice);

```

```
switch(choice)
{
case 1:
printf("Enter the ASCII code\n");
scanf("%d",&code);
printf("character corresponding to %d is %c",code,code);
break;

case 2:
printf("Enter the value of n\n");
scanf("%d",&n);
for(i=1;i<=n;i++)
{
sum=sum+i;
}
printf("sum=%d",sum);
break;

case 3:
exit(0);
break;

default:
printf("Wrong choice!");
}
}
getch();
return 0;
}
```

Programs to print given pattern

WAP to print the following pattern.

1) 1
 1 2
 1 2 3

[PU:2017 fall]

```
#include<stdio.h>
#include<conio.h>
int main()
{
int i,j;
for(i=1;i<=3;i++)
{
    for(j=1;j<=i;j++)
    {
        printf("%d\t",j);
    }
    printf("\n");
}
getch();
return 0;
}
```

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

```
#include<stdio.h>
#include<conio.h>
int main()
{
int i,j;
for(i=1;i<=5;i++)
{
    for(j=1;j<=i;j++)
    {
        printf("%d\t",j);
    }
    printf("\n");
}
getch();
return 0;
}
```

3) **1**
2 **2**
3 **3** **3**
4 **4** **4** **4**
5 **5** **5** **5** **5**

```
#include<stdio.h>
#include<conio.h>
int main()
{
int i,j;
for(i=1;i<=5;i++)
{
    for(j=1;j<=i;j++)
    {
        printf("%d\t",i);
    }
    printf("\n");
}
getch();
return 0;
}
```

4) **1**
1 **1**
1 **1** **1**
1 **1** **1** **1**
1 **1** **1** **1** **1**

```
#include<stdio.h>
#include<conio.h>
int main()
{
int i,j;
for(i=1;i<=5;i++)
{
    for(j=1;j<=i;j++)
    {
        printf("1\t");
    }
    printf("\n");
}
getch();
return 0;
}
```

5) * * * * *
* * * *
* * *
* *
*

```
#include<stdio.h>
#include<conio.h>
int main()
{
int i,j;
for(i=5;i>=1;i--)
{
    for(j=1;j<=i;j++)
    {
        printf("*\t");
    }
    printf("\n");
}
getch();
return 0;
}
```

6) **5** **5** **5** **5** **5**
4 **4** **4** **4**
3 **3** **3**
2 **2**
1

```
#include<stdio.h>
#include<conio.h>
int main()
{
int i,j;
for(i=5;i>=1;i--)
{
    for(j=1;j<=i;j++)
    {
        printf("%d\t",i);
    }
    printf("\n");
}
getch();
return 0;
}
```

7) 1 2 3 4 5
 1 2 3 4
 1 2 3
 1 2
 1

8) 1 1 1 1 1
 1 1 1 1
 1 1 1
 1 1
 1

[PU:2019 fall]

```
#include<stdio.h>
#include<conio.h>
int main()
{
int i,j;
for(i=5;i>=1;i--)
{
    for(j=1;j<=i;j++)
    {
        printf("%d\t",j);
    }
    printf("\n");
}
getch();
return 0;
}
```

```
#include<stdio.h>
#include<conio.h>
int main()
{
int i,j;
for(i=5;i>=1;i--)
{
    for(j=1;j<=i;j++)
    {
        printf("1\t");
    }
    printf("\n");
}
getch();
return 0;
}
```

9) 5 4 3 2 1
 5 4 3 2
 5 4 3
 5 4
 5

10) 1
 2 3
 4 5 6
 7 8 9 10
 11 12 13 14 15

```
#include<stdio.h>
#include<conio.h>
int main()
{
int i,j;
for(i=1;i<=5;i++)
{
    for(j=5;j>=i;j--)
    {
        printf("%d\t",j);
    }
    printf("\n");
}
getch();
return 0;
}
```

```
#include<stdio.h>
#include<conio.h>
int main()
{
int i,j,a=1;
for(i=1;i<=5;i++)
{
    for(j=1;j<=i;j++)
    {
        printf("%d\t",a);
        a++;
    }
    printf("\n");
}
getch();
return 0;
}
```

11)

1	11	21		
31	41	51		
61	71	81	91	
101	111	121	131	141

```
#include<stdio.h>
#include<conio.h>
int main()
{
int i,j,k=1;
for(i=1;i<=5;i++)
{
    for(j=1;j<=i;j++)
    {
        printf("%d\t",k);
        k=k+10;
    }
    printf("\n");
}
getch();
return 0;
}
```

12)

1	2	3	4	5
2	4	6	8	10
3	6	9	12	15
4	8	12	16	20
5	10	15	20	25

```
#include<stdio.h>
#include<conio.h>
int main()
{
int i,j;
for(i=1;i<=5;i++)
{
    for(j=1;j<=5;j++)
    {
        printf("%d\t",i*j);
    }
    printf("\n");
}
getch();
return 0;
}
```

13)

*				
*	*			
*	*	*		
*	*	*	*	
*	*	*	*	*

```
#include<stdio.h>
int main()
{
int i,j,k;
for(i=1;i<=5;i++)
{
    for(k=4;k>=i;k--)
    {
        printf("\t");
    }
    for(j=1;j<=i;j++)
    {
        printf("*\t");
    }
    printf("\n");
}
return 0;
}
```

14)

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

```
#include<stdio.h>
int main()
{
int i,j,k;
for(i=1;i<=5;i++)
{
    for(k=4;k>=i;k--)
    {
        printf("\t");
    }
    for(j=1;j<=i;j++)
    {
        printf("%d\t",i);
    }
    printf("\n");
}
return 0;
}
```

15)

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

16)

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

```
#include<stdio.h>
int main()
{
    int i,j,k;
    for(i=1;i<=5;i++)
    {
        for(k=4;k>=i;k--)
        {
            printf("\t");
        }
        for(j=1;j<=i;j++)
        {
            printf("%d\t",j);
        }
        printf("\n");
    }
    return 0;
}
```

```
#include<stdio.h>
int main()
{
    int i, j;
    for(i=0;i<5;i++)
    {
        for(j=1;j<=5-i;j++)
        {
            printf("\t%d",j);
        }
        for(j=4-i;j>=1;j--)
        {
            printf("\t%d",j);
        }
        printf("\n");
        for(j=0;j<=i;j++)
        {
            printf("\t");
        }
    }
    return 0;
}
```

Find the output of the following program.(Assume necessary header files)

```
void main()
{
    int i,j,k,c;
    i=0,j=1;
    for(k=0;k<5;k++)
    {
        i--;
        j++;
        c=i+j;
    }
    printf("value of i=%d,j=%d and c=%d",i,j,c);
    getch();
    return 0;
}
```

Solution:

Tracing the above program,

k	Condition	i=0	j=1	c
	(k<5)	i - -	j++	c=i+j
0	0<5 (T)	-1	2	c= -1+2=1
1	1<5 (T)	-2	3	c=-2+3=1
2	2<5 (T)	-3	4	c=-3+4=1
3	3<5 (T)	-4	5	c=-4+5=1
4	4<5 (T)	-5	6	c=-5+6=1
5	5<5 (F) Now Loop Terminates			

Output: i=-5, j=6 and c=1

Assignment:

Find the output of the following program.(Assume necessary header files)

```
int main()
{
int i,j;
for(i=0;i<=20;i++)
{
printf("%d\n",i++);
}
return 0;
}
```

(Trace the program yourself)

Output:

```
0
2
4
6
8
10
12
14
16
18
20
```

Programs related to series

WAP to find the sum of series.

$$1^2 + 2^2 + 3^2 + 4^2 + 5^2 + 6^2 + \dots + n^2$$

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int i,sum=0,n;
    printf("Enter the value of n");
    scanf("%d",&n);
    for(i=1;i<=n;i++)
    {
        sum=sum+(i*i);
    }
    printf("sum=%d",sum);
    getch();
    return 0;
}
```

What will be the sum of the given series.

$$1 + \frac{1}{x} + \frac{1}{x^2} + \frac{1}{x^3} + \frac{1}{x^4} + \dots + \frac{1}{x^n}$$

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
int main()
{
    int n,i;
    float x,sum=0;
    printf("Enter the value of x");
    scanf("%f",&x);
    printf("Enter the number of terms");
    scanf("%d",&n);
    for(i=0;i<=n;i++)
    {
        sum=sum+(1/pow(x,i));
    }
    printf("sum=%f",sum);
    getch();
    return 0;
}
```

What will be the sum of the given series.

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

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
int main()
{
    int n,i,j,fact;
    float x,sum=1;
    printf("Enter the value of x");
    scanf("%f",&x);
    printf("Enter the value of n");
    scanf("%d",&n);
    for(i=1;i<=n;i++)
    {
        fact=1;
        for(j=1;j<=i;j++)
        {
            fact=fact*j;
        }
        sum=sum+pow(x,i)/(float)fact;
    }
    printf("sum=%f",sum);
    getch();
    return 0;
}
```

Write a program to generate the following series and print the sum.

1x4,2x7, 3x10 n terms

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int i,j,n,sum=0;
    printf("Enter the number of term");
    scanf("%d",&n);
    for(i=1,j=4;i<=n;i++,j+=3)
    {
        printf("%dx%d\t",i,j);
        sum=sum+(i*j);
    }
    printf("\nsum of series=%d",sum);
    getch();
    return 0;
}
```

Write a program to read x, n and generate the following series and print sum.

$x+2/x^2 -3/x^3 +4/x^4 -5/x^5 \dots\dots\dots n$ terms.

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
int main()
{
    int i,x,n;
    float a,sum=0;
    printf("Enter the value of x");
    scanf("%d",&x);
    printf("Enter the value of n");
    scanf("%d",&n);
    if(n==0)
    {
        printf("There is no term");
    }
    if(n==1)
    {
        printf("The term is %d",x);
    }
    if(n>=2)
    {
        printf("The term is \t%d",x);
        for(i=2;i<=n;i++)
        {
            a=(pow(-1,i)*i)/pow(x,i);
            printf("\t%f",a);
            sum=sum+a;
        }
        printf("\nThe sum of terms=%f",sum+x);
    }
    getch();
    return 0;
}
```

WAP to compute the given series.

$\text{Sin}(x) = x - x^3/3! + x^5/5! - x^7/7! + x^9/9! \dots\dots$

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int i,n,sign=1;
    float x, nume, deno, sum=0, val, term;
    printf(" Enter the value for x in degree:");
    scanf("%f",&x);
```

```

printf(" Enter the value for n :");
scanf("%d",&n);
val=x;
x=x*3.14159/180;
nume=x;
deno=1;
sum=x;
for(i=2;i<=n;i++)
{
    nume=nume*x*x;
    deno=deno*(i*2-2)*(i*2-1);
    sign=sign*(-1);
    term=nume/deno*sign;
    sum=sum+term;
}
printf("\nsin(%f)=%f",val,sum);
getch();
return 0;
}

```

WAP to compute the given series.
Cos(x)= 1- x²/2! + x⁴/4! - x⁶/6!

```

#include<stdio.h>
#include<conio.h>
int main()
{
    int i,n,sign=1;
    float x,nume,deno,sum=0,val,term;
    printf(" Enter the value for x in degree:");
    scanf("%f",&x);
    printf(" Enter the value for n :");
    scanf("%d",&n);
    val=x;
    x=x*3.14159/180;
    nume=1;
    deno=1;
    sum=1;
    for(i=2;i<=n;i++)
    {
        nume=nume*x*x;
        deno=deno*(i*2-3)*(i*2-2);
        sign=sign*(-1);
        term=nume/deno*sign;
        sum=sum+term;
    }
    printf("\ncos(%f)=%f",val,sum);
    getch();
    return 0;
}

```

CHAPTER 5

Array and Strings

Array

What is array? [PU: 2012 fall, 2013 spring, 2015 spring, 2016 spring, 2017 spring]

Array is a set of similar data elements which are stored in consecutive memory location under a common variable name.

So to be an array,

- All elements must be of same data type
- All elements are stored in consecutive memory location.

Eg. If we want to store 100 integers in a sequence, we can create array for it.

```
int num[100];
```

Note: *The size and type of array cannot be changed after its declaration.*

Why do we use array in programming language? [2013 spring]

If we want many elements of similar type than it is not feasible to declare all variables and also manipulate these elements. So in this case we use array.

For example, if we want 100 integer variables, then instead of writing all 100 variables, we use array like int num[100]. Here 100 integer elements num[0], num[1], num[2]. num[99] are declared.

Limitation of Array

Memory allocation in array are static in nature. Which means memory is allocated before the execution of program begins (During compilation). In this type of allocation the memory cannot be resized after initial allocation. So it has some limitations.

- Wastage of memory
- Overflow of memory

eg. int num[100];

Here, the size of an array has been fixed to 100. If we just enter to 10 elements only, then there will be wastage of 90 memory location and if we need to store more than 100 elements there will be memory overflow.

Why array is called static data type? [PU: 2016 fall]

The array is called static data type because when the size of array is once allocated ,it cannot be modified.So that maximum size needed must known in advance.

eg. int num[100];

Here, the size of an array has been fixed to 100 and cannot be change during program execution.

Types of array

1. One dimensional Array
2. Multidimensional Array

One dimensional array

Elements of an array can be represented either as a single row or single column.

There is a single subscript or index whose value refers to the individual array element, which ranges from 0 to n-1, where n is the size of array.

Eg. int num[5]={5,10,15,20,25};

Assuming base address 2000, now array elements can be illustrated as:

Index of array	num[0]	num[1]	num[2]	num[3]	num[4]
Address	2000	2002	2004	2006	2008
value	5	10	15	20	25

Declaration:

data_type array_name[size];

eg. int num[10];

Here, int is a data type and num is the name of the array and 10 is the size of the array. It means num can only contain 10 elements of int type.

Initialization of 1-D array

- How can you initialize an one dimensional array? [PU:2017 fall]
- How can you initialize one dimensional array at compile time and run time? Explain with suitable example. [PU:2016 spring]

After the array is declared it must be initialized, otherwise it will contain garbage value (any random value). An array can be initialized at either compile time or runtime.

1) Compile time initialization

General form:

```
data_type array_name[size]={list of values};
```

The value in the list are separated by commas.

Example:

```
int num[5]={10,20,30,40,50};      //integer array initialization  
float area[5]={33.4,55.6,88.9,77.8,5.5}; //float array initialization
```

Note: It is also possible to initialize array without defining its size.

```
int num[]={67,87,56,24,77};
```

In this case compiler determines the size of array by calculating the number of elements in an array.

Accessing Array Elements

We can access the elements of an array by giving the name and proper subscript inside the bracket.

Eg. int num[5]={67,87,56,24,77};

Assuming base address 2000, now array elements can be illustrated as:

Index of array	num[0]	num[1]	num[2]	num[3]	num[4]
	67	87	56	24	77
Address	2000	2002	2004	2006	2008

Here, num[0] is 67 which is stored at 2000.

Similarly,

num[1]=87,

num[2]=56,

num[3]=24,

num[4]=77

Example

```
#include<stdio.h>
#include<conio.h>
int main()
{
int i;
int num[5]={67,87,56,24,77};
for(i=0;i<5;i++)
{
printf("\t%d",num[i]);
}
getch();
return 0;
}
```

Output

```
67    87    56    24    77
```

2) Runtime initialization

An array can also be initialized at runtime using `scanf()` function. This Approach is used for initializing large arrays with user specified values.

```
#include<stdio.h>
#include<conio.h>
int main()
{
int num[5];
int i;
printf("\nEnter the array elements");
for(i=0;i<5;i++)
{
scanf("%d",&num[i]);
}
printf("\nArray elements are");
for(i=0;i<5;i++)
{
printf("\n%d",num[i]);
}
getch();
return 0;
}
```

Multidimensional Array

Multidimensional array are those who have two or more than two dimensions.

Multidimensional array are defined in the same manner as one dimensional arrays, except that separate pair of square brackets is required for each subscript.

Thus, two dimensional array will require two pair of square brackets, a three dimensional array will require three pairs of square brackets and so on.

General form:

```
datatype array_name[s1][s2][s3]....[sn]
```

where, si is the size of ith dimension.

Example:

```
int survey[3][5][12];
```

Here, survey is a three dimensional array declared to contain 180 integer type elements.

Two dimensional array

- A two dimensional array is a collection of similar data items, structured in two dimensions (referred to as rows and columns).
- It is also called array of arrays.
- It is required to manipulate the data in table format or in matrix format which contains rows and columns.

Declaration:

```
data_type array_name[row_size][column_size];
```

eg. int arr[3][3];

It creates two dimensional array to store 9 elements of integer type. There are 3 rows and 3 columns in array matrix.

	Column 1	Column 2	Column 3
Row1	arr[0][0]	arr[0][1]	arr[0][2]
Row2	arr[1][0]	arr[1][1]	arr[1][2]
Row 3	arr[2][0]	arr[2][1]	arr[2][2]

Initialization of 2-D array

1) Compile time initialization

Two dimensional array may be initialized by following their declaration with a list of values enclosed in braces.

Example: int arr[3][3]={ {2,4,6},{8,9,12},{15,16,18}};

These are equivalent to following assignments

arr[0][0]= 2	arr[0][1]= 4	arr[0][2]= 6
arr[1][0]= 8	arr[1][1]= 9	arr[1][2]= 12
arr[2][0]=15	arr[2][1]= 16	arr[2][2]=18

Similarly for,

int arr[2][3]={ {1,5,10},{1,10,15}};

int arr[3][4]={ {65,85,75,50},{67,65,45,75},{35,5,60,50}};

Note: When array is completely initialized with all values, explicitly, we need not specify the size of the first dimension.

That is the statement,

int arr[][3]={ {2,4,6},{8,9,12},{15,16,18}}; is permitted.

Example:

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int i,j;
    int arr[3][3]={ {12,14,16},{5,7,15},{15,25,45}};
    printf("The matrix is\n");
    for(i=0;i<3;i++)
    {
        for(j=0;j<3;j++)
        {
            printf("%d\t",arr[i][j]);
        }
        printf("\n");
    }
    getch();
    return 0;
}
```

2) Runtime initialization

Program to input 3*3 matrix and display it.

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int i,j;
    int arr[3][3];
    printf("Enter the elements of matrix\n");
    for(i=0;i<3;i++)
    {
        for(j=0;j<3;j++)
        {
            scanf("%d",&arr[i][j]);
        }
    }
    printf("Elements of matrix are\n");
    for(i=0;i<3;i++)
    {
        for(j=0;j<3;j++)
        {
            printf("%d\t",arr[i][j]);
        }
        printf("\n");
    }
    getch();
    return 0;
}
```

**Why array is important in programming? How can you initialize different types of arrays?
Explain 2-dimensional array in C.[PU: 2014 spring,2018 fall]**

If we want many elements of similar type than it is not feasible to declare all variables and also manipulate these elements. So in this case we use array.

For example, if we want 100 integer variables, then instead of writing all 100 variables, we use array like int num[100]. Here 100 integer elements num[0], num[1], num[2]. num[99] are declared. Due to these reason array is important in programming.

Initialization of 1-D array

1) Compile time Initialization

General form: data_type array_name[size]={list of values};

The value in the list are separated by commas.

Example:

```
int num[5]={10,20,30,40,50};      //integer array initialization
```

2) Runtime initialization

An array can also be initialized at runtime using scanf() function. This Approach is used for initializing large arrays with user specified values.

Example:

```
for(i=0;i<5;i++)
{
    scanf("%d",&num[i]);
}
```

This program statement initializes 5 arrays elements num[0],num[1],num[2],num[3],num[4] with user specified values.

Initialization of 2-D array

1) Compile time initialization

Two dimensional array may be initialized by following their declaration with a list of values enclosed in braces.

Example: int arr[3][3]={ {2,4,6},{8,9,12},{15,16,18}};

These are equivalent to following assignments

arr[0][0]= 2	arr[0][1]= 4	arr[0][2]= 6
arr[1][0]= 8	arr[1][1]= 9	arr[1][2]= 12
arr[2][0]=15	arr[2][1]= 16	arr[2][2]=18

2) Runtime initialization

Runtime initialization of 2-D array is done by using `scanf()` function with the concept of rows and columns.

Example

```
for(i=0;i<3;i++)  
{  
    for(j=0;j<3;j++)  
    {  
        scanf("%d",&arr[i][j]);  
    }  
}
```

This statement initializes 9 elements of matrix of size 3*3.

Two dimensional array

- A two dimensional array is a collection of similar data items, structured in two dimensions (referred to as rows and columns).
- It is also called array of arrays.
- It is required to manipulate the data in table format or in matrix format which contains rows and columns.

Declaration:

```
data_type array_name[row_size][column_size];
```

eg. `int arr[3][3];`

It creates two dimensional array to store 9 elements of integer type. There are 3 rows and 3 columns in array matrix.

	Column 1	Column 2	Column 3
Row1	arr[0][0]	arr[0][1]	arr[0][2]
Row2	arr[1][0]	arr[1][1]	arr[1][2]
Row 3	arr[2][0]	arr[2][1]	arr[2][2]

One Dimensional Array(Program solutions)

<p>1) WAP to input 10 number in an array and display it.</p> <pre>#include<stdio.h> #include<conio.h> int main() { int i,num[10]; printf("Enter 10 array elements\n"); for(i=0;i<10;i++) { scanf("%d",&num[i]); } printf("The array elements are\n"); for(i=0;i<10;i++) { printf("%d\n",num[i]); } getch(); return 0; }</pre>	<p>3) WAP to read n elements in array and display them in reverse order.</p> <pre>#include<stdio.h> #include<conio.h> int main() { int i,num[100],n; printf("Enter number of array elements \n"); scanf("%d",&n); printf("Enter %d array elements\n",n); for(i=0;i<n;i++) { scanf("%d",&num[i]); } printf("Array elements in reverse order are\n"); for(i=n-1;i>=0;i--) { printf("%d\n",num[i]); } getch(); return 0; }</pre>
<p>2) WAP to input n number in an array and display it.</p> <pre>#include<stdio.h> #include<conio.h> int main() { int i,num[100],n; printf("Enter the number of array elements\n"); scanf("%d",&n); printf("Enter %d array elements\n",n); for(i=0;i<n;i++) { scanf("%d",&num[i]); } printf("The array elements are\n"); for(i=0;i<n;i++) { printf("%d\n",num[i]); } getch(); return 0; }</pre>	<p>4) WAP to read n numbers using array and find their average.</p> <pre>#include<stdio.h> #include<conio.h> int main() { int n,i; float num[100],sum=0,avg; printf("Enter number of array elements \n"); scanf("%d",&n); printf("Enter %d elements\n",n); for(i=0;i<n;i++) { scanf("%f",&num[i]); } for(i=0;i<n;i++) { sum=sum+num[i]; } avg=sum/n; printf("Average=%f\n",avg); getch(); return 0; }</pre>

5) WAP to input n numbers in an array and find the sum of all even numbers and count them.	6) WAP to input n numbers in an array and find the sum of all even numbers and odd numbers and count their numbers and display them.
<pre>#include<stdio.h> #include<conio.h> int main() { int num[100],i,n,esum=0,ecount=0; printf("Enter number of array elements\n"); scanf("%d",&n); printf("Enter %d elements\n",n); for(i=0;i<n;i++) { scanf("%d",&num[i]); } for(i=0;i<n;i++) { if(num[i]%2==0) { esum=esum+num[i]; ecount++; } } printf("Sum of Even numbers=%d\n",esum); printf("Number of Even numbers=%d",ecount); getch(); return 0; }</pre>	<pre>#include<stdio.h> #include<conio.h> int main() { int num[100],i,n,esum=0,ecount=0,osum=0,octount=0; printf("Enter the number of array elements\n"); scanf("%d",&n); printf("Enter %d elements\n",n); for(i=0;i<n;i++) { scanf("%d",&num[i]); } for(i=0;i<n;i++) { if(num[i]%2==0) { esum=esum+num[i]; ecount++; } else { osum=osum+num[i]; octount++; } } printf("Sum of Even numbers =%d\n",esum); printf("Number of Even numbers=%d\n",ecount); printf("Sum of Odd numbers =%d\n",osum); printf("Number of Odd numbers=%d\n",octount); getch(); return 0; }</pre>

- 7) WAP to read seven days temperature in an array and print the deviations from average temperature for each day.

```
#include<stdio.h>
#include<conio.h>
int main()
{
    float temp[7],dev,sum=0,avg;
    int i;
    for(i=0;i<7;i++)
    {
        printf("Enter the temperature of Day %d \n",i+1);
        scanf("%f",&temp[i]);
        sum=sum+temp[i];
    }
    avg=sum/7;
    printf("\nThe Average temperature of days = %f",avg);
    for(i=0;i<7;i++)
    {
        dev=temp[i]-avg;
        printf("\nThe Deviation of temperature from day %d =%0.2f",i+1,dev);
    }
    getch();
}
```

- 8) WAP to read the marks of 10 students calculate and display the average marks and deviation of marks of each student from average marks.

```
#include<stdio.h>
#include<conio.h>
int main()
{
    float marks[10],avg,dev,sum=0;
    int i;
    printf("Enter the marks of 10 students\n");
    for(i=0;i<10;i++)
    {
        scanf("%f",&marks[i]);
        sum=sum+marks[i];
    }
    avg=sum/10;
    printf("The average marks is \t%f",avg);
    printf("\nThe deviation of each student from average\n");
    for(i=0;i<10;i++)
    {
        dev=marks[i]-avg;
        printf("\nmarks[%d]=%f\tdeviation=%f",i,marks[i],dev);
    }
    getch();
    return 0;
}
```

<p>9) WAP to read n numbers in an array find the largest number among them.</p>	<p>10) Write a program to read n number from keyboard and find the smallest and largest number using array.[PU:2014 fall]</p>
<pre>#include<stdio.h> #include<conio.h> int main() { int i,num[100],n,large; printf("Enter the number of array elements\n"); scanf("%d",&n); printf("Enter %d elements\n",n); for(i=0;i<n;i++) { scanf("%d",&num[i]); } large=num[0]; for(i=1;i<=n;i++) { if(num[i]>large) { large=num[i]; } } printf("Largest number=%d",large); getch(); return 0; }</pre>	<pre>#include<stdio.h> #include<conio.h> int main() { int i,num[100],n,large,small; printf("Enter the number of array elements "); scanf("%d",&n); printf("Enter %d elements\n",n); for(i=0;i<n;i++) { scanf("%d",&num[i]); } large=num[0]; small=num[0]; for(i=1;i<n;i++) { if(num[i]>large) { large=num[i]; } if(num[i]<small) { small=num[i]; } } printf("Largest number=%d",large); printf("\nSmallest number=%d",small); getch(); return 0; }</pre> <p>Assignment:</p> <p>Write a C program using array to find largest and smallest number from the list of 100 given numbers.[PU:2017 spring]</p>

- 11) WAP to read an array of length n. Display the total count for positive numbers, negative numbers and zeros separately.

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int i,num[100],n,pcount=0,ncount=0,zcount=0;
    printf("Enter the number of array elements\n");
    scanf("%d",&n);
    printf("Enter %d elements\n",n);
    for(i=0;i<n;i++)
    {
        scanf("%d",&num[i]);

    }
    for(i=0;i<n;i++)
    {
        if(num[i]>0)
        {
            pcount++;
        }
        else if(num[i]<0)
        {
            ncount++;
        }
        else
        {
            zcount++;
        }
    }

    printf("Number of positive numbers=%d",pcount);
    printf("\nNumber of Negative numbers=%d",ncount);
    printf("\nNumber of Zeros=%d",zcount);
    getch();
    return 0;
}
```

<p>12) WAP to read n numbers in an array and display the sum of even numbers only and product of odd numbers only.</p>	<p>13) WAP to read n numbers in an array and find the sum of even numbers and odd numbers and display the result.</p>
<pre>#include<stdio.h> #include<conio.h> int main() { int i,num[100],n,esum=0,opro=1; printf("Enter the number array of elements\n"); scanf("%d",&n); printf("Enter %d elements\n",n); for(i=0;i<n;i++) { scanf("%d",&num[i]); } for(i=0;i<n;i++) { if(num[i]%2==0) { esum=esum+num[i]; } else { opro=opro*num[i]; } } printf("sum of even numbers=%d",esum); printf("\nProduct of odd numbers=%d",opro); getch(); return 0; }</pre>	<pre>#include<stdio.h> #include<conio.h> int main() { int i,num[100],n,esum=0,osum=0; printf("Enter the number of array elements\n"); scanf("%d",&n); printf("Enter %d elements\n",n); for(i=0;i<n;i++) { scanf("%d",&num[i]); } for(i=0;i<n;i++) { if(num[i]%2==0) { esum=esum+num[i]; } else { osum=osum+num[i]; } } printf("sum of even numbers=%d",esum); printf("\nsum of odd numbers=%d",osum); getch(); return 0; }</pre>

14) WAP to input n elements in array and copy to another array.	15) WAP to copy the content of one array to another array in a reverse order.
<pre>#include<stdio.h> #include<conio.h> int main() { int a[100],b[100],i,n; printf("Enter number of array elements\n"); scanf("%d",&n); printf("Enter %d elements\n",n); for(i=0;i<n;i++) { scanf("%d",&a[i]); } printf("\nThe entered elements are:\n"); for(i=0;i<n;i++) { printf("%d\n",a[i]); } for(i=0;i<n;i++) { b[i]=a[i]; } printf("\nThe copied elements are\n"); for(i=0;i<n;i++) { printf("%d\n",b[i]); } getch(); return 0; }</pre>	<pre>#include<stdio.h> #include<conio.h> int main() { int a[100],b[100],i,j,n; printf("Enter the number of array elements\n"); scanf("%d",&n); printf("Enter %d elements\n",n); for(i=0;i<n;i++) { scanf("%d",&a[i]); } printf("The array elements are\n"); for(i=0;i<n;i++) { printf("%d\n",a[i]); } for(i=0,j=n-1;i<n;i++,j--) { b[i]=a[j]; } printf("The copied array elements in reverse order are\n"); for(i=0;i<n;i++) { printf("%d\n",b[i]); } getch(); return 0; }</pre>

16) WAP to check whether the given number is present in a array or not.	17) WAP to check whether the given number is present in a array or not and if present find its position.
<pre>#include<stdio.h> #include<conio.h> int main() { int a[100],i,n,num,flag=0; printf("Enter the number of array elements\n"); scanf("%d",&n); printf("Enter %d elements\n",n); for(i=0;i<n;i++) { scanf("%d",&a[i]); } printf("\nEnter the element that you want to search \n"); scanf("%d",&num); for(i=0;i<n;i++) { if(num==a[i]) { flag=1; break; } } if(flag==1) { printf("your number is found"); } else { printf("your number is not found"); } getch(); return 0; }</pre>	<pre>#include<stdio.h> #include<conio.h> int main() { int a[100],i,n,num,flag=0,pos; printf("Enter the number of array elements\n"); scanf("%d",&n); printf("Enter %d elements\n",n); for(i=0;i<n;i++) { scanf("%d",&a[i]); } printf("\nEnter the element that you want to search \n"); scanf("%d",&num); for(i=0;i<n;i++) { if(num==a[i]) { flag=1; pos=i; break; } } if(flag==1) { printf("your number is found at index a[%d]",pos); } else { printf("your number is not found"); } getch(); return 0; }</pre>

Assignment: Write a program to search an element in one-dimensional array containing five integer elements.[PU:2017 fall]

18) WAP to input n number in an array and sort them in Ascending order.[PU:2018 spring]	19) WAP to input n number in an array and sort them in Descending order.
<pre>#include<stdio.h> #include<conio.h> int main() { int a[100],i,j,n,temp; printf("Enter number of array elements \n"); scanf("%d",&n); printf("Enter %d elements\n",n); for(i=0;i<n;i++) { scanf("%d",&a[i]); } printf("\nArray before sorting are:\n"); for(i=0;i<n;i++) { printf("%d\n",a[i]); } for(i=0;i<n-1;i++) { for(j=0;j<n-i-1;j++) { if(a[j]>a[j+1]) { temp=a[j]; a[j]=a[j+1]; a[j+1]=temp; } } } printf("\nsorted array in ascending oder are:\n"); for(i=0;i<n;i++) { printf("%d\n",a[i]); } getch(); return 0; }</pre>	<pre>#include<stdio.h> #include<conio.h> int main() { int a[100],i,j,n,temp; printf("Enter the number of array elements\n"); scanf("%d",&n); printf("Enter %d elements\n",n); for(i=0;i<n;i++) { scanf("%d",&a[i]); } printf("\nArray before sorting are:\n"); for(i=0;i<n;i++) { printf("%d\n",a[i]); } for(i=0;i<n-1;i++) { for(j=0;j<n-i-1;j++) { if(a[j]<a[j+1]) { , temp=a[j]; a[j]=a[j+1]; a[j+1]=temp; } } } printf("\nArray element in descending order are:\n"); for(i=0;i<n;i++) { printf("%d\n",a[i]); } getch(); return 0; }</pre>

20) WAP to read marks of n students and print the marks of top five.

```
#include<stdio.h>
#include<conio.h>
int main()
{
    float marks[100],temp;
    int i,j,n;
    printf("Enter the number of students\n");
    scanf("%d",&n);
    printf("Enter the marks of %d students\n",n);
    for(i=0;i<n;i++)
    {
        scanf("%f",&marks[i]);
    }
    printf("\nEnter marks of student are:\n");
    for(i=0;i<n;i++)
    {
        printf("%f\n",marks[i]);
    }
    for(i=0;i<n-1;i++)
    {
        for(j=0;j<n-i-1;j++)
        {
            if(marks[j]<marks[j+1])
            {
                temp=marks[j];
                marks[j]=marks[j+1];
                marks[j+1]=temp;
            }
        }
    }
    printf("\nMarks of Top five students are:\n");
    for(i=0;i<5;i++)
    {
        printf("%f\n",marks[i]);
    }
    getch();
    return 0;
}
```

21) WAP to Store Fibonacci series of 10 terms in array and display them.	22) WAP to store n numbers in an array and display prime numbers stored in array and calculate the sum of those prime numbers
<pre>#include<stdio.h> #include<conio.h> int main() { int fib[10],i; fib[0]=0; fib[1]=1; printf("The fibonacci series is \n"); printf("%d\t%d\t",fib[0],fib[1]); for(i=2;i<=9;i++) { fib[i]=fib[i-1]+fib[i-2]; } for(i=2;i<=9;i++) { printf("%d\t",fib[i]); } getch(); return 0; }</pre>	<pre>#include<stdio.h> #include<conio.h> int main() { int a[100],flag,n,i,j,sum=0; printf("Enter the no. of elements:"); scanf("%d",&n); printf("Enter %d array elements\n",n); for(i=0;i<n;i++) { scanf("%d",&a[i]); } printf("\nprime numbers are:\n"); for(i=0;i<n;i++) { flag=0; for(j=2;j<a[i];j++) { if(a[i]%j==0) { flag=1; break; } } if(flag==0) { printf("%d\t",a[i]); sum=sum+a[i] } } printf("\nThe sum of prime no. is %d",sum); getch(); }</pre>

Two Dimensional Array (Program solution)

1) WAP to input 3*3 matrix and display it.	2) WAP to read the matrix of size 2*3 from user and display it to the screen.
<pre>#include <stdio.h> #include <conio.h> int main() { int arr[3][3]; int i,j; printf("Enter the 9 elements of matrix:\n"); for(i=0; i<3; i++) { for(j=0; j<3 ;j++) { scanf("%d",&arr[i][j]); } } printf("Entered matrix is:\n"); for(i=0; i<3; i++) { for(j=0; j<3 ;j++) { printf("%d\t",arr[i][j]); } printf("\n"); } getch(); return 0; }</pre>	<pre>#include<stdio.h> #include<conio.h> int main() { int matrix[2][3],i,j; printf("Enter the matrix element\n"); for(i=0;i<2;i++) { for(j=0;j<3;j++) { scanf("%d",&matrix[i][j]); } } printf("Entered matrix is :\n"); for(i=0;i<2;i++) { for(j=0;j<3;j++) { printf("%d\t",matrix[i][j]); } printf("\n"); } getch(); return 0; }</pre>

3) WAP to input m*n order matrix and display it.	4) Write a program to read matrix of order m*n from user and multiply each element of matrix by 3.
<pre>#include<stdio.h> #include<conio.h> int main() { int arr[20][20],i,j,m,n; printf("Enter the size of row and column of matrix\n"); scanf("%d%d",&m,&n); printf("Enter %d elements of matrix",m*n); for(i=0;i<m;i++) { for(j=0;j<n;j++) { scanf("%d",&arr[i][j]); } } printf("Enterd matrix is :\n"); for(i=0;i<m;i++) { for(j=0;j<n;j++) { printf("%d\t",arr[i][j]); } printf("\n"); } getch(); return 0; }</pre>	<pre>#include<stdio.h> #include<conio.h> int main() { int arr[20][20],i,j,m,n; printf("Enter the Row size and column size of matrix\n"); scanf("%d%d",&m,&n); printf("Enter %d elements of matrix",m*n); for(i=0;i<m;i++) { for(j=0;j<n;j++) { scanf("%d",&arr[i][j]); } } for(i=0;i<m;i++) { for(j=0;j<n;j++) { arr[i][j]=3*arr[i][j]; } printf("\n"); } for(i=0;i<m;i++) { for(j=0;j<n;j++) { printf("%d\t",arr[i][j]); } printf("\n"); } getch(); return 0; }</pre>

5) WAP to input 3*3 order matrix and perform the following operations

- Find sum of all elements
- Product of all elements

*(Perform similar operation for m*n matrix)*

Write a program to find the sum of all elements of 3 x 3 matrix.[PU:2013 spring]

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

```

int main()
{
    int arr[3][3],sum=0;
    int i,j;
    printf("Enter the 9 elements of matrix:\n");
    for(i=0; i<3; i++)
    {
        for(j=0; j<3 ;j++)
        {
            scanf("%d",&arr[i][j]);
        }
    }
    for(i=0; i<3; i++)
    {
        for(j=0; j<3 ;j++)
        {
            sum=sum+arr[i][j];
        }
    }
    printf("sum of all elements in matrix is %d",sum);
    getch();
    return 0;
}

```

Modify the above program to find average of all elements of m*n matrix (Do yourself)

Program to input 3*3 order matrix and find product of all elements

```

#include <stdio.h>
#include <conio.h>
int main()
{
    int arr[3][3],pro=1;
    int i,j;
    printf("Enter the elements of matrix:\n");
    for(i=0; i<3; i++)
    {
        for(j=0; j<3 ;j++)
        {
            scanf("%d",&arr[i][j]);
        }
    }
    for(i=0; i<3; i++)
    {
        for(j=0; j<3 ;j++)
        {
            pro=pro*arr[i][j];
        }
    }
    printf("product of all elements in matrix is %d",pro);
    getch();
    return 0;
}

```

<p>6) WAP to input m*n order matrix and find its transpose.</p> <pre>#include<stdio.h> #include<conio.h> int main() { int arr[20][20],i,j,m,n; printf("Enter the number of rows and column of matrix\n"); scanf("%d%d",&m,&n); printf("Enter %d elements of matrix\n",m*n); for(i=0;i<m;i++) { for(j=0;j<n;j++) { scanf("%d",&arr[i][j]); } } printf("Entered matrix is :\n"); for(i=0;i<m;i++) { for(j=0;j<n;j++) { printf("%d\t",arr[i][j]); } printf("\n"); } printf("Transpose of the matrix is :\n"); for(i=0;i<n;i++) { for(j=0;j<m;j++) { printf("%d\t",arr[j][i]); } printf("\n"); } getch(); return 0; }</pre>	<p>7) WAP to find transpose of 4*5 matrix.</p> <pre>#include<stdio.h> #include<conio.h> int main() { int arr[20][20],i,j; printf("Enter 20 elements of matrix\n"); for(i=0;i<4;i++) { for(j=0;j<5;j++) { scanf("%d",&arr[i][j]); } } printf("Entered matrix is :\n"); for(i=0;i<4;i++) { for(j=0;j<5;j++) { printf("%d\t",arr[i][j]); } printf("\n"); } printf("Transpose of the matrix is :\n"); for(i=0;i<5;i++) { for(j=0;j<4;j++) { printf("%d\t",arr[j][i]); } printf("\n"); } getch(); return 0; }</pre> <p>Assignment: Write a program to find the transpose of 4*3 matrix. [PU:2019 spring]</p>
---	--

8) Write a program to read a matrix and find the sum of all digits in its main diagonal.

[PU:2015 fall] OR

Write a program to find sum of diagonal elements of mxn matrix.(diagonal elements from left/trace of matrix)
[PU:2016 fall]

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int arr[20][20],i,j,m,n,sum=0;
    printf("Enter the order of matrix\n");
    scanf("%d%d",&m,&n);
    if(m==n)
    {
        printf("Enter %d elements of matrix\n",m*n);
        for(i=0;i<m;i++)
        {
            for(j=0;j<n;j++)
            {
                scanf("%d",&arr[i][j]);
            }
        }
        for(i=0;i<m;i++)
        {
            for(j=0;j<n;j++)
            {
                if(i==j)
                {
                    sum=sum+arr[i][j];
                }
            }
        }
        printf("sum of diagonal elements from left=%d",sum);
    }
    else
    {
        printf("Invalid order of matrix for this operation\n");
    }
    getch();
    return 0;
}
```

9) WAP to read m*n matrix and find the sum of diagonal elements from right

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int arr[20][20],i,j,m,n,sum=0;
    printf("Enter the order of matrix\n");
    scanf("%d%d",&m,&n);
    if(m==n)
    {
        printf("Enter %d elements of matrix\n",m*n);
        for(i=0;i<m;i++)
        {
            for(j=0;j<n;j++)
            {
                scanf("%d",&arr[i][j]);
            }
        }
        for(i=0;i<m;i++)
        {
            for(j=0;j<n;j++)
            {
                if((i+j)==(m-1))
                {
                    sum=sum+arr[i][j];
                }
            }
        }
        printf("sum of diagonal elements from right=%d",sum);
    }
    else
    {
        printf("Invalid order of matrix for this operation\n");
    }
    getch();
    return 0;
}
```

Assignment: Modify the above program for (3*3) matrix .

10) WAP to input m*n order matrix and find Largest element.	11) WAP to input m*n order matrix and find Smallest element.
<pre>#include<stdio.h> #include<conio.h> int main() { int arr[20][20],i,j,m,n,large; printf("Enter the order of matrix \n"); scanf("%d%d",&m,&n); printf("Enter %d elements of matrix\n",m*n); for(i=0;i<m;i++) { for(j=0;j<n;j++) { scanf("%d",&arr[i][j]); } } large=arr[0][0]; for(i=0;i<m;i++) { for(j=0;j<n;j++) { if(large<arr[i][j]) { large=arr[i][j]; } } } printf("largest element=%d",large); getch(); return 0; }</pre>	<pre>#include<stdio.h> #include<conio.h> int main() { int arr[20][20],i,j,m,n,small; printf("Enter the order of matrix\n"); scanf("%d%d",&m,&n); printf("Enter %d elements of matrix\n",m*n); for(i=0;i<m;i++) { for(j=0;j<n;j++) { scanf("%d",&arr[i][j]); } } small=arr[0][0]; for(i=0;i<m;i++) { for(j=0;j<n;j++) { if(small>arr[i][j]) { small=arr[i][j]; } } } printf("smallest element=%d",small); getch(); return 0; }</pre>

- 12) WAP to input m*n order matrix and find sum of all even and odd numbers and count them.

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int arr[20][20],i,j,m,n,esum=0,osum=0,ecount=0,octcount=0;
    printf("Enter the order of matrix\n");
    scanf("%d%d",&m,&n);
    printf("Enter %d elements of matrix\n",m*n);
    for(i=0;i<m;i++)
    {
        for(j=0;j<n;j++)
        {
            scanf("%d",&arr[i][j]);
        }
    }
    for(i=0;i<m;i++)
    {
        for(j=0;j<n;j++)
        {
            if(arr[i][j]%2==0)
            {
                esum=esum+arr[i][j];
                ecount++;
            }
            else
            {
                osum=osum+arr[i][j];
                octcount++;
            }
        }
    }
    printf("sum of even number=%d\n",esum);
    printf("Number even number=%d\n",ecount);
    printf("Sum of odd number=%d\n",osum);
    printf("Number of odd number=%d",octcount);
    getch();
    return 0;
}
```

- 13) Write a program to read values of 3*3 order matrix the compute the sum of even elements.[PU:2012 fall]

```

#include<stdio.h>
#include<conio.h>
int main()
{
    int arr[3][3],i,j,esum=0;
    printf("Enter the elements of matrix\n");
    for(i=0;i<3;i++)
    {
        for(j=0;j<3;j++)
        {
            scanf("%d",&arr[i][j]);
        }
    }
    for(i=0;i<3;i++)
    {
        for(j=0;j<3;j++)
        {
            if(arr[i][j]%2==0)
            {
                esum=esum+arr[i][j];
            }
        }
    }
    printf("Sum of even elements=%d",esum);
    getch();
    return 0;
}

```

14) Write a program to enter values in 3*3 order matrix and compute the sum of odd elements.[PU:2019 fall]

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int arr[3][3],i,j,osum=0;
    printf("Enter the elements of matrix\n");
    for(i=0;i<3;i++)
    {
        for(j=0;j<3;j++)
        {
            scanf("%d",&arr[i][j]);
        }
    }
    for(i=0;i<3;i++)
    {
        for(j=0;j<3;j++)
        {
            if(arr[i][j]%2!=0)
            {
                osum=osum+arr[i][j];
            }
        }
    }
    printf("Sum of odd elements=%d",osum);
    getch();
    return 0;
}
```

15) WAP to input m*n matrix and find sum of each row	16) WAP to input m*n matrix and find sum of each column
<pre>#include<stdio.h> #include<conio.h> int main() { int arr[20][20],i,j,m,n,sum; printf("Enter the order of matrix\n"); scanf("%d%d",&m,&n); printf("Enter %d elements of matrix\n",m*n); for(i=0;i<m;i++) { for(j=0;j<n;j++) { scanf("%d",&arr[i][j]); } } for(i=0;i<m;i++) { sum=0; for(j=0;j<n;j++) { sum=sum+arr[i][j]; } printf("sum of %d row is %d\n",i+1,sum); } getch(); return 0; }</pre>	<pre>#include<stdio.h> #include<conio.h> int main() { int arr[20][20],i,j,m,n,sum; printf("Enter order of matrix\n"); scanf("%d%d",&m,&n); printf("Enter %d elements of matrix\n",m*n); for(i=0;i<m;i++) { for(j=0;j<n;j++) { scanf("%d",&arr[i][j]); } } for(i=0;i<m;i++) { sum=0; for(j=0;j<n;j++) { sum=sum+arr[j][i]; } printf("sum of %d column is %d\n",i+1,sum); } getch(); return 0; }</pre>

17) WAP to input m*n order matrix and find the sum of particular row requested by user.

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int arr[20][20],i,j,r,sum=0,m,n;
    printf("Enter the order of matrix\n");
    scanf("%d%d",&m,&n);
    printf("Enter %d elements of matrix\n",m*n);
    for(i=0;i<m;i++)
    {
        for(j=0;j<n;j++)
        {
            scanf("%d",&arr[i][j]);
        }
    }
    printf("Enter the row that you want find sum\n");
    scanf("%d",&r);
    if(r<=m)
    {
        r=r-1;
        for(i=0;i<n;i++)
        {
            sum=sum+arr[r][i];
        }
        printf("sum of %d row is %d\n",r+1,sum);
    }
    else
    {
        printf("Invalid request");
    }

    getch();
    return 0;
}
```

18) WAP to input m*n matrix and find sum of particular column requested by user.

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int arr[20][20],i,j,c,sum=0,m,n;
    printf("Enter the order of matrix\n");
    scanf("%d%d",&m,&n);
    printf("Enter %d elements of matrix\n",m*n);
    for(i=0;i<m;i++)
    {
        for(j=0;j<n;j++)
        {
            scanf("%d",&arr[i][j]);
        }
    }
    printf("Enter the column that you want find sum\n");
    scanf("%d",&c);
    if(c<=n)
    {
        c=c-1;
        for(i=0;i<m;i++)
        {
            sum=sum+arr[i][c];
        }
        printf("sum of %d column is %d\n",c+1,sum);
    }
    else
    {
        printf("Invalid request!");
    }

    getch();
    return 0;
}
```

19) WAP to input m*n order matrix and convert it to the upper triangular matrix.	20) WAP to input m*n order matrix and convert it to the lower triangular matrix.
<pre> #include<stdio.h> #include<conio.h> int main() { int arr[20][20],i,j,m,n; printf("Enter the order of matrix\n"); scanf("%d%d",&m,&n); if(m==n) { printf("Enter %d elements of matrix\n",m*n); for(i=0;i<m;i++) { for(j=0;j<n;j++) { scanf("%d",&arr[i][j]); } } printf("Entered matrix is :\n"); for(i=0;i<m;i++) { for(j=0;j<n;j++) { printf("%d\t",arr[i][j]); } printf("\n"); } printf("The upper triangular matrix is\n"); for(i=0;i<m;i++) { for(j=0;j<n;j++) { if(i>j) { printf("0\t"); } else { printf("%d\t",arr[i][j]); } } printf("\n"); } else { printf("Invalid order of matrix for given operation\n"); } getch(); return 0 ; </pre>	<pre> #include<stdio.h> #include<conio.h> int main() { int arr[20][20],i,j,m,n; printf("Enter the order of matrix\n"); scanf("%d%d",&m,&n); if(m==n) { printf("Enter %d elements of matrix\n",m*n); for(i=0;i<m;i++) { for(j=0;j<n;j++) { scanf("%d",&arr[i][j]); } } printf("Entered matrix is :\n"); for(i=0;i<m;i++) { for(j=0;j<n;j++) { printf("%d\t",arr[i][j]); } printf("\n"); } printf("The Lower triangular matrix is\n"); for(i=0;i<m;i++) { for(j=0;j<n;j++) { if(j>i) { printf("0\t"); } else { printf("%d\t",arr[i][j]); } } printf("\n"); } else { printf("Invalid order of matrix for given operation\n"); } getch(); return 0; </pre>

<p>21) WAP to find the norm of a matrix.</p>	<p>22) Write a program to add two 3X3 matrix. Display the sum stored in third matrix.[PU:2018 spring]</p>
<pre>#include<stdio.h> #include<conio.h> #include<math.h> int main() { int arr[20][20],i,j,m,n,sum=0; float norm; printf("Enter the order of matrix\n"); scanf("%d%d",&m,&n); printf("Enter %d elements of matrix\n",m*n); for(i=0;i<m;i++) { for(j=0;j<n;j++) { scanf("%d",&arr[i][j]); } } printf("Entered matrix is :\n"); for(i=0;i<m;i++) { for(j=0;j<n;j++) { printf("%d\t",arr[i][j]); } printf("\n"); } for(i=0;i<m;i++) { for(j=0;j<n;j++) { sum=sum+pow(arr[i][j],2); } } norm=sqrt((float)sum); printf("Norm of the matrix=%f",norm); getch(); return 0; }</pre>	<pre>#include<stdio.h> #include<conio.h> int main() { int a[3][3],b[3][3],sum[3][3],i,j; printf("Enter elements of first matrix\n"); for(i=0;i<3;i++) { for(j=0;j<3;j++) { scanf("%d",&a[i][j]); } } printf("Enter elements of second matrix\n"); for(i=0;i<3;i++) { for(j=0;j<3;j++) { scanf("%d",&b[i][j]); } } for(i=0;i<3;i++) { for(j=0;j<3;j++) { sum[i][j]=a[i][j]+b[i][j]; } } printf("sum of matrix is :\n"); for(i=0;i<3;i++) { for(j=0;j<3;j++) { printf("%d\t",sum[i][j]); } printf("\n"); } getch(); return 0; }</pre>

23) WAP a Program to read two 3*3 matrix and subtract them	24) WAP to read two 3*3 matrix and multiply them.
<pre>#include<stdio.h> #include<conio.h> int main() { int a[3][3],b[3][3],sub[3][3],i,j; printf("Enter elements of first matrix\n"); for(i=0;i<3;i++) { for(j=0;j<3;j++) { scanf("%d",&a[i][j]); } } printf("Enter 9elements of second matrix\n"); for(i=0;i<3;i++) { for(j=0;j<3;j++) { scanf("%d",&b[i][j]); } } for(i=0;i<3;i++) { for(j=0;j<3;j++) { sub[i][j]=a[i][j]-b[i][j]; } } printf("substraction of matrix is :\n"); for(i=0;i<3;i++) { for(j=0;j<3;j++) { printf("%d\t",sub[i][j]); } printf("\n"); } getch(); return 0; }</pre>	<pre>#include<stdio.h> #include<conio.h> int main() { int a[3][3],b[3][3],mul[3][3],i,j,k; printf("Enter 9 elements of first matrix\n"); for(i=0;i<3;i++) { for(j=0;j<3;j++) { scanf("%d",&a[i][j]); } } printf("Enter 9 elements of second matrix\n"); for(i=0;i<3;i++) { for(j=0;j<3;j++) { scanf("%d",&b[i][j]); } } for(i=0;i<3;i++) { for(j=0;j<3;j++) { mul[i][j]=0; for(k=0;k<3;k++) { mul[i][j]=mul[i][j]+a[i][k]*b[k][j]; } } } printf("Mulplication of matrix is :\n"); for(i=0;i<3;i++) { for(j=0;j<3;j++) { printf("%d\t",mul[i][j]); } printf("\n"); } getch(); return 0; }</pre>

25) WAP to read two m*n matrix and multiply them if possible.

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int a[20][20],b[20][20],mul[20][20],i,j,k,r1,c1,r2,c2;
    printf("Enter the row and column of first matrix\n");
    scanf("%d%d",&r1,&c1);
    printf("Enter the row and column of second matrix\n");
    scanf("%d%d",&r2,&c2);
    if(c1!=r2)
    {
        printf("Matrix multiplication is not possible");
    }
    else
    {
        printf("Enter %d elements of first matrix\n",r1*c1);
        for(i=0;i<r1;i++)
        {
            for(j=0;j<c1;j++)
            {
                scanf("%d",&a[i][j]);
            }
        }
        printf("Enter %d elements of second matrix\n",r2*c2);
        for(i=0;i<r2;i++)
        {
            for(j=0;j<c2;j++)
            {
                scanf("%d",&b[i][j]);
            }
        }
        for(i=0;i<r1;i++)
        {
            for(j=0;j<c2;j++)
            {
                mul[i][j]=0;
                for(k=0;k<c1;k++)
                {
                    mul[i][j]=mul[i][j] +a[i][k]*b[k][j];
                }
            }
        }
    }
}
```

```

printf("Multiplication of matrix is :\n");
for(i=0;i<r1;i++)
{
    for(j=0;j<c2;j++)
    {
        printf("%d\t",mul[i][j]);
    }
    printf("\n");
}
getch();
return 0;
}

```

26) WAP to test whether given two matrix are equal or not.[PU:2013 fall]

```

#include <stdio.h>
#include<conio.h>
#include <stdlib.h>
int main()
{
int a[20][20], b[20][20];
int i, j, m1, n1, m2, n2, flag = 1;
printf("Enter the order of first matrix \n");
scanf("%d%d", &m1, &n1);
printf("Enter the order of second matrix \n");
scanf("%d%d", &m2, &n2);
if (m1==m2&&n1==n2)
{
    printf("Enter the elements of first matrix \n");
    for (i = 0;i<m1;i++)
    {
        for (j = 0; j<n1; j++)
        {
            scanf("%d",&a[i][j]);
        }
    }
    printf("Enter the elements of second matrix \n");
    for (i = 0; i < m2; i++)
    {
        for (j = 0; j <n2; j++)
        {
            scanf("%d",&b[i][j]);
        }
    }
}

```

```

for (i =0;i<m2;i++)
{
    for (j =0;j<n2; j++)
    {
        if (a[i][j] != b[i][j])
        {
            flag = 0;
            break;
        }
    }
}
if (flag == 1)
{
    printf("Two matrices are equal \n");
}
else
{
    printf("Two matrices are not equal \n");
}
}

else
{
    printf("Two matrix cannot be compared\n");
}
getch();
return 0;
}

```

- 27) Write a program that asks a user for a number and find outs if the number is present in 2D array given below.

```

int arr[3][3]=
{
{6,37,33},
{12,11,13},
{14,85,96}
};

```

```

#include<stdio.h>
#include<conio.h>
int main()
{
int arr[3][3]={{6,37,33},{12,11,13},{14,85,96}};
int i,j,num,flag=0;
printf("Enter the number that you want to search \n");
scanf("%d",&num);
for(i=0;i<3;i++)
{
    for(j=0;j<3;j++)
    {
        if(num==arr[i][j])
        {
            flag=1;
            break;
        }
    }
}
if(flag==1)
{
    printf("your number is found") ;
}
else
{
    printf("your number is not found") ;
}
getch();
return 0;
}

```

Strings

What is string? [PU: 2014 fall]

- Strings are sequence of characters stored in consecutive memory location.
- Each character in string occupies one byte of memory.
- Strings always terminated with null character '\0'.

Declaration:

```
char string_name[size];
```

Here, size determines the number of characters in string_name.

Eg. char name[5];

Note: When the compiler assigns a character string to character array, it automatically supplies a null character ('\0') at the end of string. Therefore, size should be equal to maximum number of characters in string plus one.

Initialization of strings

Compile time initialization

General form:

```
char string_name[size] = "list of character";
```

eg. char name[5] = "raju";

OR

```
char string_name[size] = {list of character};
```

eg. char name[5] = {'r', 'a', 'j', 'u', '\0'};

Note: C also permits to initialize a character array without specifying the number of elements.

In such cases size of array will be determined automatically, based on number of elements initialized.

Eg. char name[] = {'r', 'a', 'j', 'u', '\0'};

defines the array string as a five element array.

Program to illustrate compile time initialization of string

```
#include<stdio.h>
#include<conio.h>
int main()
{
```

```
char greeting[6]={'H','e','l','l','o','\0'};
char name[5]="raju";
printf("Greeting message:%s",greeting);
printf("\nName: %s",name);
getch();
return 0;
}
```

Output:

```
Greeting message:Hello
Name:raju
```

Runtime initialization

The familiar input function scanf() can be used with %s format specification to read string.

Example:

```
char name[20];
scanf("%s",name);
```

Limitation: Here, string variable takes only single word, It is because when whitespace is encountered the scanf() function terminates,

Output:

```
Enter your name: Dennis Ritchie
Your name is: Dennis
```

To overcome this problem the gets() function is used to read a string of text containing whitespaces, until newline character is encountered.

```
#include<stdio.h>
#include<conio.h>
int main()
{
    char name[20];
    printf("Enter your name:");
    gets(name);
    printf("Your name is:");
    puts(name);
    getch();
    return 0;
}
```

Output:

```
Enter your name: Dennis Ritchie
Your name is: Dennis Ritchie
```

Array of strings

Array of strings means two dimensional array of characters.

Eg. char name[5][10];

Here, first dimension tells, how many strings can be stored in array. The second dimension tells the maximum length of each string.

In above declaration, we can store 5 strings, each can store maximum 9 characters. Last 10th space is for null terminator in each string.

Program to input name of 5 students and display it.

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int i;
    char name[5][20];
    printf("Enter the name of 5 students\n");
    for(i=0;i<5;i++)
    {
        gets(name[i]);
    }
    printf("The names are\n");
    for(i=0;i<5;i++)
    {
        puts(name[i]);
    }
    getch();
    return 0;
}
```

String handling function

Write a short notes on:

String handling functions. [PU: 2013 fall, PU: 2014 spring2017 spring PU: 2017 fall]

Explain any seven functions related to string.[PU:2019 fall]

String handling function makes the manipulation of string easier and available in string.h header file. Some of them are as follows:

Functions	Purpose	Syntax
strlen()	Finds the length of a string excluding null character	integer_variable=strlen(string);
strcpy()	Copies one string to another including null character	strcpy(destination_string,source_string);
strrev()	Reverses all characters in string except null character	strrev(string);
strcmp()	Compares two strings to find out whether they are same or different. This functions accepts two strings as parameters and returns an integer whose value is i) Less than 0 if the first string is less than second ii) Equal to 0 if both are same iii) Greater than 0 if first string is greater than second	integer_variable=strcmp(string1,string2);
strcat()	Concatenates two strings. i.e. it appends one string at the end of other	strcat(string1,string2);
strlwr()	converts the uppercase string into lowercase	strlwr(string);
strupr()	converts the lowercase string into uppercase	strupr(string);

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
int main ()
{
    int l ;
    char str1[20] = "pokhara";
    char str2[20] = "university";
    char str3[20];
    l=strlen(str1);
    printf("Length of the str1 is %d\n",l);
    strcpy(str3,str1);
    printf("string copied from str1 to str3 is %s\n",str3 );
    strcat(str1,str2);
    printf("string after concatenation(s1+s2)is %s\n",str1);
    strrev(str3);
    printf("Reverse of string str3 is %s\n",str3);
   strupr(str3);
    printf("Uppercase of string str3 is %s\n",str3);
    getch();
    return 0;
}
```

Output:

```
Length of the str1 is 7
string copied from str1 to str3 is pokhara
string after concatenation(s1+s2)is pokharauniversity
Reverse of string str3 is arahkop
Uppercase of string str3 is ARAHKOP
```

String handling functions

Explain any three string handling functions with examples.[PU:2018 fall]

String handling functions makes manipulation of strings easier. String handing functions are available in string.h header file. Some of them are as follows:

a) **strlen()**

This function returns the an integer which denotes the length of a string passed. Length of a string is number of characters present in it, excluding the terminating null character.

Its syntax is :

```
integer_variable =strlen(string);
```

Program

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
int main()
{
    char str[20];
    int len;
    printf("Enter the string:");
    gets(str);
    len=strlen(str);
    printf("The length of a string is %d",len);
    getch();
    return 0;
}
```

Output:

```
Enter the string : ramesh
The length of string is 6
```

b) **strcpy()**

This function copies one string to another. The function accepts two strings as parameters and copies the second string character by character into first one upto including the null character of the second string.

The syntax is :

```
strcpy(destination_string,source_string);
```

i.e. strcpy(str2,str1) means the content of str1 is copied to str2.

Program

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
int main()
{
    char str1[20]="ramesh",str2[20];
    strcpy(str2,str1);
    printf("The copied string is %s",str2);
    getch();
    return 0;
}
Output:
```

The copied string is ramesh

c) **strcat()**

This function concatenates two strings i.e it appends one string at the end of another.
This function accepts two strings as parameters and stores the contents of second string at the end of first. Its syntax is :

```
strcat (string1,string2);
ie.string1=string1+string2
```

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
int main()
{
    char str1[20]="pokhara",str2[20]="university";
    strcat(str1,str2);
    printf("The concatenated string is %s",str1);
    getch();
    return 0;
}
```

Output:

The concatenated string is pokharauniversity

d) **strcmp()**

This function compares two strings to find out whether they are same or different. This function accepts two strings as parameters and returns an integer whose value is

- i) less than 0 if the first string is less than second
- ii) equal to 0 if both are same
- iii) greater than 0 if first string is greater than second

The two strings are compared character by character until there is a mismatch or end of one string is reached. Whenever two characters in two string differ, the string which has the character with higher ASCII value is greater.

For example, consider two strings "ram" and "rajesh". The first two characters are same but the third character in string ram and that is in rajesh are different. Since ASCII value of character m in string is ram is greater than that of j in string rajesh, the string ram is greater than rajesh.

Its syntax is:

```
integer_variable=strcmp(string1,string2);
```

Program

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
int main()
{
    char str1[20],str2[20];
    int diff;
    printf("Enter the first string:");
    gets(str1);
    printf("Enter the second string:");
    gets(str2);
    diff=strcmp(str1,str2);
    if(diff>0)
    {
        printf("%s is greater than %s",str1,str2);
    }
    else if(diff<0)
    {
        printf("%s is greater than %s ",str2,str1);
    }
    else
    {
        printf("Both strings are same");
    }
    getch();
    return 0;
}
```

Output

```
Enter first string: ram
Enter second string: rajesh
ram is greater than rajesh
```

e) **strrev()**

This function is used to reverse all characters in a string except null character at the end of string. The reverse of string “abc” is “cba”.

It's syntax is strrev(string);

For example:

strrev(s) means it reverses the characters in string s and stores reversed string in s.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
int main()
{
    char str[20]="hello";
    strrev(str);
    printf("Reversed string is %s\n",str);
    getch();
    return 0;
}
```

Output

Reversed string is olleh

f) **strupr()**

This function converts the lowercase string into uppercase.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
int main()
{
    char str[20]="hello";
   strupr(str);
    printf("The uppercase of given string is %s",str);
    getch();
    return 0;
}
```

Output:

The uppercase of given string is HELLO

g) strlwr()

This function converts the uppercase string into lowercase.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
int main()
{
    char str[20]="HELLO";
    strlwr(str);
    printf("The lowercase of given string is %s",str);
    getch();
    return 0;
}
```

Output

The lowercase of given string is hello.

<p>1) WAP to concatenate two strings inputted through keyboard. The result of concatenation should be displayed after copied on third variable.</p>	<p>2) WAP to sort n students name in alphabetical order.[PU: 2014 Fall]</p>
<pre>#include<stdio.h> #include<conio.h> #include<string.h> int main() { char str1[20],str2[20],str3[40]; printf("Enter the first string\n"); gets(str1); printf("Enter the second string\n"); gets(str2); strcpy(str3,strcat(str1,str2)); puts(str3); getch(); return 0; }</pre>	<pre>#include<stdio.h> #include<conio.h> #include<string.h> int main() { int i,j,n; char name[100][20],temp[20]; printf("Enter the number of students\n"); scanf("%d",&n); printf("Enter the name of %d students:\n",n); for(i=0;i<n;i++) { gets(name[i]); } for(i=0;i<n-1;i++) { for(j=0;j<n-1-i;j++) { if(strcmp(name[j],name[j+1])>0) { strcpy(temp,name[j]); strcpy(name[j],name[j+1]); strcpy(name[j+1],temp); } } } printf("Name of students in alphabetical order are\n"); for(i=0;i<n;i++) { puts(name[i]); } getch(); return 0; }</pre>

Assignment:

Write a program to read n employees names and display them in alphabetical order.[PU:2014 spring]

<p>3) Write a program to check whether the given string is palindrome or not.(palindrome is a word which reads same from left to right and right to left.eg LIRIL,MADAM etc.</p> <p>[PU: 2013 spring]</p>	<p>4) Program to check whether a given string is palindrome or not, without using string handling function.</p>
<pre>#include<stdio.h> #include<conio.h> #include<string.h> int main() { char str1[20],str2[20]; printf("Enter the string\n"); gets(str1); strcpy(str2,str1); strrev(str2); if(strcmp(str1,str2)==0) { printf("String is palindrome"); } else { printf("String is not palindrome"); } getch(); return 0; }</pre>	<pre>#include<stdio.h> #include<conio.h> int main() { char str[20]; int i,len=0,flag=1; printf("Enter the string\n"); gets(str); for(i=0;str[i]!='\0';i++) { len++; } for(i=0;i<len/2;i++) { if(str[i]!=str[len-i-1]) { flag=0; break; } } if(flag==1) { printf("String is palindrome"); } else { printf("String is not palindrome"); } getch(); return 0; }</pre>

5) Write a program to insert a given character at a given array index of a given string. For example if the given string is “Gnesh”, given character is ‘a’, and the given array index is 1, the resulting string should be “Ganesh”. [PU: 2013 Fall]

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
int main()
{
    char str[20],ch;
    int len,i,pos;
    printf("Enter the string\n");
    gets(str);
    printf("Enter the index position you want to insert\n");
    scanf("%d",&pos);
    printf("Enter the character you want to insert\n");
    scanf("%c",&ch);
    len=strlen(str);
    for(i=len+1;i>pos;i--)
    {
        str[i]=str[i-1];
    }
    str[pos]=ch;
    printf("Resulting string is \n");
    puts(str);
    getch();
    return 0;
}
```

Note: This solution is similar for this question.

Write a program to insert a given character in the array index of string. For example if the string is ‘Nepl’, given character is ‘a’, and the given array index is 3, resulting string should be Nepal.

6) WAP to convert all the uppercase letter to lowercase and vice versa in a string given by user.

```
#include<stdio.h>
#include<conio.h>
int main()
{
    char str[20];
    int i;
    printf("Enter the string\n");
    gets(str);
    for(i=0;str[i]!='\0';i++)
    {
        if(str[i]>='a'&&str[i]<='z')
        {
            str[i]=str[i]-32;
        }
        else
        {
            str[i]=str[i]+32;
        }
    }
    printf("The converted string is %s",str);
    getch();
    return 0;
}
```

<p>7) Alternative solution to convert all the uppercase letter to lowercase and vice versa in a string given by user</p>	<p>8) WAP to search a specified word in a given string.</p>
<pre>#include<stdio.h> #include<conio.h> #include<string.h> #include<ctype.h> int main() { char str[20]; int i,l; printf("Enter the string\n"); gets(str); l=strlen(str); for(i=0;i<l;i++) { if(islower(str[i])) { str[i]=toupper(str[i]); } else { str[i]=tolower(str[i]); } } printf("\nThe converted string is\n"); puts(str); getch(); return 0; }</pre>	<pre>#include<stdio.h> #include<conio.h> int main() { char str[100], word[20]; int i, index, flag = 0; printf("Enter any string: "); gets(str); printf("Enter word to be searched: "); gets(word); for(index=0;str[index] != '\0';index++) { if(str[index] == word[0]) { flag = 1; for(i=0;word[i] != '\0';i++) { if(str[index + i] != word[i]) { flag = 0; break; } } } if(flag == 1) { break; } } if(flag == 1) { printf("Word is found in a given string"); } else { printf("Word is not found in a given string"); } getch(); return 0; }</pre>

9) WAP to check whether the Entered character is found in a given string or not.

```
#include <stdio.h>
#include<conio.h>
#include <string.h>
int main()
{
    char ch, str[50];
    int i,len,flag = 0;
    printf("Enter the string:");
    gets(str);
    printf("Enter character: ");
    scanf("%c", &ch);
    len=strlen(str);
    for (i = 0; i <len; i++)
    {
        if (str[i] == ch)
        {
            flag = 1;
            break;
        }
    }
    if (flag==1)
    {
        printf("character is found in given
string\n");
    }
    else
    {
        printf("Character is not found in given
string\n");
    }
    getch();
    return 0;
}
```

10) Write a program to declare name of 10 persons in an array in a program already and input any name from the keyboard and check whether it is present in name list or not.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
int main()
{
    char
list[10][20]={"ram","hari","sita","gopal","narayan","an
"madhav", "manoj", "anish", "roshan"};
    int i,j,flag=0;
    char name[20];
    printf("Enter the name that you want to search\n");
    gets(name);
    for(i=0;i<10;i++)
    {
        if(strcmp(list[i],name)==0)
        {
            flag=1;
            break;
        }
    }
    if(flag==1)
    {
        printf("Entered name is found");
    }
    else
    {
        printf("Entered name is not found");
    }
    getch();
    return 0;
}
```

<p>11) WAP to print the following pattern: [PU:2009 Fall]</p> <p>1N 2EE 3PPP 4AAAA 5LLLL</p>	<p>12) WAP to print the following pattern:</p> <p>E EN ENG ENGI ENGIN ENGINE ENGINEE ENGINEER ENGINEERI ENGINEERIN ENGINEERING</p>
<pre>#include<stdio.h> #include<conio.h> #include<string.h> int main() { char str[20]="NEPAL"; int i,j,len; len=strlen(str); for(i=0;i<len;i++) { printf("%d",i+1); for(j=0;j<=i;j++) { printf("%c",str[i]); } printf("\n"); } getch(); return 0; }</pre>	<pre>#include<stdio.h> #include<conio.h> #include<string.h> int main() { char str[20]="ENGINEERING"; int i,j,l; l=strlen(str); for(i=0;i<l;i++) { for(j=0;j<=i;j++) { printf("%c",str[j]); } printf("\n"); } getch(); return 0; }</pre>

Assignment:

P
PR
PRO
PROG
PROGR
PROGRAM
PROGRAMM
PROGRAMMI
PROGRAMMIN
PROGRAMMING

Write a program to generate the following pattern by initializing string at first.[PU:2019 spring]

PROGRAMMING
PROGRAMMIN
PROGRAMMI
PROGRAMM
PROGRAM
PROGRA
PROGR
PROG
PRO
PR
P

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
int main()
{
    char str[20]="PROGRAMMING";
    int i,j,len;
    len=strlen(str);
    for(i=len;i>=0;i--)
    {
        for(j=0;j<i;j++)
        {
            printf("%c",str[j]);
        }
        printf("\n");
    }
    getch();
    return 0;
}
```

13) What will be the output after executing the following codes.(assume necessary header files)

```
void main()
{
    char str[20]={"university"};
    char str1[20];
    strcpy(str1,"pokhara");
    printf("\n%s",str1);
    strcat(str1," ");
    strcat(str1,str);
    printf("\n%s",str1);
    strrev(str1);
    printf("\n%s",str1);
    getch();
}
```

Output:

```
pokhara
pokhara university
ytisrevinu arahkop
```

(Trace the program yourself)

What will be the output after executing the following codes.(assume necessary header files)

```
void main()
{
    char str[]="HELLO WORLD";
    for(m=0;str[m]!='\0';m++)
    {
        if(m%2==0)
            printf("%c",str[m]);
    }
    getch();
}
```

Output:

```
HLOWRD
```

(Trace the program yourself)

14) Without using header file <string.h> perform the following operations.

- i. Find the length of string
- ii. Copy a string
- iii. Reverse a string
- iv. Concatenate a string

Program to find the length of a string without using header file<string .h>	WAP to reverse a string without using <string.h>
<pre>#include<stdio.h> #include<conio.h> int main() { char str[20]; int i,len=0; printf("Enter the string\n"); gets(str); for(i=0;str[i]!='\0';i++) { len++; } printf("Length of string is %d",len); getch(); return 0; }</pre>	<pre>#include<stdio.h> #include<conio.h> int main() { int len=0,i,j; char str1[20],str2[20] ; printf("Enter the string\n"); gets(str1); for(i=0;str1[i]!='\0';i++) { len++; } for(i=len-1,j=0;j<len;i--,j++) { str2[j]=str1[i]; } str2[j]='\0'; printf("The Original string is\n"); puts(str1); printf("The Reverse string is\n"); puts(str2); getch(); return 0; }</pre>
Program to copy a string without using header file<string.h>	WAP to concatenate two strings without using <string.h>
<pre>#include<stdio.h> #include<conio.h> int main() { char str1[20],str2[20]; int i; printf("Enter the string\n"); gets(str1); for(i=0;str1[i]!='\0';i++) { str2[i]=str1[i]; } str2[i]='\0'; printf("Copied string is:"); puts(str2); getch(); return 0; }</pre>	<pre>#include<stdio.h> #include<conio.h> int main() { int i,j; char str1[20],str2[20],str3[40]; printf("Enter the fist string\n"); gets(str1) ; printf("Enter the second string\n"); gets(str2) ; for(i=0;str1[i]!='\0';i++) { str3[i]=str1[i]; } for(j=0;str2[j]!='\0';j++) { str3[i+j]=str2[j]; } str3[i+j]='\0' ; puts(str3); getch(); return 0; }</pre>

<p>15) WAP to input a string and count no of vowels, comma, semicolon, space, digits and constants.</p> <pre>#include<stdio.h> #include<conio.h> int main() { char str[50]; int i, vowel=0,comma=0,semicolon=0,space=0,digit=0,consonant=0; printf("Enter the string\n"); gets(str); while(str[i]!='\0') { if(str[i]=='a' str[i]=='A' str[i]=='e' str[i]=='E' str[i]=='i' str[i]=='I' str[i]=='O' str[i]=='o' str[i]=='U' str[i]=='u') { vowel++; } else if(str[i]==',') { comma++; } else if(str[i]==';') { semicolon++; } else if(str[i]==' ') { space++; } else if(str[i]>='0'&&str[i]<='9') { digit++; } else { consonant++; } i++; } printf("\nNumber of vowels=%d",vowel); printf("\nNumber of comma =%d",comma); printf("\nNumber of semicolon =%d",semicolon); printf("\nNumber of space =%d",space); printf("\nNumber of digits =%d",digit); printf("\nNumber of consonants=%d",consonant); getch(); return 0; }</pre>	<p>16) WAP to read string and rewrite in Alphabetical order.</p> <pre>#include<stdio.h> #include<conio.h> #include<string.h> int main() { char str[20]; int i,j,temp,len; char name[20]; printf("Enter the string\n"); gets(str); len=strlen(str); for(i=0;i<len-1;i++) { for(j=0;j<len-i-1;j++) { if(str[j]>str[j+1]) { temp=str[j]; str[j]=str[j+1]; str[j+1]=temp; } } } puts(str); getch(); return 0; }</pre>
--	---

CHAPTER-6

Function

- Function is a self-contained block of code or statement that performs a particular task.
- Every program must contain one function named main() from where the program execution begins.
- Complex problems can be solved by breaking them into sets of sub-problems, called modules or functions. This technique is called divide and conquer. Each module can be implemented independently and later can be combined into a single unit.

Advantages of function

- It makes programs significantly easier to understand and maintain by breaking them up into easily manageable chunks.
- A single function can be used multiple times which avoids rewriting of the same code again and again.
- Different programmers working on one large project divide the workload by writing different functions.
- Programs that use functions are easier to design, debug and maintain.
- C functions can be used to build a customized library of frequently used routines.

Disadvantage

It increases the execution time because when a function is called, the control has to jump to the function definition to perform the particular task, and after completion of the task, the control again comes back to the function call.

1. What do you mean by functions in C programming? [PU: 2012 Fall]
2. Why functions are used? [PU: 2016 Spring]
3. Without using functions, we can write a program. But we need functions in our program. What are the benefits of using them. [PU 2008 Fall]

Types of functions

C functions are classified into two categories. They are:

1. Library function

These are the functions which are already written, compiled and placed in C Library and they are not required to be written by a programmer. The function name, its return type, their argument number and types have been already defined. We can use these functions as required by just calling them. For example: printf(), scanf(), sqrt(), getch() are examples of library functions. The library functions are also known as built-in functions and they are defined within a particular header file.

2. User defined function

These are the functions which are defined by user at the time of writing a program. The user has choice to choose its name, return type, arguments and their types.

eg. int sum(int a,int b).The task of each user-defined functions is as defined by user. A complex C program can be divided into number of user-defined functions.

Differentiate between library functions and user-defined functions with examples.
[PU: 2015 Fall]

Library functions	User defined functions
1. They are predefined functions/built in functions in C library.	1. They are the function which are created by the user as per his own requirements.
2. They are the part of header files (such as stdio.h, conio.h, math.h) which are called during runtime.	2. They are the part of program which is compiled at runtime.
3. The name of function ID is given by developers which can't be changed.	3. The name of function id is declared by user which can be changed.
4. The function's name, its return type, their arguments number and their types have been already defined .	4. The user has choice to choose function name, its return type, number of argument and their types.
5.Example. printf(),scanf(),sqrt(),pow(),clrscr() etc.	5.Example. int fact(int n), float sum(float x,float y)

Program to illustrate use of user defined function and library functions.

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
int square(int);
int main()
{
int num=5,result;
result=pow(num,2);
printf("Square of number using library function =%d",result);
result= square(num);
printf("\nSquare of number using user defined function=%d",result);
}
int square(int x)
{
return (x * x);
}
```

Output:

```
Square of number using library function=25  
Square of number using user defined function=25
```

Components associated with function

1) Function definition

The collection of program statements that describes the specific task to be done by the function is called function definition. It consists of:

Function header: which defines function's name, its return type and its argument list

Function body : which is a block of code enclosed in parenthesis.

Syntax:

```
return_type function_name(data_type varibale1,data_type variable2,.....,data_type  
variable n)  
{  
statements ;  
.....  
.....  
return value;  
}
```

Note: If a function doesn't return any value , then its return type is void

Syntax:

```
void function_name (data_type varibale1,data_type variable2,.....,data_type  
variable n)  
{  
statements ;  
.....  
.....  
}
```

2) Function prototype/ function declaration [PU:2018 fall,PU:2019 fall]

The function declaration or prototype is a model or blueprint of a function. If a function is used before it is defined in a program, then function declaration or prototype is needed to provide the following information to the compiler.

- The name of function
- The type of the value returned by the function
- The number and type of arguments that must be supplied while calling the function.

The syntax for function declaration is :

```
return_type function_name(type1, type2 ,type3 ..... type n);
```

Here, return_type specifies the data type of the value returned by the function. A function can return value of any data type. If there is no return value, the keyword void is used.

The function declaration and declarator or header in function definition must use the same function name, number of arguments, argument types and return types. Some examples of function prototype are:

```
int add(int a,int b);
```

```
int factorial(int n);
```

3) Accessing/calling a function

When a program calls a function, the program control is transferred to the called function. A called function performs a defined task and when its return statement is executed or when its function-ending closing brace is reached, it returns the program control back to the main program.

A function can be called or accessed by specifying its name, followed by a list of arguments enclosed in parentheses and separated by commas. For example, the function add() with two arguments is called by add(a,b) to add two numbers.

During function call, function name must match with function prototype name, the type of return type, the number of arguments and order of arguments.

4) Return statement:

The job of return statement is to hand over some value given by function body to the point from where the call was made. The function defined with its return type must return a value of that type. For example: If a function has return type int, it returns an integer value from the called function to the calling function.

The main functions return statement are:

- It immediately transfers the control back to the calling program after execution of return statement (i.e. no statement within the function body is executed after the return statement.)
- It returns value to the calling function.

The syntax for return is:

```
return (expression);
```

Where expression must evaluate to a value of the type specified in the function header for the return value.

5) Function parameter (Arguments)

Parameters provide the data communication between calling function and called function.

They can be classified into actual parameters and formal parameters.

Actual parameters:

- These are the parameters that transfer data from the calling function to the called function.
- This type of parameter is used in function call.

Formal parameters:

- These parameters used to receive the data sent by the calling function. These are also used to send data from calling function to the called function.
- These parameters used in function definition.

In given example the parameter a and b are actual parameter which are used in calling function and x and y are formal parameters which are used in called function.

Define function, function definition, function calling, function declaration with code example.[2016 spring]

Program

```
#include<stdio.h>
#include<conio.h>
int sum(int,int);           //function declaration

int main()
{
    int a, b,result;
    printf("Enter the first number\n");
    scanf("%d",&a);
    printf("Enter the second number\n");
    scanf("%d",&b);

    /* Calling the function here, the function return type
       is integer so we need an integer variable to hold the
       returned value of this function.
    */
    result =sum(a,b);
    /*actual arguments a and b are passed in calling function*/
    printf ("Sum of two numbers= %d, result);

    return 0;
}

/*formal arguments x and y are passed in called function*/
int sum(int x, int y) //function definition
{
    int r;
    r=x+y;

    /* Function return type is integer so we are returning
       an integer value, the sum of the passed numbers.
    */
    return r;
}
```

Passing argument to function

- Distinguish between call by value and call by reference with examples.
[PU:2008 fall,2011 spring,2012 fall,2013 spring]
- How arguments can be passed by using call by value and call by reference? Explain with examples.**[2017 Fall]**
- Explain the call by value and call by pointer with suitable example.
[2017 spring][PU:2019 fall]

The arguments in function can be passed in two ways, namely **call by value** and **call by reference**.

Function call by value (or pass by value)

- In this method the value of each actual arguments in the calling function is copied into corresponding formal arguments of the called function.
- With this method the changes made to the formal arguments in the called function have no effect on the values the actu[al argument in the calling function.

```
#include<stdio.h>
#include<conio.h>
void swap(int x,int y);
int main()
{
    int a,b;
    a=10;
    b=20;
    printf("value before swapping a=%d and b=%d",a,b);
    swap(a,b);
    printf("\nvalue after swapping a=%d and b=%d",a,b);
    getch();
    return 0;
}
void swap(int x,int y)
{
    int temp;
    temp=x;
    x=y;
    y=temp;
}
```

Output:

Value before swapping a=10 and b=20

Value after swapping a=10 and b=20

Explanation:

In this example, the values of a and b are passed in function swap() by value. The value of a is copied into formal argument x and value of b is copied into formal argument y. The copy of value of a and b to x and y means the original values of a and b remain same and these values are copied into the variables x and y also. Thus when x and y are changed within the function , the values of the variables x and y are changed but the original value of a and b remains same.

Function call by reference (Or pass by Reference)

- In this method the address of actual arguments in the calling function are copied into formal arguments of the called function .This means that using these addresses the actual arguments can be accessed.
- In call by reference since the address of the value is passed any changes made to the value reflects in the calling function.

```
#include<stdio.h>
#include<conio.h>
void swap(int *x,int *y);

int main()
{
    int a,b;
    a=10;
    b=20;
    printf("The value before swapping are a=%d and b=%d",a,b);
    swap(&a,&b);
    printf("\n\nThe values after swapping are a=%d and b=%d",a,b);
    getch();
    return 0;
}

void swap(int *x,int *y)
{
    int temp;
    temp=*x;
    *x=*y;
    *y=temp;

}
```

Value before swapping a=10 and b=20
Value after swapping a=20 and b=10

Explanation: In this example ,the address of variables (ie.&a and &b) are passed in function .These addresses are received by corresponding formal arguments in function definition. To receive addresses, the formal arguments must be of type pointers which stores the address of variables. The address of aand b are copied to the pointer variable x and y in function definition. When the values in addresses pointed by these pointer variables are altered, the values of original variables are also changed as the content of their addresses have been changed .Thus, while arguments are passed in function by reference ,the original values are changed if they are changed within function.

Category of functions:

Function can be categorized in four types, on the basis of arguments and return value.

- 1.Function with no arguments and no return values
- 2.Function with arguments but no return values
- 3.Function with no arguments but with return values
- 4.Function with arguments and return values

1. Function with no arguments and no return values

When a function has no arguments, it does not receive any data from the calling function. Similarly, when it doesn't return a value, the calling function does not receive any data from the called function. Thus in such type of functions, there is no transfer between the calling function and the called function. This type of function is defined as.

```
void function_name( )  
{  
/*body of the function*/  
}
```

The keyword **void** means the function does not return any value. There is no arguments within parenthesis which implies function has no argument and it does not receive any data from the called function.

Program to illustrate the “function with no arguments and no return values”

```
#include<stdio.h>
#include<conio.h>
void sum();
int main()
{
    sum();
    getch();
    return 0;
}

void sum()
{
    int x,y,r;
    printf("Enter the two numbers\n");
    scanf("%d%d",&x,&y);
    r=x+y;
    printf("The sum of numbers=%d",r);
}
```

2. Function with arguments but no return value

These type of function has arguments and receives the data from the calling function. The function completes the task and does not return any values to the calling function. Such type of functions are defined as

```
void function_name(argument list)
{
    /*body of function*/
}
```

Program to illustrate the “function with arguments but no return values”

```
#include<stdio.h>
#include<conio.h>
void sum(int,int);
int main()
{
    int a,b;
    printf("Enter the two numbers\n");
    scanf("%d%d",&a,&b);
    sum(a,b);
    getch();
    return 0;
}
```

```

void sum(int x,int y)
{
    int r;
    r=x+y;
    printf("The sum of numbers=%d",r);

}

```

3. Function with no arguments but with return values

These type of function does not receive any arguments but the function return values to the calling function. Such type of functions are defined as

```

return_type function_name( )
{
    /*body of function*/
}

```

Program to illustrate the “function with no arguments and with return values”

```

#include<stdio.h>
#include<conio.h>
int sum();

int main()
{
    int result;
    result=sum();
    printf("The sum of numbers=%d",result);
    getch();
    return 0;
}

int sum()
{
    int x,y,r;
    printf("Enter the two numbers\n");
    scanf("%d%d",&x,&y);
    r=x+y;
    return r;
}

```

4. Function with arguments and return value

The function of this category has arguments and receives the data from the calling function. After completing its task ,it returns the result to the calling function through return statement. Thus there is data transfer between called function and calling function using return values and arguments. These type of functions are defined as

```
return_type function_name(argument list)
{
/*body of the function*/
}
```

Program to illustrate the “function with arguments and return values”

```
#include<stdio.h>
#include<conio.h>
int sum(int,int);

int main()
{
    int a,b,result;
    printf("Enter the two numbers\n");
    scanf("%d%d",&a,&b);
    result=sum(a,b);
    printf("The sum of numbers=%d",result);
    getch();
    return 0;
}

int sum(int x,int y)
{
    int r;
    r=x+y;
    return r;
}
```

Global vs Local variable

[PU: 2012 fall, 2016 spring, 2018 fall]

Global variables:

Global variables are accessible to all functions defined in the program. Global variables are declared outside any function, generally at the top of program after preprocessor directives. It is useful to declare variable global if it is to be used by many functions in the program. The default initial values for this variable is zero. The lifetime is as long as the program execution does not come to an end.

Local variables:

The variables that are defined within the body of a function or block and local to that function or block only are known as local variables. The name and value of local variables are valid within the function in which it is declared. They are unknown to other. The local variables are created when the function is called and destroyed automatically when function is exited function.

Example:

```
#include<stdio.h>
#include<conio.h>
void fun();
int x=5;      //global declaration
int main()
{
    int a=10;      //local declaration
    printf("value of x=%d,a=%d",x,a);
    fun();
    getch();
    return 0;
}
void fun()
{
    int b=20;      //local declaration
    printf("\nvalue of x=%d,b=%d",x,b);
}
```

Output

```
Value of x=5, a=10
Value of x=5, b=20
```

Here x is global variable and both main () function and fun() function can access them. a and b are local variables a can be accessed by only main() function and b can be only accessed only by fun() function.

Storage classes in c

What do you mean by storage class? Explain different types of storage classes in C? Use examples to illustrate.[PU:2014 spring,2014 fall,2016 fall, 2016 spring,2017 spring]

Write a short notes on: storage class in c? [PU: 2017 spring]

In C language, each variable has a storage class which decides the following things:

- **Scope:** where the value of the variable would be available inside a program.
- **Default initial value:** if we do not explicitly initialize that variable, what will be its default initial value.
- **Storage location of that variable**
- **Lifetime of that variable:** for how long will that variable exist.

Syntax: storage_classSpecifier typeSpecifier variableName

	Storage location	Default initial value	Scope	Lifetime
Auto	Memory	Garbage value	local to function in which variable is defined.	Till control remains within block where it is defined
register	CPU register	Garbage value	local to function in which variable is defined.	Till control remains within block where it is defined
Static	Memory	Zero	local to function in which variable is defined.	Value of the variable persist between different function call
extern	Memory	Zero	Everywhere in the program	Till the program doesn't finish its execution

Explanation:

1. Automatic or Local variables(auto storage class)

- The automatic variables are always declared within a function or block.
- As local variables are defined within the body of the function or the block other functions cannot access these variables, the compiler shows error in case other function try to access the variables.
- The local variables are created when the function is called and destroyed automatically when the function is exited.
- The keyword **auto** is used for storage class specification while declaration of variable.
- A variable declared inside a function without storage class specification **auto** is, by default, an automatic variable.

Storage Location	Memory
Default initial value	Garbage value
Scope	local to function in which variable is defined.
Life time	Till control remains within block where it is defined

Example:

```
#include<stdio.h>
#include<conio.h>
void function1(void);
int main()
{
    auto int a=5;
    printf("a=%d\n",a);
    function1();
    getch();
    return 0;

}
void function1()
{
    auto int b=10;
    printf("b=%d",b);
}
```

Output

a=5
b=10

2. Register variables (Register storage class)

Register variables inform the compiler to store the variable in CPU register instead of memory. Register variables have faster accessibility than a normal variable. Generally, the frequently used variables are kept in registers. But only a few variables can be placed inside registers. If the declaration of register variables exceeds the availability, they will be automatically converted into non register variables (automatic variable). One application of register storage class can be in using loops, where the variable gets used a number of times in the program, in a very short span of time. Register variable are declared by using the keyword **register**.

Storage Location	CPU Register
Default initial value	Garbage value
Scope	local to function in which variable is defined.
Life time	Till control remains within block where it is defined

Example:

```
#include<stdio.h>
#include<conio.h>
int main()
{
    register int i;
    for (i=0;i<5;i++)
    {
        printf("%d\t", i);
    }
    getch();
    return 0;
}
```

Output

0 1 2 3 4

3. External or Global variables(extern storage class)

- The external or global variables are declared outside any block or function.
- Global variable can be accessed by any function in the program.

Storage Location	Memory
Default initial value	Zero
Scope	(Global) Everywhere in the program
Life time	As long as the program execution doesn't come to end.

Example:

```
#include<stdio.h>
#include<conio.h>
void fun1(void);
int a=10;
int main()
{
    printf("Initial value=%d",a);
    fun1();
    printf("\nFinal value=%d",a);
    getch();
    return 0;
}
void fun1()
{
    a++;
}
```

Output

```
Initial value=10
Final value=11
```

4. Static variables(static storage class)

A static variable tells the compiler to persist/save the variable until the end of program. Instead of creating and destroying a variable every time when it comes into and goes out of scope, static variable is initialized only once and remains into existence till the end of the program.

Storage Location	Memory
Default initial value	Zero
Scope	local to function in which variable is defined.
Life time	Value of the variable persist between different function call

Example:

```
#include<stdio.h>
#include<conio.h>
void increment(void);
int main()
{
increment();
increment();
increment();
increment();
getch();
return 0;
}
void increment(void)
{
static int i = 1 ;
printf ( "%d\t", i );
i++;
}
```

Output

1 2 3 4

Preprocessor directives

What is preprocessor directives? [PU: 2015 fall,2019 spring]

Preprocessor is a program that process source code before it passes through the compiler. It operates under the control of which is known as preprocessor directives.

Preprocessor directives are preceded by a hash (#) sign.

Preprocessor directives are often placed in beginning of program before main function.

No semicolon (;) is expected at the end of the preprocessor directive.

Example:

```
#include
#define
#ifndef
#define
#endif
#if
#else
#endif
```

We use preprocessor directive for

1. File inclusion

Source code of the given “filename” is included in the main program in specified place.

Syntax:

#include<filename>

#include "filename"

Eg.#include<stdio.h>

2. Conditional compilation

Set of commands are included or excluded in source program before compilation with respect to condition.

Eg.

#ifdef	Returns true if this macro is defined
#ifndef	Returns true if this macro is not defined
#if	Test if compile time condition is true
#else	The alternative for if
#endif	Ends preprocessor conditional

3. Macro expansion

Macros are a piece of code in a program which is given some name. Whenever this name is encountered by the compiler the compiler replaces the name with the actual piece of code. The ‘#define’ directive is used to define a macro.

Eg. #define PI 3.1415

Here, when PI is used in program its value is replaced with 3.1415

MACROS

Write a short notes on: **macros [PU: 2013 spring, 2015 spring, 2019 fall]**

A macro is a fragment of code which has been given a name. Whenever the name is used, it is replaced by the contents of the macro. Macro is defined by #define directive.

#define macro_name macro_expansion

- macro_name is any valid C identifier, and it is generally in capital letters to distinguish it from other variables.
- macro_expansion can be any text

There are two types of macros:

1) Object-like Macros

The object-like macro is an identifier that is replaced by value. It is widely used to represent numeric constants.

For example: #define PI 3.14

Here, PI is the macro name which will be replaced by the value 3.14.

Example

```
#include <stdio.h>
#define PI 3.14
int main()
{
    float r,area;
    printf("Enter the radius: ");
    scanf("%f", &r);
    area = PI*r*r;
    printf("Area=%.2f",area);
    return 0;
}
```

2) Function-like Macros

The function-like macro looks like function call. For example:

#define max(a,b) ((a)>(b)?(a):(b)) Here, max is the macro name.

```

#include<stdio.h>
#include<conio.h>
#define max(a,b) ((a>b)?(a):(b))
int main()
{
    int a,b;
    printf("Enter the two numbers\n");
    scanf("%d%d",&a,&b);
    printf("Maximum number=%d",max(a,b));
    getch();
    return 0;
}

```

C Predefined Macros

ANSI C defines many predefined macros that can be used in c program.

No.	Macro	Description
1	_DATE_	represents current date in "MMM DD YYYY" format.
2	_TIME_	represents current time in "HH:MM:SS" format.
3	_FILE_	represents current file name.
4	_LINE_	represents current line number.
5	_STDC_	It is defined as 1 when compiler complies with the ANSI standard.

C predefined macros example

```

#include<stdio.h>
#include<conio.h>
int main()
{
    printf("File :%s\n", __FILE__ );
    printf("Date :%s\n", __DATE__ );
    printf("Time :%s\n", __TIME__ );
    printf("Line :%d\n", __LINE__ );
    printf("STDC :%d\n", __STDC__ );
    getch();
    return 0;
}

```

Differentiate between macro and function.

[PU: 2009 Fall, 2015 Fall]

Macro	Function
1. Macro is preprocessed.	1. Function is compiled.
2. Before Compilation macro name is replaced by macro value.	2. During function call , Transfer of Control takes place
3. Macros are faster in execution than function.	3. Functions are bit slower in execution.
4. Useful where small code appears many time	4. Useful where large code appears many time
5. Generally Macros do not extend beyond one line	5. Function can be of any number of lines
6. Increase the program size.	6. Makes program smaller and compact
7. Macros do not check for compilation error which leads to unexpected output.	7. Function checks for compilation error and there is a less chance of unexpected output
8. Macro can't call it recursively	8. Function can call it recursively.
9. A macro is defined using #define preprocessor directive. Therefore it is preprocessed before submission of source code to the compiler.	9. A function is defined outside main program. It is processed after submission of source code to compiler.

Example:

Program to find the square of a number by using both macro and function

```
#include<stdio.h>
#include<conio.h>
#define NUM 5
#define square(num) (num*num)
void func();
int main()
{
    printf("Square of a number using macro=%d",square(NUM));
    func();
    return 0;
}
```

```
void func()
{
    int result,a=5;
    result=a*a;
    printf("\nSquare of number using function=%d",result);
}
```

Output:

Square of number using macro=25
Square of number using function=25

Header files in C

A header file is a file with extension .h which contains C function declarations and macro definitions to be shared between several source files. There are two types of header files: the files that the programmer writes and the files that comes with your compiler.

Including a header file is equal to copying the content of the header file but we do not do it because it will be error-prone and it is not a good idea to copy the content of a header file in the source files, especially if we have multiple source files in a program.

Both the user and the system header files are included using the preprocessing directive #include. It has the following two forms –

#include <file>

This form is used for system header files. It searches for a file named 'file' in a standard list of system directories.

#include "file"

This form is used for header files of your own program. It searches for a file named 'file' in the directory containing the current file.

Why header files in C are included in program? Give reasons. Also list out different header files you know. Illustrate the program showing the use of header file. [PU: 2015 fall]

Header files are the special files which stores the predefined library functions. Suppose if we are using printf() & scanf() functions in your program then we have to include <stdio.h> header file. It is for standard i/o. If we don't include the the header files we can't run our program. Therefore we have to include the header files.

Another header file is #include<conio.h> which is for console i/o. This header file stores clrscr() & getch() functions. clrscr() is for clearing the screen & getch() is holding the screen until we press any single character from keyboard.

Also we have lots of different header files which stores different functions

Header file	function
#include<stdio.h>	Used to perform input and output operations in C like scanf() and printf().
#include<string.h>	Perform string manipulation operations like strlen and strcpy
#include<conio.h>	Perform console input and console output operations like clrscr() to clear the screen and getch() to get the character from the keyboard.
#include<stdlib.h>	Perform standard utility functions like dynamic memory allocation, using functions such as malloc() and calloc().
#include<math.h>	Perform mathematical operations like sqrt() and pow(). To obtain the square root and the power of a number respectively.
#include<signal.h>	Perform signal handling functions like signal() and raise(). To install signal handler and to raise the signal in the program respectively
#include<errno.h>	Used to perform error handling operations like errno().

Program to illustrate the use of header file

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
int main()
{
    int num,result;
    printf("Enter the number");
    scanf("%d",&num);
    result=pow(num,2);
    printf("Square of a given number is %d",result);
    getch();
    return 0;
}
```

Here,

Function used	Header file must included to use function
printf() and scanf()	stdio.h
getch()	conio.h
pow()	math.h

Recursive function (Nesting of function)

1. What is recursive function? [PU:2013 fall,2013 spring]
2. List out the major advantages of recursive function? [PU:2018 spring,2016 Fall,2019 spring]

A function that calls itself is known as a recursive function. Recursion is a process by which function calls itself repeatedly until some specified condition will be satisfied.

To solve a problem using recursive method, two conditions must be satisfied .They are

- Problem could be written or defined in terms of previous result
- Problem statement must include stopping condition

Advantages

- Avoid unnecessary calling of functions.
- Through Recursion one can solve problems in easy way while its iterative solution is very big and complex
- Reduces time complexity.
- Using recursion, the length of the program can be reduced.
- Extremely useful when applying the same solution repeatedly.

Disadvantages

- Recursive solution is always logical and it is very difficult to trace. (debug and understand).
- For each step we make a recursive call to a function. For which it occupies significant amount of stack memory with each step.
- It is usually slower due to the overhead of maintaining the stack.
- May cause stack-overflow if the recursion goes too deep to solve the problem
- If the programmer forgets to specify the exit condition in the recursive function, the program will execute out of memory.

WAP to find the factorial of a given number using recursive function
[PU: 2013 Fall, 2013 spring]

```
#include<stdio.h>
#include<conio.h>
long int fact(int n);
int main()
{
    int num;
    long int f;
    printf("Enter the number\n");
    scanf("%d",&num);
    f=fact(num);
    printf("The factorial of a given number is %ld",f);
    getch();
    return 0;
}
long int fact(int n)
{
    if(n==1)
        return 1;
    else
        return n*fact(n-1);
}
```

Write a program to find the sum of first n natural number using recursive function
[PU: 2018 Fall]

```
#include<stdio.h>
#include<conio.h>
int snatural(int n);
int main()
{
    int num,s;
    printf("Enter the value of n \n");
    scanf("%d",&num);
    s=snatural(num);
    printf("The sum of n natrual number is %d",s);
    getch();
    return 0;
}
```

```

int snatural(int n)
{
    if(n==1)
        return 1;
    else
        return n+snatural(n-1);
}

```

**Write a program to generate the Fibonacci series upto nth term using recursive function.
Fibonacci series is 0,1,1,2,3,5.... [PU: 2012 fall, 2010 fall, 2006 fall, 2019 fall]**

```

#include<stdio.h>
#include<conio.h>
int fib(int n);
int main()
{
    int n,result,i;
    printf("Enter how many terms you want to generate");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        result=fib(i);
        printf("%d\t",result);
    }
    getch();
    return 0;
}

int fib(int n)
{
    if (n==0 || n==1)
        return n;
    else
        return(fib(n-1) + fib(n-2));
}

```

Write a recursive program to generate the first 15 numbers of Fibonacci series.

[PU: 2014 Spring]

```
#include<stdio.h>
#include<conio.h>
int fib(int n);
int main()
{
    int i,result;
    for(i=0;i<15;i++)
    {
        result=fib(i);
        printf("%d ",result);
    }
    getch();
    return 0;
}
int fib(int n)
{
    if (n==0 || n==1)
        return n;
    else
        return(fib(n-1) + fib(n-2));
}
```

Write a recursive program to generate 10 terms Fibonacci sequence starting from 2.

[PU: 2016 Fall]

```
#include<stdio.h>
#include<conio.h>
int fib(int n);
int main()
{
    int i,result;
    for(i=2;i<12;i++)
    {
        result=fib(i);
        printf("%d\t",result);
    }
    getch();
    return 0;
}
```

```

int fib(int n)
{
    if (n==2 || n==3)
        return n;
    else
        return(fib(n-1) + fib(n-2));
}

```

**Write a program to find nth term of Fibonacci number using recursive function.
[PU: 2010 Fall]**

```

#include<stdio.h>
#include<conio.h>
int fib(int n);
int main()
{
    int n,result;
    printf("Enter the term:");
    scanf("%d",&n);
    result=fib(n);
    printf("The %d fibonacci term is %d",n,result);
    getch();
    return 0;
}

```

```

int fib(int n)
{
    if (n==1)
        return 0;
    else if(n==2)
        return 1;
    else
        return(fib(n-1) + fib(n-2));
}

```

Write a program to read an integer number and find the sum of digits using recursive function. [PU: 2015 Fall]

```
#include<stdio.h>
#include<conio.h>
int sum(int n);
int main()
{
    int num, result;
    printf("Enter the number: ");
    scanf("%d", &num);
    result = sum(num);
    printf("Sum of digits of given number=%d",result);
    return 0;
}

int sum (int n)
{
    if (n == 0)
        return 0;
    else
        return (n % 10 + sum(n / 10));
}
```

Use recursive functions calls to evaluate. [PU: 2015 spring, 2016 spring]

$$f(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$$

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
int fact(int n);
int main()
{
    float sum=0,nume,deno,x;
    int n,i,sign;
    printf("\nEnter the value of x and n\n");
    scanf("%f%d",&x,&n);
    for(i=1;i<=n;i++)
    {
        nume=pow(x,2*i-1);
        deno=fact(2*i-1);
        sign=pow(-1,i+1);
        sum=sum+sign*nume/deno;
    }
    printf("sum of series=%f",sum);
    getch();
    return 0;
}
int fact(int n)
{
    if(n==1 || n==0)
        return 1;
    else
        return (n*fact(n-1));
}
```

Write a program to read an integer and find the number of digits present in it using recursive function.

```
#include<stdio.h>
#include<conio.h>
int count(int n);
int main()
{
    int num, result;
    printf("Enter the number: ");
    scanf("%d", &num);
    result = count(num);
    printf("Number of digits in a given number=%d",result);
    return 0;
}

int count(int n)
{
    if (n == 0)
        return 0;
    else
        return (1+ count(n/10));
}
```

Write a recursive program to raise the power of X to n. (i.e. X^n) [PU:2019 spring]

```
#include<stdio.h>
#include<conio.h>
float power (float x,int n);
int main()
{
float x,result;
int n;
printf("Enter the value of x\n");
scanf("%f",&x);
printf("Enter the value of n\n");
scanf("%d",&n);
result=power(x,n);
printf("Result=%f",result);
getch();
return 0;
}
```

```

float power(float x,int n)
{
    if(n==0)
    {
        return 1;
    }
    else if (n>0)
    {
        return x*power(x,n-1);
    }
    else
    {
        return (1/x)*power(x,n+1);
    }
}

```

WAP to check whether the given number is Armstrong number or not using recursive function. (for three digit number)

```

#include<stdio.h>
#include<conio.h>
int armstrong(int);
int main()
{
    int num,a;
    printf("Enter any three digit number");
    scanf("%d",&num);
    a=armstrong(num);
    if(a==num)
    {
        printf("The number is armstrong number");
    }
    else
    {
        printf("The number is not armstrong number");
    }
}

```

```

int armstrong(int x)
{
    if(x==0)
        return 0;
    else
        return((x%10)*(x%10)*(x%10)+armstrong(x/10));
}

```

Recursion vs Iteration

Recursion	Iteration
1. The statement in a body of function calls the function itself.	1. Allows the set of instructions to be repeatedly executed.
2. Recursion is always applied to functions.	2. Iteration is applied to iteration statements or "loops".
3. In recursive function, only termination condition (base case) is specified	3. Iteration includes initialization, condition, execution of statement within loop and updation (increments/decrements) the control variable
4. If the function does not converge to some condition called (base case), it leads to infinite recursion.	4. If the control condition in the iteration statement never become false, it leads to infinite iteration
5. The stack is used to store the set of new local variables and parameters each time the function is called	5. Does not uses stack.
6. Infinite recursion can crash the system.	6. Infinite loop uses CPU cycles repeatedly.
7. Slow in execution	7. Fast in execution.

Describe the output generated by each of the following programs.[PU:2015 spring]

```
#include<stdio.h>
int a=100,b=200;
int funct1(int c);
main()
{
int count,c;
for(count=1;count<=10;++count)
{
    c=4*count;
    printf("%d\n",funct1(c));
}
}
funct1(int x)
{
    int c;
    c=(x<30)?(a-x):(b+x);
    return (c);
}
```

Here a=100,b=200 Where a and b are global variables.so that they can accessed from anywhere.

count	c=4*count	funct1(c)	c=(x<30)?(a-x) : (b+x)	print c
1	c=4*1=4	funct1(4)	(4<30)?(True) c=a-x=100-4=96	96
2	c=4*2=8	funct1(8)	(8<30)?(True) c=100-8=92	92
3	c=4*3=12	funct1(12)	(12<30)?True c=100-12=88	88
4	c=4*4=16	funct1(16)	(16<30)?True c=100-16=84	84
5	c=4*5=20	funct1(20)	(20<30)? True c=100-20=80	80
6	c=4*6=24	funct1(24)	(24<30)?True c=100-24=76	76
7	c=4*7=28	funct1(28)	(28<30)?True c=100-28=72	72
8	c=4*8=32	funct1(32)	(32<30)?False c=b+x=200+32=232	232
9	c=4*9=36	funct1(36)	(36<30)?False c=b+x=200+36=236	236
10	c=4*10=40	funct1(40)	(40<30)?False c=b+x=200+40=240	240

Output:

```
96  
92  
88  
84  
80  
76  
72  
232  
236  
240
```

Describe the output generated by each of the following program.

```
#include<stdio.h>  
#include<conio.h>  
void func1(int n);  
main()  
{  
    int i;  
    clrscr();  
    for(i=1;i<=4;i++)  
    {  
        func1(i);  
    }  
    getch();  
}  
  
void func1(int n)  
{  
    int num=3;  
    printf("%d\n\n",n*num);  
}
```

Tracing

i	i<=4	func1(i)	n	num	print (n*num)
1	1<=4(T)	func1(1)	1	3	3
2	2<=4(T)	func1(2)	2	3	6
3	3<=4(T)	func1(3)	3	3	9
4	4<=4(T)	func1(4)	4	3	12
5	5<=4(F)	terminate			

Output:

```
3
6
9
12
```

Some programs using function

1. WAP to calculate the area of two circles having different radius using the same function area.

```
#include<stdio.h>
#include<conio.h>
float area(float r);
int main()
{
    float r1,r2,a1,a2;
    printf("Enter the radius of first circle");
    scanf("%f",&r1);
    a1=area(r1);
    printf("Enter the radius of second circle");
    scanf("%f",&r2)      ;
    a2=area(r2);
    printf("Area of first circle=%f",a1);
    printf("\nArea of second circle=%f",a2);
    getch();
}
float area(float r)
{
    float a;
    a=3.14*r*r;
    return a;
}
```

2. Calculate the sum of n natural numbers using function.

```
#include<stdio.h>
#include<conio.h>
int snatural(int n);
int main()
{
    int num,s;
    printf("Enter the value of n\n");
    scanf("%d",&num);
    s=snatural(num);
    printf("Sum of n natural number is %d",s);
    getch();
    return 0;
}
```

```

int snatural(int n)
{
    int i,sum=0;
    for(i=1;i<=n;i++)
    {
        sum=sum+i;
    }
    return sum;
}

```

3. WAP to calculate the area and perimeter of rectangle using function.

```

#include<stdio.h>
#include<conio.h>
void area(int,int);
void perimeter(int,int);
int main()
{
int l,b;
printf("Enter the length and breadth of rectangle\n");
scanf("%d%d",&l,&b);
area(l,b);
perimeter(l,b);
getch();
return 0;
}
void area(int x,int y)
{
    int area;
    area=x*y;
    printf("area=%d",area);
}

void perimeter(int x,int y)
{
int peri;
peri=2*(x+y);
printf("Perimeter=%d",peri) ;
}

```

4. WAP to check maximum of 3 numbers using function.

```
#include<stdio.h>
#include<conio.h>
int max(int,int,int);
int main()
{
int a,b,c,result;
printf("Enter the three numbers\n");
scanf("%d%d%d",&a,&b,&c);
result=max(a,b,c);
printf("Maximum number=%d",result);
getch();
return 0;
}

int max(int x,int y,int z)
{
if(x>y&&x>z)
return x;
else if(y>z)
return y;
else
return z;
}
```

WAP to check the maximum of two numbers using function.(Do yourself)

**5. WAP to check the given number is prime or not using user defined function.
[PU:2019 spring]**

```
#include<stdio.h>
#include<conio.h>
void prime(int n);
int main()
{
int num;
printf("Enter the number you want to check\n");
scanf("%d",&num);
prime(num);
getch();
return 0;
}
```

```

void prime(int n)
{
    int i;
    for(i=2;i<n;i++)
    {
        if(n%i==0)
        {
            printf("Number is not prime");
            break;
        }
    }

    if(i==n)
    {
        printf("Number is prime");
    }
}

```

6. WAP to check given number is palindrome or not using function.

```

#include<stdio.h>
#include<conio.h>
void palindrome(int n);
int main()
{
int num;
printf("Enter the number you want to check\n");
scanf("%d",&num);
palindrome(num);
getch();
return 0;
}

```

```

void palindrome(int n)
{
    int rem,rev=0,a;
    a=n;
    while(n!=0)
    {
        rem=n%10;
        rev=rev*10+rem;
        n=n/10;
    }

    if(a==rev)
    {
        printf("Number is palindrome");
    }
    else
    {
        printf("Number is not palindrome");
    }
}

```

WAP to find the reverse of a number using function.(Do yourself)

7. By using function WAP to generate the Fibonacci series upto nth term when initial value is given by user.

```

#include<stdio.h>
#include<conio.h>
void fibo(int n,int x,int y);
int main()
{
int a,b,num;
printf("Enter the number of terms you want to generate\n");
scanf("%d",&num);
printf("Enter the two initial values\n");
scanf("%d%d",&a,&b);
fibo(num,a,b);
getch();
return 0;
}

```

```

void fibo(int n,int x,int y)
{
    int i,c;
    printf("%d\t%d",x,y);
    for(i=1;i<=n-2;i++)
    {
        c=x+y;
        printf("\t%d",c);
        x=y;
        y=c;
    }
}

```

Passing Array to Function

Passing one dimensional Array to function

In C, arrays are automatically passed by reference to a function. The name of the array represents the address of its first element. By passing the array name, we are in fact, passing the address of the array to the called function. The array in the called function now refers to same array stored in the memory. Therefore any changes in the array in the called function will be reflected in the original array.

- The corresponding formal argument in function definition is written in the same manner, though it must be declared as an array.
- When declaring a one dimensional array as a formal argument, the array name is written with a pair of empty square brackets. The size of the array is not specified within the formal argument declaration.
- To pass an array as an argument to function, the array name must be specified, without brackets or subscripts, and the size of an array as arguments.

Syntax for function declaration that receives array as argument

```
return_type function_name(data_type array_name[ ] );
```

Syntax to pass array to the function.

```
function_name(arrayname);
```

Note: We cannot pass a whole array by value as we did in the case of ordinary variables

Program to illustrate passing array to a function one element at a time.

```
#include<stdio.h>
#include<conio.h>
void display(int x);
int main()
{
    int i;
    int a[5]={5,10,15,20,25};
    for(i=0;i<5;i++)
    {
        display(a[i]);
    }
    getch();
    return 0;
}

void display(int x)
{
printf("%d\t",x);
}
```

Write a program to pass one dimensional array to a function and display that array in that called function.[PU:2019 fall]

Program to illustrate passing an entire array to a function

```
#include<stdio.h>
#include<conio.h>
void display(int x[],int n);
int main()
{
    int i;
    int a[5]={5,10,15,20,25};
    display(a,5);
    getch();
    return 0;
}
void display(int x[],int n)
{
    int i;
    for(i=0;i<n;i++)
    {
        printf("%d\t",x[i]);
    }
}
```

1) Write a program that passes an array to function and print the largest and smallest element

```
#include<stdio.h>
#include<conio.h>
void fun(int x[],int n);
int main()
{
    int a[100],n,i;
    printf("Enter how many elements\n");
    scanf("%d",&n);
    printf("Enter the array elements\n");
    for(i=0;i<n;i++)
    {
        scanf("%d",&a[i]);
    }
    fun(a,n);
    getch();
    return 0;
}

void fun(int x[],int n)
{
    int i,small,large;
    large=x[0];
    small=x[0];
    for(i=0;i<n;i++)
    {
        if(large<x[i])
        {
            large=x[i];
        }
        if(small>x[i])
        {
            small=x[i];
        }
    }
    printf("Largest element =%d",large);
    printf("Smallest element=%d",small);
}
```

2) Write a program to read n number in an array and sort them in ascending order using function

```
#include<stdio.h>
#include<conio.h>
void sort(int x[],int n);
void disp(int d[],int m);
int main()
{
    int a[100],n,i;
    printf("Enter how many elements\n");
    scanf("%d",&n);
    printf("Enter the array elements\n");
    for(i=0;i<n;i++)
    {
        scanf("%d",&a[i]);
    }
    printf("\nArray elements before sorting are\n");
    disp(a,n);
    sort(a,n);
    printf("\nArray elements after sorting are\n");
    disp(a,n);
    getch();
    return 0;
}
```

```
void sort(int x[],int n)
{
    int i,j,temp;
    for(i=0;i<n-1;i++)
    {
        for(j=0;j<n-1-i;j++)
        {
            if(x[j]>x[j+1])
            {
                temp=x[j];
                x[j]=x[j+1];
                x[j+1]=temp;
            }
        }
    }
}
```

```

void disp(int d[],int m)
{
    int i;
    for(i=0;i<m;i++)
    {
        printf("%d\n",d[i]);
    }
}

```

- 3) Write a program to find the sum of all prime numbers in a given array. The main function of your program should take the help of user-defined function that tests whether a given number is prime or not.[PU -2013 Fall]

```

#include<stdio.h>
#include<conio.h>
int checkprime(int n);
int main()
{
    int a[50],n,i,sum=0,result;
    printf("Enter the no. of elements:");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        scanf("%d",&a[i]);
    }
    printf("\nprime numbers are:\n");
    for(i=0;i<n;i++)
    {
        result=checkprime(a[i]);
        if(result==1)
        {
            printf("%d\t",a[i]);
            sum=sum+a[i];
        }
    }
    printf("\nThe sum of prime no. is %d",sum);

    getch();
}

```

```

int checkprime(int n)
{
    int i;

    for ( i = 2 ; i <n ; i++ )
    {
        if ( n% i == 0 )
            return 0;
    }
    if ( n == i )
        return 1;
}

```

Practice Questions:

- 1) Write a program to input n numbers in an array and count number of odd and even numbers and find their sum using function.
- 2) WAP to input n number in an array and print in reverse order using function.
- 3) WAP to input n numbers in an array and check if given number is present or not using function.

Passing 2D array to function

- Just as in 1 dimensional array, when two dimensional array are passed as a parameter, the base address of the actual array is sent to function.
- Any change made to element inside function will carry over to the original location of array that is passed to function.
- The size of all dimensions except the first must included in the function prototype (declaration) and in function definition.

Eg. void display(int[][10],int,int);

is a valid declaration.

1) Write a program to input m*n order matrix and find sum of all elements using function.

```
#include<stdio.h>
#include<conio.h>
void summatrix(int s[][20],int x,int y);
void disp(int d[][20],int m,int n);
int i,j;
int main()
{
    int a[20][20],r,c;
    printf("Enter the row and column size of matrix\n");
    scanf("%d%d",&r,&c);
    printf("Enter the array elements\n");
    for(i=0;i<r;i++)
    {
        for(j=0;j<c;j++)
        {
            scanf("%d",&a[i][j]);
        }
    }
    disp(a,r,c);
    summatrix(a,r,c);
    getch();
    return 0;
}
```

```
void summatrix(int s[][20],int x,int y)
{
    int sum=0;
    for(i=0;i<x;i++)
    {
        for(j=0;j<y;j++)
        {
            sum=sum+s[i][j];
        }
    }
    printf("sum of all elements=%d",sum);
}
```

```

void disp(int d[][20],int m,int n)
{
    printf("Entered matrix is\n");
    for(i=0;i<m;i++)
    {
        for(j=0;j<n;j++)
        {
            printf("%d\t",d[i][j]);
        }
        printf("\n");
    }
}

```

2) Write a program to input m*n order matrix and find its transpose using function

```

#include<stdio.h>
#include<conio.h>
void transpose(int t[][20],int x,int y);
void disp(int d[][20],int m,int n);
int i,j;
int main()
{
    int a[20][20],r,c;
    printf("Enter the row and column size of matrix\n");
    scanf("%d%d",&r,&c);
    printf("Enter the array elements\n");
    for(i=0;i<r;i++)
    {
        for(j=0;j<c;j++)
        {
            scanf("%d",&a[i][j]);
        }
    }
    disp(a,r,c);
    transpose(a,r,c);
    getch();
    return 0;
}

```

```

void transpose(int t[][20],int x,int y)
{
    printf("Transpose of matrix is \n");
    for(i=0;i<y;i++)
    {
        for(j=0;j<x;j++)
        {
            printf("%d\t",t[j][i]);
        }
        printf("\n");
    }
}

void disp(int d[][20],int m,int n)
{
    printf("Entered matrix is\n");
    for(i=0;i<m;i++)
    {
        for(j=0;j<n;j++)
        {
            printf("%d\t",d[i][j]);
        }
        printf("\n");
    }
}

```

3. Addition of two m*n order matrix using function

```
#include<stdio.h>
#include<conio.h>
void input(int a[][20],int x,int y);
void disp(int d[][20],int m,int n);
void addition(int a1[][20],int a2[][20],int p,int q);
int i,j;
int main()
{
    int matrix1[20][20],matrix2[20][20],r,c;
    printf("Enter the row and column size of matrix\n");
    scanf("%d%d",&r,&c);
    printf("Enter the first matrix");
    input(matrix1,r,c);
    printf("Enter the second matrix");
    input(matrix2,r,c);
    printf("The first matrix is\n");
    disp(matrix1,r,c);
    printf("The second matrix is\n");
    disp(matrix2,r,c);
    addition(matrix1,matrix2,r,c);
    getch();
    return 0;
}
void input(int a[][20],int x,int y)
{
    for(i=0;i<x;i++)
    {
        for(j=0;j<y;j++)
        {
            scanf("%d",&a[i][j]);
        }
    }
}
```

```

void disp(int d[][20],int m,int n)
{
    for(i=0;i<m;i++)
    {
        for(j=0;j<n;j++)
        {
            printf("%d\t",d[i][j]);
        }
        printf("\n");
    }
}

void addition(int a1[][20],int a2[][20],int p,int q)
{
    int sum[20][20];
    for(i=0;i<p;i++)
    {
        for(j=0;j<q;j++)
        {
            sum[i][j]=a1[i][j]+a2[i][j];
        }
    }
    printf("Resultant matrix is\n");
    disp(sum,p,q);
}

```

Practice Questions.

1. WAP to input $m \times n$ order matrix and count even number of elements present in matrix and find their sum using function.
2. WAP to input $m \times n$ matrix and find highest and lowest element using function.
3. WAP to input $m \times n$ order matrix and find highest and lowest element using function.

CHAPTER-7

Pointer

What is a pointer in c? [PU: 2012 fall, 2014 fall, 2015 fall, 2019 fall]

Pointer is a variable that stores/points the address of another variable. Pointer variable can only contains the address of a variable of the same data type.

eg.

```
int *ptr; //pointer declaration  
int a=5;  
ptr=&a; //address of variable a is assigning to pointer variable ptr
```

Here ptr is a pointer variable that only contains the address of integer data type.

What are the advantages of using pointer?

- 1) Pointers are more efficient in handling arrays.
- 2) Pointers can be used to return multiple values from functions via function arguments.
- 3) The pointer enables us to access a variable that is defined outside function.
- 4) Pointer allows C to support dynamic memory management.
- 5) Pointers reduce length and complexity of programs.
- 6) They increase the execution speed and thus reduce the program execution time.
- 7) Pointers provide efficient tool for manipulating dynamic data structures such as structures, linked lists, queues, stack and trees.

Pointer declaration

Pointer variable is declared with an asterisk (*) operator before a variable name. This operator is called pointer/indirection or dereference operator.

Syntax: data_type *pointer_variable_name;

Data type of the pointer must be same as the data type of the variable to which the pointer variable is pointing.

eg. int *ptr;

Declares the variable ptr which is a pointer variable that points to an integer data type.

Similarly the statement

float *x;

Declares the variable x which is a pointer variable that points to an float data type.

Initialization of pointers

Pointer initialization is the process of assigning address of variable to a pointer variable .In C programming language ampersand (&) operator is used to determine the address of variable. The & (immediately preceding a variable name) returns the address of variable associated with it.

Pointer variable always points to variables of same data type.

```
int main()
{
    int a=10;
    int *ptr;    //pointer declaration
    ptr=&a;    //pointer initialization
}
```

Here, pointer variable ptr of integer type is pointing the address of integer variable a. So this is valid.

Let's have another example

```
int main()
{
    float a;
    int *ptr;
    ptr=&a;    //ERROR, type mismatch
}
```

Here, pointer variable of integer type ptr is pointing the address of float variable a. So this is invalid.

Once, address of variable is assigned to pointer, then to access the value of variable, pointer is dereferenced using indirection/dereference operator (*).

Example:

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int a=5;
    int *ptr;          //declaration of pointer variable
    ptr=&a;           //initializing the pointer
    printf("\n%d",a); //prints the value of a
    printf("\n%d",*ptr); //prints the value of a
    printf("\n%d",*(&a)); //prints the value of a
    printf("\n%u",&a); //prints the address of a
    printf("\n%u",ptr); //prints the address of a
    printf("\n%u",&ptr); //prints the address of pointer variable ptr
    getch();
    return 0;
}
```

Bad pointer

- When a pointer variable is declared and if any valid address is not assigned to it then pointer is known as bad pointer. A dereference operation on a bad pointer is a serious runtime error.
- Each pointer must be assigned a valid address before it can support dereference operations. Before that pointer is bad and must not be used.
- It is always a best practice to initialize a pointer NULL values, when they are declared and check for whether the pointer is NULL pointer when using the pointer.

Void pointer

Write a short notes on: Void pointer [PU: 2018 spring, 2016 Fall]

Void pointer is a special type of pointer that can point any data type (int or float or char or double). Using void pointer, the pointed data cannot be dereferenced (cannot access the value at the address stored in pointer variable). so that reason we will always have to change type of void pointer to some other pointer type that points to a concrete data type before dereferencing it. This is done by performing type-casting.

Declaration:

```
void *pointer_name;
```

Example:

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int a=10;
    float b=4.5;
    void *ptr;
    ptr=&a;
    printf("a=%d",*(int*)ptr);
    ptr=&b;
    printf("b=%f", *(float*)ptr);
    getch();
    return 0;
}
```

Output:

```
a=10
b=4.500000
```

Null pointer

- A Null pointer is a pointer which is pointing to nothing.
- Pointer which is initialized with NULL value is considered as NULL pointer.
- We can define a null pointer using a predefined constant NULL, which is defined in header files stdio.h , stddef.h, stdlib.h.
- Some of the most common use cases for NULL are
 - a) To initialize a pointer variable when that pointer variable isn't assigned any valid memory address yet.

```
int * ptr = NULL;
```
 - b) To check for null pointer before accessing any pointer variable. By doing so, we can perform error handling in pointer related code e.g. dereference pointer variable only if it's not NULL.

```
if(ptr != NULL) /*We could use if (ptr) as well*/  
{ /*Some code*/}  
else  
{ /*Some code*/}
```

Pointer Arithmetic

Write a short notes on: Pointer Arithmetic. [PU: 2016 spring]

As a pointer holds the memory address of variable, some arithmetic operations can be performed with pointers. They are as follows.

- Increment
- Decrement
- Addition
- Subtraction

If arithmetic operations done values will be incremented or decremented as per data type chosen. Let `ptr=&arr[0]=1000` i.e Base address of array.

int type pointer (2-byte space)	float type pointer (4-byte space)	char type pointer (1-byte pointer)
<code>ptr++=ptr+1=1000+2=1002</code> is the address of next element.	<code>ptr++=ptr+1=1000+4=1004</code> is the address of next element	<code>ptr++=ptr+1=1000+1=1001</code> is the address of next element
<code>ptr=ptr+4=1000+(2*4)=1008</code> ie. Address of 5 th integer type element (<code>&arr[4]</code>)	<code>ptr=ptr+4=1000+(4*4)=1016</code> ie. Address of 5 th float type element (<code>&arr[4]</code>)	<code>ptr=ptr+4=1000+(1*4)=1004</code> ie. Address of 5 th char type element(<code>&arr[4]</code>)

Note: Similar operation can be done for decrement.

Example for pointer increment/decrement

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int arr[5] = {10,20,30,40,50};
    int *ptr1,*ptr2;
    ptr1=&arr[0];
    ptr1++;
    printf("\nvalue %d has address %u",*ptr1,ptr1); //points to 2nd element
    ptr2=&arr[4];
    ptr2--;
    printf("\nvalue %d has address %u",*ptr2,ptr2); //points to 4th element
    getch();
    return 0;
}
```

Example for pointer addition/subtraction with integer constant

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int arr[5] = {10,20,30,40,50};
    int *ptr1,*ptr2;
    ptr1=&arr[0];
    ptr1=ptr1+4;
    printf("\nvalue %d has address %u",*ptr1,ptr1); //points to 5th element
    ptr2=&arr[4];
    ptr2=ptr2-4 ;
    printf("\nvalue %d has address %u",*ptr2,ptr2); //points to 1st element
    getch();
    return 0;
}
```

Some other Operations that can be performed on pointer

- 1) A pointer variable can be assigned the address of an ordinary variable.

```
int main()
{
    int a=5;
    int *ptr;
    ptr=&a;
    printf("value of a=%d",a);
    printf("Address of a=%u",ptr);
    return 0;
}
```

- 2) Content of one pointer variable can be assigned to other pointer variable provided they point to same data type.

```
int main()
{
    int arr[5]={1,2,3,4,5};
    int *ptr1,*ptr2;
    ptr1=&arr[0];
    ptr2=ptr1; //assign element of ptr1 to ptr2
    printf("ptr1 contains %u and ptr2 contains %u",ptr1,ptr2);
    return 0;
}
```

- 3) One pointer variable can be subtracted from another provided that both variables points to the element of same array.

```
int main()
{
    int arr[5]={1,2,3,4,5};
    int *ptr1,*ptr2;
    ptr1=arr;
    ptr2=&arr[4];
    printf("Difference of two pointer =%d",ptr2-ptr1);
    return 0;
}
```

- 4) Two pointers variables can be compared provided both pointers points to object of same data type.

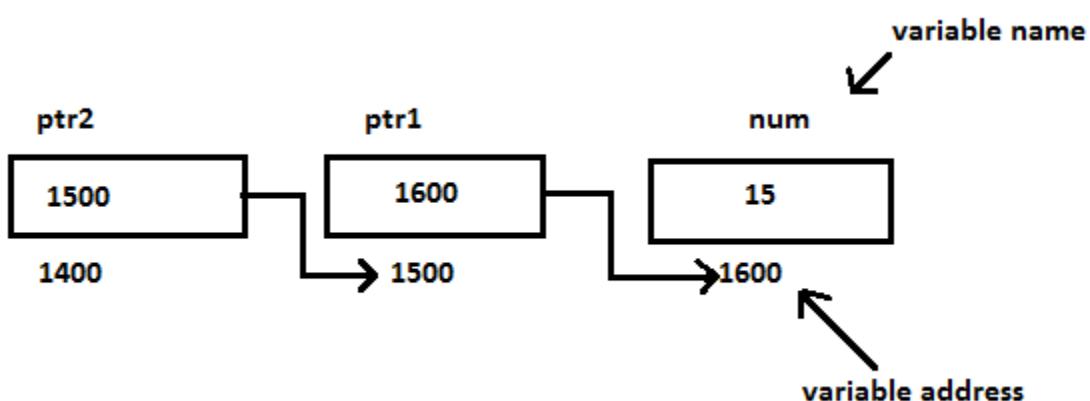
```
int main()
{
    int arr[5]={1,2,3,4,5};
    int *ptr1,*ptr2;
    ptr1=&arr[0];
    ptr2=&arr[4];
    if(ptr2>ptr1)
        printf("ptr2 is far from ptr1");
    else
        printf("ptr1 is far from ptr2");
    return 0;
}
```

Invalid pointer operations

- Addition of two pointer ($\text{ptr1} + \text{ptr2}$)
- Multiplication of two pointer ($\text{ptr1} * \text{ptr2}$)
- Addition or subtraction of float or double data type to or from a pointer
- Multiplication of pointer with constant ($\text{ptr1} * 5$)
- Division of two pointer or with constant ($\text{ptr1} / 5$)

Double pointer/chain of pointer /pointer to pointer/ Double indirection

When pointer holds the address of another pointer then such type of pointer is called double pointer /chain of pointer.



As per diagram, here ptr1 is a normal pointer that holds the address of an integer variable num. There is another pointer ptr2 in the diagram that holds the address of another pointer ptr1, the pointer ptr2 here is pointer to pointer(or double pointer)

A variable that is pointer to pointer must declared using two indirection operator symbol in front of name.e.g. int **ptr2;

Example

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int num=15;
    int *ptr1;
    int **ptr2;
    ptr1=&num;
    ptr2=&ptr1;
    printf("num=%d\n",*ptr1);
    printf("num=%d",**ptr2);
    getch();
    return 0;
}
```

Output

```
num= 15
num= 15
```

Passing pointer to function

Pointers can also be passed to a function as an argument. When we pass a pointer as an argument then the address of the variable is passed instead of the value. So any change made by the function using the pointer is reflected in actual argument that is passed through function. This mechanism of passing argument to function is known as call by reference.

Program to illustrate passing pointer to function

```
#include<stdio.h>
#include<conio.h>
void addGraceMarks(int *m);
int main()
{
    int marks;
    printf("Enter the actual marks\n");
    scanf("%d",&marks);
    addGraceMarks(&marks);
    printf("The final marks is:%d",marks);
    getch();
    return 0;
}

void addGraceMarks(int *m)
{
    *m=*m+10;
}
```

Program to convert uppercase letter into lower and vice versa passing pointer to function

```
#include<stdio.h>
#include<conio.h>
void conversion(char *);
int main()
{
    char ch;
    printf("Enter the character\n");
    scanf("%c",&ch);
    conversion(&ch);
    printf("The corresponding character is:%c",ch);
    getch();
    return 0;
}

void conversion(char *c)
{
    if(*c>='a'&&*c<='z')
        *c=*c-32;
    else if (*c>='A'&&*c<='Z')
        *c=*c+32;
}
```

What are the advantages of using pointer in a function? [PU: 2009 spring, 2011 spring]

The advantages of using pointers in a functions are:

- Address of variable can be passes to the function so that changes to value of arguments in calling function can be easily reflected in called function.
- In general function cannot return more than one values, But it can be possible using concept of pointers.
- Arrays and structures can be passes to the function efficiently.
- It is possible to pass a portion of an array rather than an entire array to a function using pointer.
- The use of pointer as a function arguments permits the corresponding data items to be altered globally from within the function.

Write a program with user defined function using pointer to convert all the upper case to lower case and vice-versa in string given by the user. [PU-2012 Fall]

```
#include<stdio.h>
#include<conio.h>
void conversion(char *s);
int main( )
{
    char str[50];
    printf("Enter the string\n");
    gets(str);
    conversion(str);
    printf("String after conversion");
    puts(str);
    getch();
    return 0;
}
void conversion(char *s)
{
    int i;
    for(i=0; *(s+i) != '\0'; i++)
    {
        if(*(s+i)>='a'&&*(s+i)<='z')
        {
            *(s+i)=*(s+i)-32;
        }
        else if(*(s+i)>='A'&&*(s+i)<='Z')
        {
            *(s+i)=*(s+i)+32;
        }
    }
}
```

Returning multiple values from function

- Can function return more than one value? Justify your answer with examples [PU: 2012 fall]
- Does function return single or multiple value? When and how a function will return single or multiple value? Illustrate with examples.[PU:2017 fall,2013 fall]
- How can function return multiple values? Explain with example? [PU: 2018 spring,PU:2019 fall]

Normally, (when function arguments are passed by value) more than one value cannot be returned at a time using function. But when we use pointer in function we can return more than one value.

When we pass the function argument with their address and make changes in their value using pointer, then any change made by the function is reflected in actual argument that is passed through function. Hence we can pass any number of parameters as reference, which we want to return from function.

Example:

```
#include<stdio.h>
#include<conio.h>
void areaperi(int r,float *area,float *peri);
int main()
{
    int rad;
    float a,p;
    printf("Enter the radius\n");
    scanf("%d",&rad);
    areaperi(rad,&a,&p);
    printf("Area of circle=%f",a);
    printf("Perimeter of circle=%f",p);
    getch();
    return 0;
}
void areaperi(int r,float *area,float *peri)
{
    *area=3.14*r*r;
    *peri=2*3.14*r;
}
```

Pointer and arrays

Pointer and 1D-Array

When the array is declared,

- Compiler allocates the sufficient amount of memory to contain all the elements of array in contiguous memory location.
- Base address ie. address of the first element of the array is allocated by the compiler.

Suppose, we declare an array arr.

```
int arr[5]={5,10,15,20,25};
```

Assuming that base address of arr is 1000 and each integer requires two bytes, the five elements will be stored as follows.

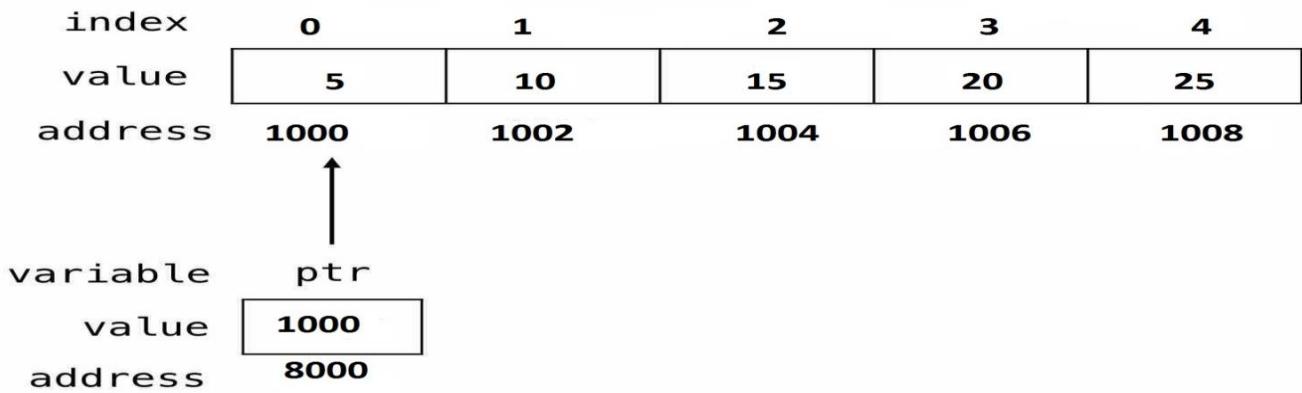
Index	0	1	2	3	4
Value	5	10	15	20	25
Address	1000	1002	1004	1006	1008

Here variable **arr** holds the address of the first element of the array.i.e. Points at the starting memory of address.

So, arr contains the address of arr[0] ie.1000

If we want to declare ptr as integer pointer then we can make the pointer ptr to point the array arr by following assignment.

```
int *ptr;  
ptr=arr; OR ptr=&arr[0];
```



Now we can access every element of array arr by incrementing the value of ptr to move one element to another.

Illustration:

```
int arr[5]={5,10,15,20,25};
```

```
int *ptr, i;
```

```
ptr=arr; //similar to ptr=&arr[0]
```

By assuming array starts at location 1000

&arr[0]=1000	arr[0]=5	ptr+0=1000	*(ptr+0)=5
&arr[1]=1002	arr[1]=10	ptr+1=1002	*(ptr+1)=10
&arr[2]=1004	arr[2]=15	ptr+2=1004	*(ptr+2)=15
&arr[3]=1006	arr[3]=20	ptr+3=1006	*(ptr+3)=20
&arr[4]=1008	arr[4]=25	ptr+4=1008	*(ptr+4)=25

Thus

Accessing value	Accessing Address
printf("%d", *(ptr+i)); //displays the arr[i] value	printf("%u", (ptr+i)); //displays the address of arr[i]
printf("%d", *ptr); //display the arr[0] value	printf("%u", ptr); //display the address of arr[0]
printf("%d", *(arr+i)); //displays the arr[i] value	printf("%u", (arr+i)); //displays the address of arr[i]

Program to illustrate accessing Array elements using pointer

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int i;
    int arr[5]={1,2,3,4,5};
    int *ptr=arr;
    for(i=0;i<5;i++)
    {
        printf("\nArray element=%d and Memory location=%u",*(ptr+i), (ptr+i));
    }
    getch();
    return 0;
}
```

WAP to input n number in array and display it using pointer.

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int arr[100],i,n,*ptr;
    ptr=arr;
    printf("Enter the number of elements you want to enter");
    scanf("%d",&n);
    printf("\nEnter %d elements",n);
    for(i=0;i<n;i++)
    {
        scanf("%d",(ptr+i));
    }
    printf("Entered elements are\n");
    for(i=0;i<n;i++)
    {
        printf("%d",*(ptr+i));
    }
    getch();
    return 0;
}
```

WAP to input 5 number in array and display it using pointer. (Do yourself)

- WAP to input n number in array and find sum of all elements using pointer.
[PU: 2012 fall]
- WAP using pointer to read an array of integers. Next add the elements in the array and display sum on screen. **[PU:2015 fall]**

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int arr[100],n,i,*ptr,sum=0;
    ptr=arr;
    printf("Enter number of elements you want to enter");
    scanf("%d",&n);
    printf("Enter %d elements",n)      ;
    for(i=0;i<n;i++)
    {
        scanf("%d",(ptr+i));
    }
    for(i=0;i<n;i++)
    {
        sum=sum+*(ptr+i);
    }
    printf("The sum of all elements is %d",sum);
    getch();
    return 0;
}
```

Write a program to sort n integer values in an array using pointer.
[PU: 2014 fall]

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int arr[100],n,i,j,temp,*ptr;
    ptr=arr;
    printf("Enter number of elements you want to enter");
    scanf("%d",&n);
    printf("Enter %d elements",n);
    for(i=0;i<n;i++)
    {
        scanf("%d",(ptr+i));
    }
    for(i=0;i<n-1;i++)
    {
        for(j=0;j<n-i-1;j++)
        {
            if(*(ptr+j)>*((ptr+j)+1))
            {
                temp=*(ptr+j);
                *(ptr+j)=*((ptr+j)+1);
                *((ptr+j)+1)=temp;
            }
        }
    }
    printf("\nThe sorted array are");
    for(i=0;i<n;i++)
    {
        printf("\n%d",*(ptr+i));
    }

    getch();
    return 0;
}
```

- Write a program using pointers to read in an array of integers and prints its elements in reverse order. [PU: 2013 spring]
- WAP to input n number in an array and print in reverse order using pointer.

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int arr[100],i,n,*ptr;
    ptr=arr;
    printf("Enter number of elements you want to enter");
    scanf("%d",&n);
    printf("Enter %d elements",n)      ;
    for(i=0;i<n;i++)
    {
        scanf("%d",*(ptr+i));
    }

    printf("\nThe array elements are");
    for(i=0;i<n;i++)
    {
        printf("\n%d",*(ptr+i));
    }
    printf("\nThe array elements in reverse order are");
    for(i=n-1;i>=0;i--)
    {
        printf("\n%d",*(ptr+i));
    }
    getch();
    return 0;
}
```

WAP to input n number in an array and find highest and lowest element using pointer

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int arr[100],n,i,*ptr,high,low;
    ptr=arr;
    printf("Enter number of elements you want to enter");
    scanf("%d",&n);
    printf("Enter %d elements",n);
    for(i=0;i<n;i++)
    {
        scanf("%d",(ptr+i));
    }
    high=*ptr;
    low= *ptr;
    for(i=0;i<n;i++)
    {
        if(high<*(ptr+i))
            high=*(ptr+i);
        if(low>*(ptr+i))
            low=*(ptr+i);
    }
    printf("Highest element=%d and lowest element=%d",high,low);
    getch();
    return 0;
}
```

WAP to input n number in an array and find sum of odd and even elements and count them using pointer

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int arr[100],n,i,*ptr,esum=0,ecount=0,osum=0,octount=0;
    ptr=arr;
    printf("Enter number of elements you want to enter");
    scanf("%d",&n);
    printf("Enter %d elements",n)      ;
    for(i=0;i<n;i++)
    {
        scanf("%d",(ptr+i));
    }

    for(i=0;i<n;i++)
    {
        if(*(ptr+i)%2==0)
        {
            esum=esum+*(ptr+i);
            ecount++;
        }
        else
        {
            osum=osum+*(ptr+i) ;
            octount++;
        }
    }
    printf("sum of even elements=%d and number of even elements=%d",esum,ecount);
    printf("\nsum of odd elements=%d and number of odd elements=%d",osum,octount);
    getch();
    return 0;
}
```

WAP to input n number in an array and check the given number is present or not using pointer

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int arr[100],n,i,*ptr,num,flag=0;
    ptr=arr;
    printf("Enter number of elements you want to enter");
    scanf("%d",&n);
    printf("Enter %d elements",n)      ;
    for(i=0;i<n;i++)
    {
        scanf("%d",(ptr+i));
    }
    printf("Enter a number you want to search");
    scanf("%d",&num);
    for(i=0;i<n;i++)
    {
        if(*(ptr+i)==num)
        {
            flag=1;
            break;
        }
    }
    if(flag==1)
    {
        printf("Your number is found");
    }
    else
    {
        printf("Your number is not found");
    }
    getch();
    return 0;
}
```

Pointers and two dimensional Arrays

The elements of 2-D array can be accessed with the help of pointer notation also.

Let us take a two dimensional array arr[3][4]:

```
int arr[3][4] = { {1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12} };
```

	Col 1	Col 2	Col 3	Col 4
Row 1	1	2	3	4
Row 2	5	6	7	8
Row 3	9	10	11	12

Since memory in computer is organized linearly it is not possible to store the 2-D array in rows and columns. The concept of rows and columns is only theoretical, actually a 2-D array is stored in row major order i.e rows are placed next to each other. The following figure shows how the above 2-D array will be stored in memory.

row-wise memory allocation

	<— row 0 —>				<— row 1 —>				<— row 2 —>			
value	1	2	3	4	5	6	7	8	9	10	11	12
address	1000	1002	1004	1006	1008	1010	1012	1014	1016	1018	1020	1022

↑
Base address ie. address of first element of array arr

Each row of the 2-D array treated as a 1-D array, so a two-dimensional array can be considered as a collection of one-dimensional arrays that are placed one after another.

So here arr is an array is an array of 3 elements where each element is a 1-D array of 4 integers.

Syntax for declaration of 2-D array using pointer notation

```
datatype(*ptr_variable)[size2];
```

(Instead of `data_type [size1][size2];` in normal array)

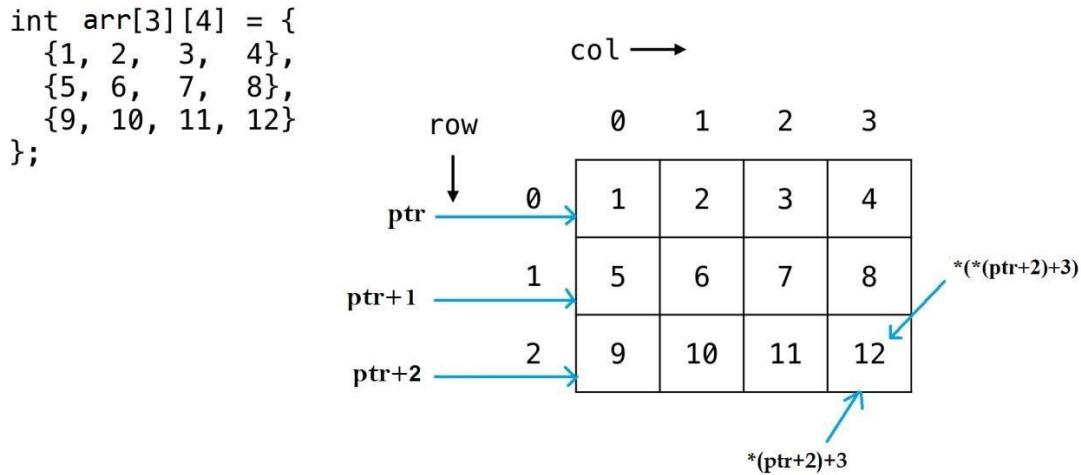
Eg. Let us suppose arr is a 2-D integer array having 3 rows and 4 columns, we can declare arr as

```
int(*ptr)[4];
```

Here, ptr is a pointer to an array of 4 integers

According to pointer arithmetic, in general

<code>ptr</code>	Pointer to first row
<code>ptr+i</code>	Pointer to <i>i</i> th row
<code>*(ptr+i)</code>	Pointer to first element in the <i>i</i> th row
<code>*(ptr+i)+j</code>	Pointer to the <i>j</i> th element in the <i>i</i> th row
<code>*(*(ptr+i)+j)</code>	Value stored in the cell(<i>i,j</i>) (<i>i</i> th row and <i>j</i> th column)



<code>&arr[0][0]=*(ptr+0)+0=**ptr</code>	<code>arr[0][0]=*(*(ptr+0)+0)=**ptr</code>
<code>&arr[0][1]=*ptr+1</code>	<code>arr[0][1]=*(*(ptr+0)+1)=*(ptr+1))</code>
<code>&arr[1][1]=*(ptr+1)+1</code>	<code>arr[1][1]= *(*(ptr+1)+1)</code>
<code>&arr[2][3]=*(ptr+2)+3</code>	<code>arr[2][3]=*(*(ptr+2)+3)</code>

Example:

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int arr[3][4] ={{1,2,3,4},{5,6,7,8},{9,10,11,12}};
    int i, j;
    int (*ptr)[4];
    ptr = arr;          //&arr[0][0]
    for(i = 0; i < 3; i++)
    {
        for(j = 0; j < 4; j++)
        {
            printf("\narr[%d][%d]=%d", i, j, *( *(ptr + i) + j));
        }
        printf("\n");
    }
    getch();
    return 0;
}
```

Note: As array name holds the base address (i.e. address of first element of array).Then, if arr is a 2D array we can access any element arr[i][j] of the array using the pointer expression.
 $*(\text{arr} + i + j)$

Example:

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int arr[2][3] ={{11,12,13},{14,15,16}};
    int i, j;
    printf("Contents of array are\n");
    for(i = 0; i < 2; i++)
    {
        for(j = 0; j < 3; j++)
        {
            printf("\narr[%d][%d]=%d", i, j, *( *(arr + i) + j));
        }
        printf("\n");
    }
    getch();
    return 0;
}
```

WAP to add two matrix of order m*n by using the concept of pointer.

```
#include<stdio.h>
#include<conio.h>
int main()
{
int a[20][20],b[20][20],c[20][20],i,j,m,n;
printf("Enter the value of m & n:");
scanf("%d%d",&m,&n);
printf("Enter the first matrix:");
for(i=0;i<m;i++)
{
    for(j=0;j<n;j++)
    {
        scanf("%d",*(a+i)+j));
    }
}
printf("\nEnter second matrix:");
for(i=0;i<m;i++)
{
    for(j=0;j<n;j++)
    {
        scanf("%d",*(b+i)+j));
    }
}
printf("\nAddition of two Matrix:\n");
for(i=0;i<m;i++)
{
    for(j=0;j<n;j++)
    {
        *(c+i)+j)=*(a+i)+j)+ *(b+i)+j);
        printf("%d ",*(c+i)+j));
    }
    printf("\n");
}
getch();
return 0;
}
```

WAP to input m*n matrix and find the sum of all elements.

```
#include<stdio.h>
#include<conio.h>
int main()
{
int arr[20][20],i,j,m,n,sum=0;
printf("Enter the value of m & n:");
scanf("%d%d",&m,&n);
printf("Enter the matrix:");
for(i=0;i<m;i++)
{
    for(j=0;j<n;j++)
    {
        scanf("%d",(*(arr+i)+j));
    }
}
for(i=0;i<m;i++)
{
    for(j=0;j<n;j++)
    {
        sum=sum+*(arr+i)+j);
    }
}
printf("sum of matrix is %d",sum);
getch();
return 0;
}
```

Array of pointers

- Array of pointers is a collection of address.
- The address present in the array of pointers can be address of variable or address of array element.

Array of pointers can be declared as:

```
datatype *pointer_name[size];
eg.int *ptr[4];
```

This statement declares an array of 4 pointers, each of which points to an integer value. The first pointer is called ptr[0] ,the second is ptr[1] ,third is ptr[2] and fourth is ptr[3].

Example:

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int a=5,b=6,c=7,d=8,i;
    int *ptr[4];
    ptr[0]=&a;
    ptr[1]=&b;
    ptr[2]=&c;
    ptr[3]=&d;
    for(i=0;i<4;i++)
    {
        printf("%d",*ptr[i]);
    }
    getch();
    return 0;
}
```

WAP to input 5 integer numbers and add all elements using array of pointer

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int *ptr[5];
    int arr[5],i,sum=0;
    for(i=0;i<5;i++)
    {
        ptr[i]=&arr[i];
    }
    printf("Enter 5 elements\n");
    for(i=0;i<5;i++)
    {
        scanf("%d",&arr[i]);
    }
    for(i=0;i<5;i++)
    {
        sum=sum+*ptr[i];
    }
    printf("sum=%d",sum);
    getch();
    return 0;
}
```

Pointer to string

As the string is an array of characters, the name of string variable is pointer to the first character of the string and can be used to access and manipulate the characters.

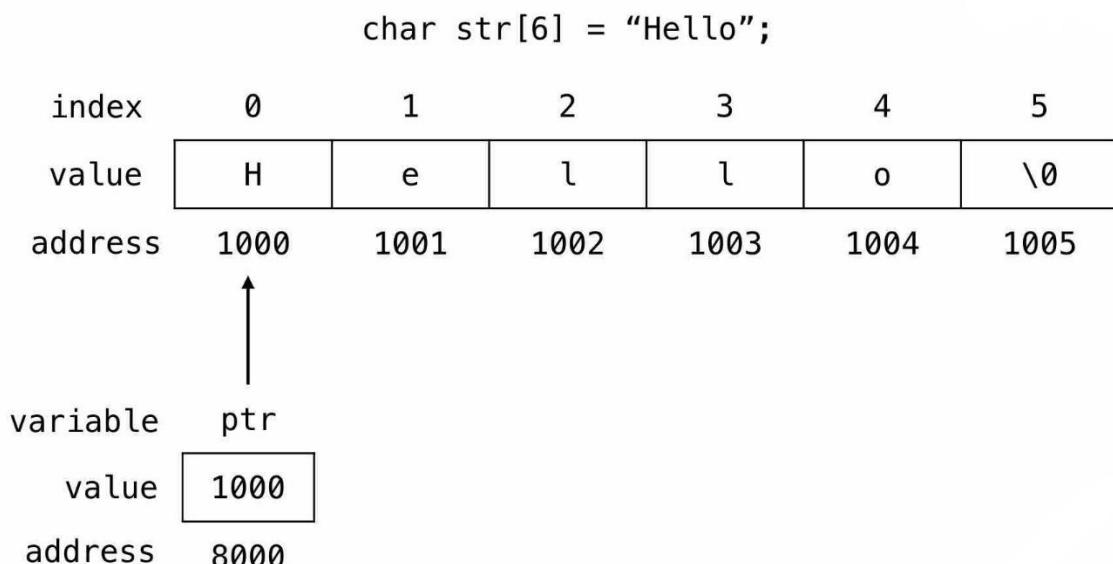
eg.

```
char str[6] = "Hello";
```

```
char *ptr = str; // assigning the address of the string str to the pointer ptr.
```

Here, the variable name of the string str holds the address of the first element of the array i.e., it points at the starting memory address.

We can represent the character pointer variable ptr as follows.



The pointer variable `ptr` is allocated memory address 8000 and it holds the base address of the string variable `str` i.e., 1000.

Accessing string via pointer

To access and print the elements of the string we can use a loop and check for the \0 null character. In the following example we are using while loop to print the characters of the string variable str.

```
#include <stdio.h>
#include<conio.h>
int main()
{
    char str[6] = "Hello"; // string variable
    char *ptr = str;      // pointer variable
    while(*ptr != '\0')
    {
        printf("%c", *ptr); // print the string
        ptr++;             // move the ptr pointer to the next memory location
    }
    getch();
    return 0;
}
```

Difference between array and pointer

Array	Pointer
1. Array is a collection of similar data items under a common variable name.	1. Pointer is a variable that points/stores the address of another variable.
2. An array name represents a fixed memory address that cannot be changed. We can index it, but we can't change the address it refers to.	2. As pointer is a variable contains address, we can change the address stored in that variable to point to something else.
3. <i>Declaration:</i> data_type array_name[size]; eg. int arr[10];	3. <i>Declaration:</i> data_type *pointer_variable_name; eg. int *ptr;
4. We cannot use name of array as variable. eg. int arr[10]; arr=ptr; // is not allowed	4. In pointer variable we can assign the address of another variable. eg. int *ptr; ptr=arr; // is allowed
5. An array can store the number of elements, mentioned in the size of array variable	5. A pointer variable can store the address of only one variable at a time.

Passing array element to a function using pointer

Write a program to pass array element to a function using pointer.[PU:2019 fall]

OR

Write a program to input n number in an array and display it using function and pointer.

```
#include<stdio.h>
#include<conio.h>
void display(int *p,int m);
int main()
{
int arr[100];
int *ptr,n,i;
ptr=arr;
printf("Enter the number of elements\n");
scanf("%d",&n);
printf("Enter %d elements\n",n);
for(i=0;i<n;i++)
{
    scanf("%d",(ptr+i));
}
printf("\nThe array elements are:\n");
display(arr,n);
getch();
return 0;
}
void display(int *p,int m)
{
    int i;
    for(i=0;i<m;i++)
    {
        printf("%d\t",*(p+i));
    }
}
```

WAP to input n number in an array and find sum of all elements using function and pointer.

```
#include<stdio.h>
#include<conio.h>
void display(int *p,int m);
void sumarray(int *p,int n);
int main()
{
int arr[100];
int *ptr,n,i;
ptr=arr;
printf("Enter the number of elements\n");
scanf("%d",&n);
printf("Enter %d elements\n",n);
for(i=0;i<n;i++)
{
    scanf("%d",(ptr+i));
}
printf("\nThe array elements are:\n");
display(arr,n);
sumarray(arr,n);
getch();
return 0;
}
void sumarray(int *p,int n)
{
    int i,sum=0;
    for(i=0;i<n;i++)
    {
        sum=sum+*(p+i);
    }
    printf("\nsum of all elements=%d",sum);
}
void display(int *p,int m)
{
    int i;
    for(i=0;i<m;i++)
    {
        printf("%d\t",*(p+i));
    }
}
```

WAP to input n number in an array and find the largest element using function and pointer.

```
#include<stdio.h>
#include<conio.h>
void display(int *p,int m);
void largest(int *p,int n);
int main()
{
int arr[100];
int *ptr,n,i;
ptr=arr;
printf("Enter the number of elements\n");
scanf("%d",&n);
printf("Enter %d elements\n",n);
for(i=0;i<n;i++)
{
    scanf("%d",(ptr+i));
}
printf("\nThe array elements are:\n");
display(arr,n);
largest(arr,n);
getch();
return 0;
}
void largest(int *p,int n)
{
    int large,i;
    large=*p;
    for(i=0;i<n;i++)
    {
        if(large<*(p+i))
        {
            large=*(p+i);
        }
    }
    printf("\nlargest element=%d",large);
}
void display(int *p,int m)
{
    int i;
    for(i=0;i<m;i++)
    {
        printf("%d\t",*(p+i));
    }
}
```

WAP to input n number in an array and sort them using function and pointer.

```
#include<stdio.h>
#include<conio.h>
void display(int *p,int m);
void sort(int *p,int n);
int main()
{
int arr[100];
int *ptr,n,i;
ptr=arr;
printf("Enter the number of elements\n");
scanf("%d",&n);
printf("Enter %d elements\n",n);
for(i=0;i<n;i++)
{
    scanf("%d",(ptr+i));
}
printf("\nThe array elements before sorting are:\n");
display(arr,n);
sort(arr,n);
printf("\nThe array elements after sorting are:\n");
display(arr,n);
getch();
return 0;
}
void sort(int *p,int n)
{
    int i,j,temp;
    for(i=0;i<n-1;i++)
    {
        for(j=0;j<n-i-1;j++)
        {
            if(*(p+j)>*((p+j)+1))
            {
                temp=*(p+j);
                *(p+j)=*((p+j)+1);
                *((p+j)+1)=temp;
            }
        }
    }
}
```

```

void display(int *p,int m)
{
    int i;
    for(i=0;i<m;i++)
    {
        printf("%d\t",*(p+i));
    }
}

```

Dynamic memory allocation

- What is dynamic memory allocation? Explain the different functions used in dynamic memory allocation? **[PU: 2018 fall]**
- Explain dynamic memory allocation. **[PU: 2014 spring]**
- How memory of a variable can be initialized dynamically? Explain with example. **[PU: 2014 fall]**
- How dynamic memory allocation can be achieved? Explain with suitable examples. **[PU:2015 spring,2016 spring,2019 fall]**

- The process of allocating and releasing memory at runtime is known as Dynamic memory allocation.
- Using DMA memory space is reserved during program execution and release space when no longer required.

Functions used in Dynamic Memory allocation

There are four library functions: malloc(),calloc(), free() and realloc() are for dynamic memory management. These functions are defined within header file stdlib.h.

1) malloc()

- The malloc() function reserves a block of memory of specified size and returns a pointer of type void, which can be casted into pointer of any form.
- The memory allocation can fail if the space in heap is not sufficient to satisfy the request. If it fails it returns NULL.

Syntax:

ptr =(cast-type*)malloc(byte-size);
ptr is pointer of type cast-type.

eg.

```

int *ptr;
ptr=(int*)malloc(100*sizeof(int));

```

A memory space equivalent to “100 times the size of integer” is reserved and address of the first memory allocated is assigned to pointer ptr of type int.

2) **calloc()**

`calloc()` allocates multiple block of storage, each of same size. All bytes are initialized to zero and pointer to the first byte of allocated region is returned. If there is not enough space, a NULL pointer is returned.

Syntax:

```
ptr=(cast-type*) calloc(n, element-size);
```

eg.

```
int *ptr;
```

```
ptr=(int*) calloc(25,sizeof(int));
```

This statement allocates contiguous space in memory for 25 elements each with size of int.

3) **realloc()**

This function is used to modify the size of previously allocated space.

Syntax:

```
ptr=realloc(ptr,newsize);//Reallocation of space
```

Sometimes previously allocated space is not sufficient we need to additional space and sometimes allocated memory is much larger than necessary. In both situation, we can change the memory size already allocated with the help of function realloc()

This function allocates the new memory space of size newsize to the pointer variable ptr and returns a pointer to the first byte of new memory.

4) **free()**

It releases the previously allocated space by `malloc()`, `calloc()`, and the `realloc()` function.

Its syntax is

```
free(ptr);
```

The memory dynamically allocated is not returned to the system until the programmer returns the memory explicitly. This can be done by using `free()` fuction. This function is used to release the space when it is not required.

Where ptr is a pointer to memory block which has already been created by `malloc()`, `calloc()` or `realloc()` function.

Example:

Program to dynamically allocate memory for the array elements using calloc() function and read and display the array elements.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int main()
{
    int *ptr;
    int i,n;
    printf("Enter number of elements to be entered\n");
    scanf("%d",&n);
    ptr=(int*)calloc(n,sizeof(int));
    if(ptr == NULL)
    {
        printf("Error! memory not allocated.");
        exit(0);
    }
    printf("Enter %d numbers\n",n);
    for(i=0;i<n;i++)
    {
        scanf("%d",(ptr+i));
    }
    printf("The elements are\n");
    for(i=0;i<n;i++)
    {
        printf("\n%d",*(ptr+i));
    }
    free(ptr);
    getch();
    return 0;
}
```

Perform similar operations using malloc()

Hint:

```
ptr=(int*)malloc(n*sizeof(int));
```

Program to illustrate the use of realloc() function

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int main()
{
    int *ptr = (int *)malloc(sizeof(int)*2);
    int i;
    int *ptrnew;
    *ptr = 10;
    *(ptr + 1) = 20;
    ptrnew = (int *)realloc(ptr, sizeof(int)*3);
    *(ptrnew + 2) = 30;
    for(i = 0; i < 3; i++)
    {
        printf("%d\t", *(ptrnew + i));
    }
    getch();
    return 0;
}
```

Output:

10 20 30

Program to find sum of n elements of an array using Dynamic memory allocation.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int main()
{
    int *ptr;
    int i,n,sum=0;
    printf("Enter number of elements to be entered\n");
    scanf("%d",&n);
    ptr=(int*)calloc(n,sizeof(int));
    if(ptr == NULL)
    {
        printf("Error! memory not allocated.");
        exit(0);
    }
```

```

printf("Enter %d numbers\n",n);
for(i=0;i<n;i++)
{
    scanf("%d",(ptr+i));
}
for(i=0;i<n;i++)
{
    sum=sum+*(ptr+i);
}
printf("The sum of all elements is %d",sum);
free(ptr);
getch();
return 0;
}

```

Write a program to find the sum of 5 numbers supplied by users using DMA. [PU: 2016 Fall]

```

#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int main()
{
    int *ptr;
    int i,n,sum=0;
    ptr=(int*)calloc(5,sizeof(int));
    if(ptr == NULL)
    {
        printf("Error! memory not allocated.");
        exit(0);
    }
    printf("Enter 5 numbers\n",n);
    for(i=0;i<5;i++)
    {
        scanf("%d",(ptr+i));
    }
    for(i=0;i<5;i++)
    {
        sum=sum+*(ptr+i);
    }
    printf("The sum of 5 numbers is %d",sum);
    free(ptr);
    getch();
    return 0;
}

```

Write a program to sort an array using dynamic memory allocation.(ascending order)
[PU: 2016 fall]

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int main()
{
    int *ptr;
    int i,j,temp,n;
    printf("Enter number of elements to be entered\n");
    scanf("%d",&n);
    ptr=(int*)calloc(n,sizeof(int));
    if(ptr == NULL)
    {
        printf("Error! memory not allocated.");
        exit(0);
    }
    printf("Enter %d numbers\n",n);
    for(i=0;i<n;i++)
    {
        scanf("%d",(ptr+i));
    }
    printf("The array before sorting are\n");
    for(i=0;i<n;i++)
    {
        printf("\n%d",*(ptr+i));
    }

    for(i=0;i<n-1;i++)
    {
        for(j=0;j<n-1-i;j++)
        {
            if(*(ptr+j)>*((ptr+j)+1))
            {
                temp=*(ptr+j);
                *(ptr+j)=*((ptr+j)+1);
                *((ptr+j)+1)=temp;
            }
        }
    }
}
```

```

printf("The sorted elements are\n");
for(i=0;i<n;i++)
{
    printf("\n%d",*(ptr+i));
}
free(ptr);
getch();
return 0;
}

```

Write a program to read 'n' numbers dynamically and sort it in descending order.

Hint:

```

for(i=0;i<n-1;i++)
{
    for(j=0;j<n-1-i;j++)
    {
        if(*(ptr+j)<*((ptr+j)+1))
        {
            temp=*(ptr+j);
            *(ptr+j)=*((ptr+j)+1);
            *((ptr+j)+1)=temp;
        }
    }
}

```

Write a program to print reverse element of an array using dynamic memory allocation.

[2013 fall]

```

#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int main()
{
    int *ptr;
    int i,n;
    printf("Enter number of elements to be entered\n");
    scanf("%d",&n);
    ptr=(int*)calloc(n,sizeof(int));
    if(ptr == NULL)
    {
        printf("Error! memory not allocated.");
        exit(0);
    }
}

```

```

printf("Enter %d numbers\n",n);
for(i=0;i<n;i++)
{
    scanf("%d",(ptr+i));
}

printf("Entered array elements are\n");
for(i=0;i<n;i++)
{
    printf("\n%d",*(ptr+i));
}
printf("Array elements in reverse order are\n");
for(i=n-1;i>=0;i--)
{
    printf("\n%d",*(ptr+i));
}
free(ptr);
getch();
return 0;
}

```

WAP to find the highest and lowest element of an array using DMA.

```

#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int main()
{
    int *ptr;
    int i,n,large,small;
    printf("Enter number of elements to be entered\n");
    scanf("%d",&n);
    ptr=(int*)calloc(n,sizeof(int));
    if(ptr == NULL)
    {
        printf("Error! memory not allocated.");
        exit(0);
    }
    printf("Enter %d numbers\n",n);
    for(i=0;i<n;i++)
    {
        scanf("%d",(ptr+i));
    }

```

```

printf("Entered array elements are\n");
for(i=0;i<n;i++)
{
    printf("\n%d",*(ptr+i));
}

large=*ptr;
small=*ptr;

for(i=0;i<n;i++)
{
    if(large<*(ptr+i))
        large=*(ptr+i);
    if(small>*(ptr+i))
        small=*(ptr+i);
}
printf("largest element =%d",large);
printf("smallest element =%d",small);
free(ptr);
getch();
return 0;
}

```

Assignment:

- Write a program to input n numbers in an array and find the sum of all even and odd numbers and count them using DMA.
- WAP to check if any given number is present in an array or not using DMA.

Write a program to find highest number in an array using dynamic memory allocation and function.

```

#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int i;
void display(int *p,int m);
void largest(int *ptr,int y);

```

```

int main()
{
    int *ptr;
    int i,n;
    printf("Enter the value of n\n");
    scanf("%d",&n);
    ptr=(int*)malloc(n*sizeof(int));
    printf("Enter %d elements\n",n);
    for(i=0;i<n;i++)
    {
        scanf("%d",*(ptr+i));
    }
    printf("\nThe elements are:\n");
    display(ptr,n);
    largest(ptr,n);
    free(ptr);
    getch();
}
void display(int *p,int m)
{
    for(i=0;i<m;i++)
    {
        printf("%d\t",*(p+i));
    }
}
void largest(int *ptr,int y)
{
    int large=*ptr;
    for(i=0;i<y;i++)
    {
        if (large<*(ptr+i))
        {
            large=*(ptr+i);
        }
    }
    printf("\nlargest element=%d",large);
}

```

Memory leak

[PU: 2013 fall]

A memory leak occurs when a block of memory that was previously allocated by a programmer is not properly de-allocated by the programmer. Even though that memory is no longer in use by the program, it is still “reserved”, and that block of memory cannot be used by the program until it is properly de-allocated by the programmer. That’s why it’s called a memory leak. To avoid memory leaks, memory allocated on heap should always be freed when no longer needed.

```
/* Function without memory leak */  
#include <stdlib.h>  
int main()  
{  
    int *ptr = (int *) malloc(sizeof(int));  
    /* Do some work */  
    free(ptr);  
    return 0;  
}
```

Explain the memory allocation in C programming. [PU: 2015 fall]

In c programming, Memory can be allocated in two ways.

They are:

1) Static Memory allocation

In static memory allocation the memory is allocated before the execution of program begins (During compilation). In this type of allocation the memory cannot be resized after initial allocation. So it has some limitations. Like

- Wastage of memory
- Overflow of memory

eg. int arr[100];

Here, the size of an array has been fixed to 100. If we just enter to 10 elements only, then there will be wastage of 90 memory location and if we need to store more than 100 elements there will be memory overflow.

2) Dynamic memory allocation

To overcome the limitation of static memory allocation, dynamic memory allocation technique is used. In this type of memory allocation the memory is allocated during execution of program, so that we can allocate and release memory as per requirement during runtime. Functions calloc(), malloc(), realloc() and free() are used in DMA which are defined in stdlib.h header file .

Let's use of calloc() function to allocate memory for 20 integer values

```
int *ptr;  
ptr=(int*)calloc(20,sizeof(int));
```

Differentiate between static memory allocation and dynamic memory allocation.[PU:2019 spring]

Static memory allocation	Dynamic memory allocation
1. Memory is allocated before the execution of program begins.(During Compilation)	1. Memory is allocated during the execution of program.
2. Variable remain permanently allocated.	2. Allocated only when program unit is active.
3. In this type of allocation memory cannot be resized after initial allocation.	3. In this type of allocation memory can be dynamically expanded and shrunk as necessary.
4. Implemented using stacks.	4. Implemented using heap.
5. Faster than dynamic memory allocation.	5. Slower than static memory allocation.
6. It is less efficient than dynamic memory allocation strategy.	6. It is more efficient than static memory allocation strategy.
7. Memory cannot be reuse when it is no longer needed.	7. Memory can be freed when it is no longer needed and reuse and reallocate during execution.

Advantages of dynamic memory allocation

- Dynamic Memory allocation is done at runtime.
- No need to know amount of memory prior to allocation
- We can create additional storage whenever we need them.
- We can de-allocate (free/delete) dynamic space whenever we are done with them.
- No wastage of memory
- No shortage of memory.

Disadvantages of dynamic memory allocation

- Slower execution.
- Memory need to be freed.

Assignment

- Why dynamic memory allocation is needed?
- How dynamic memory allocation is better than static memory allocation?

Write a short notes on: **Dynamic memory management [PU: 2017 fall]**

Dynamic memory management refers to managing system memory at runtime. This memory management technique allows us to allocating and releasing memory during runtime. There are four library functions that are defined in header file <stdio.h> for dynamic memory allocation.

Function	Syntax	Use of function
malloc()	ptr=(cast-type*)malloc(byte-size)	Allocates requested size of bytes and returns a pointer first byte of allocated space
calloc()	ptr=(cast-type*)calloc(n,element-size);	Allocates space for an array elements, initializes to zero and then returns a pointer to memory
realloc()	ptr=realloc(ptr,newsize);	Change the size of previously allocated space
free()	free(ptr);	De-allocate the previously allocated space

Let's use of calloc() function to allocate memory dynamically

```
int *ptr;  
ptr=(int*) calloc(25,sizeof(int));
```

This statement allocates contiguous space in memory for 25 elements each with size of int.

malloc() vs calloc()

Malloc	Calloc
1. malloc() allocates memory block of given size (in bytes)	1. calloc() allocates a region of memory large enough to hold "n elements" of "size" bytes each.
2. malloc() doesn't initialize the allocated memory.	2. The allocated region is initialized to zero.
3. It takes one argument and allocates the memory in bytes given in argument.	3. The calloc() function takes two arguments number of variables to be allocated and size of each variable.
4. Syntax: ptr=(cast-type*)malloc(byte-size);	4. Syntax: ptr=(cast-type*)calloc(n,element-size);
5. eg. <pre>int *ptr; ptr=(int*)malloc(100*sizeof(int)); A memory space equivalent to "100 times the size of integer" is reserved</pre>	5. eg. <pre>int *ptr; ptr=(int*) calloc(25,sizeof(int)); This statement allocates contiguous space in memory for 25 elements each with size of int.</pre>

Below are the different definitions of the function **search()** [PU:2013 spring]

i) void search(int *m[],int x)
 {
 }
ii) void search(int *m,int x)
 {
 }

Are they equivalent? Explain.

Solution:

In function definition,

```
void search(int *m ,int x)
{
}
```

Here, pointer is used as an function argument, which has been used to accept the address of integer type element. Here, we can pass address of one element at a time. In case of array beginning address of array is passed.

But, in function definition

```
void search(int *m[],int x)
{
}
```

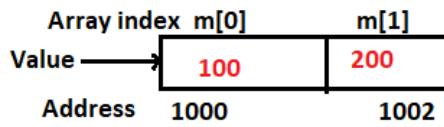
When array is passed as an argument to function, the address of entire array can be passed. Which means array of pointers to an int.

Write the output:[PU:2013 spring]

```
void main()
{
int m[2];
int *p=m;
m[0]=100;
m[1]=200;
printf("%d%d",++*p,*p);
}
```

Solution:

Tracing the program logi ,by assuming 1000 is an starting address of an array



p
1000
2000

Assuming ,1000 is starting address of array
 $*p=m;$ //implies address 1000 is stored in
pointer variable p

Now,Dereferencing ponter p,
 $*p$ refers to the value stored in memory location
that is pointed by p which is 100

so $++*p = 100+1=101$

Output :

```
101
100
```

3) Find the output:

```
void fun(int *p);
void main()
{
    int x=4;
    printf("%d\n",x);
    fun(&x);
    printf("%d\n",x);
```

}

```
void fun(int *p)
{
```

$*p = *p / 2 + 13;$

}

(Trace the program yourself)

Output:

```
4
15
```

- 4) Trace the output of the following program.

```
#include<stdio.h>
int main()
{
    int x=25;
    int *y;
    int **z;
    y=&x;
    z=&y;
    printf("x=%d\n",++x);
    printf("y=%d\n",*y);
    printf("z=%d\n",**z++);
    return 0;
}
```

(Trace the program yourself)

Output:

```
x=26
y=26
z=26
```

- 5) Trace the output of the following program.

```
void main()
{
    int arr[]={10,20,30,45,67,58,74}
    int *i,*j;
    i=&arr[1];
    j=&arr[5];
    printf("%d\n%d",j-i,*j-*i);
}
```

(Trace the program yourself)

Output

```
4
38
```

- 6) Trace the output of the following program.

```
#include<stdio.h>
#include<conio.h>
int main()
{
float a[5]={13.24,1.5,1.5,5.4,3.5};
float *j,*k;
j=&a[0];
j=j+4;
k=&a[2];
printf("\n%.2f\n%.2f\n%.2f\n%d", *j, *k, *j-*k, j-k);
getch();
}
```

(Trace the program yourself)

Output:

```
3.50
1.50
2.00
2
```

- 7) Trace the output of the following program.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int x,y,z,k;
int *ptr;
x=10;
ptr=&x;
printf("%d\n",x);
printf("%d\n",*ptr);
z=*ptr+1;
k=++(*ptr);
printf("%d\n",k);
printf("%d\n",*ptr);
printf("%d",z);
}
```

(Trace the program yourself)

Output

```
10  
10  
11  
11  
11
```

- 8) Trace the output of the following program.

```
void main()  
{  
    float f[]={10.5,1.5,2.5,3.5,4.5};  
    float *p,*q;  
    p=f;  
    p=p+4;  
    q=&f[2];  
    printf("%f\t%f\t%f",*p,*q,*p-*q);  
    getch();  
}
```

(Trace the program yourself)

Output:

```
4.50000    2.500000    2.000000
```

CHAPTER-8

Structure and Union

Structure

- Structure is a collection of different or similar data types variable under a common variable name.
- Structure is a convenient tool for handling a group of logically related data items.

Why structure is needed?

Suppose we need to store the data of students like name, address, class, rollno, age etc.

One way of doing this would be creating a different variable name, age, address to store these information separately.

However when we need to store the data of multiple students ,in that case we would need to create these several variables again for each student .This is such a big headache to store data in this way.

We can solve this problem easily by using structure. We can create a structure that has members name, address, class, rollno, age etc. and then we can create the variables of this structure or each student.

Example:

```
struct student
{
    char name[20];
    char address[20];
    int class;
    int rollno;
    int age;
}
```

Now multiple variable of struct student type can be declared as:

```
int main()
{
    struct student st1, st2,st3.... stn;
    .....
    return 0;
}
```

Defining structure

Structure must be defined first for their format that may be used later to declare structure variables. The general format or syntax to create structure is:

```
struct structure_name  
{  
    data_type member1;  
    data_type member2;  
    .....  
    .....  
    data_type memben;  
};
```

For example

```
struct student
```

```
{  
    char name[20];  
    int roll;  
    float marks;  
};
```

Here a derived type struct student is defined. Student is a structure name and name, roll, marks, are called structure elements or members. Each of these members belongs to different data types.

Declaration of structure

Method I	
<p>Syntax:</p> <pre>struct structure_name { datatype member1; datatype member2; datatype memben; }; int main() { struct structure_name variable1, variable2,.....variable n; return 0; }</pre>	<p>Example</p> <pre>struct student { char name[20]; int roll; float marks; }; int main() { struct student st1, st2, st3; }</pre>

Method II	
<p>Syntax:</p> <pre>struct structure_name { datatype member1; datatype member2; datatype memebn; } variable1, variable2, variable n;</pre>	<p>Example</p> <pre>struct student { char name[20]; int roll; float marks; } st1,st2, st3;</pre>

Accessing Member of Structure

How members of the structure are accessed? Show it with example. [PU: 2017 Spring]

To access any member of a structure, we use the dot operator (.) .

Syntax: structure_variable.member

Here structure variable refers to the name of structure type variable and member refers to name of member within structure.

Program:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
struct student
{
    char name[20];
    int roll;
    float marks;
};
int main()
{
    struct student st;
    strcpy(st.name,"Ram");
    st.roll=15;
    st.marks=85.5;
    printf("Name of student is %s",st.name);
    printf("\nRoll number= %d",st.roll);
    printf("\nMarks obtained=%f",st.marks);
    getch();
    return 0;
}
```

Output

Name of student is Ram
Roll number=15
Marks Obtained=85.5

Structure initialization

Initialization of structure at compile time

Structure variable can be also initialized at compile time. The values to be initialized must appear in order as in the definition of structure within braces and separated by commas.

Syntax:

```
struct structure_name structure_variable={value1,value2, . . . . . , value n };
```

where value1 is initialized to the first member, value2 is initialized to second member and so on.

For example:

If the structure is declared as

```
struct student
{
    char name[20];
    int rollno;
    float marks;
}
```

The variable of this type can be initialized during its declaration as shown below.

```
struct student st={"Ram",100,56.5};
```

This line is equivalent to

```
struct student st;
strcpy(st.name,"Ram");
st.rollno=15;
st.marks=56.5;
```

Program:

```
#include<stdio.h>
#include<conio.h>
struct student
{
    char name[20];
    int roll;
    float marks;
};
int main()
{
    struct student st={"Ram",15,85.5};
    printf("Name of student is %s",st.name);
    printf("\nRoll number= %d",st.roll);
    printf("\nMarks obtained=%f",st.marks);
    getch();
    return 0;
}
```

Initialization of structure at runtime

```
#include<stdio.h>
#include<conio.h>
struct student
{
    char name[20];
    int roll;
    float marks;
};
int main()
{
    struct student st;
    printf("Enter the name of student");
    gets(st.name);
    printf("Enter the Roll number of student");
    scanf("%d",&st.roll);
    printf("Enter the marks of student");
    scanf("%f",&st.marks);
    printf("Name of student is");
    puts(st.name);
    printf("Roll number= %d",st.roll);
    printf("Marks obtained=%f",st.marks);
    getch();
    return 0;
}
```

Nested structure

What is nested structure? [PU: 2013 fall, 2014 spring, 2015 spring]

The structure within structure is called Nested structure. In other words the individual members of structure can be other structure as well.

Let us consider structure date which has members day, month and year. This structure can be nested another structure say employee.i.e. structure date is member of another structure employee.

Eg.

```
struct date
{
    int day;
    int month;
    int year;
};
```

This structure can be nested within another structure as its member.

```
structure employee
{
    char name[50];
    int id;
    struct date dob;
    float salary;
}e;
```

Accessing member of nested structure

1. Write a program enlightening how to access the nested structure components.
[PU: 2012 fall]
2. How the members of the nested structure are accessed? Show it with example.
[PU: 2016 fall]

Program to illustrate accessing the nested structure components

```
#include<stdio.h>
#include<conio.h>
struct date
{
    int day;
    int month;
    int year;
};

struct employee
{
    char name[20];
    int id;
    struct date dob;
    float salary;
}e;

int main()
{
    printf("Enter the name of employee");
    scanf("%s",e.name);
    printf("Enter ID of employee");
    scanf("%d",&e.id);
    printf("Enter the year of birthday\n");
    scanf("%d",&e.dob.year);
    printf("Enter the month of birthday\n");
    scanf("%d",&e.dob.month);
    printf("Enter the day of birthday\n");
    scanf("%d",&e.dob.day);
    printf("Enter the salary of employee\n");
    scanf("%f",&e.salary);
    printf("Detail information of employee is\n");
    printf("Name\tId\tDate\tMonth\tYear\tsalary\n");
    printf("%s\t%d\t%d\t%d\t%d\t%.2f",e.name,e.id,e.dob.day,e.dob.month,e.dob.year,e.salary);
    getch();
    return 0;
}
```

In nested structure the member within inner structure is accessed as:
structure_variable.member.submember;

In above example,

The member within date are accessed as
e.dob.day;
e.dob.month;
e.dob.year;

Array of structure

Array of structure is the collection of multiple structures variables where each variables contain information about different entities.

Let us consider the structure employee

```
struct employee
{
    char name[20];
    int id;
    float salary;
};
```

Normally, when we want to store the record of multiple employee ,Let's suppose 10 ,we have to declare 10 different structure variables such as e1,e2,e3..... e9, e10.

By using array of structure we can simply declared in following ways;

<u>First way</u>	<u>Second way</u>
<pre>struct employee { char name[20]; int id; float salary; }; int main() { struct employee e[10]; }</pre>	<pre>struct employee { char name[20]; int id; float salary; } e[10];</pre>

Example:

Array of structures that stores information of 10 employees and display it.

```
#include<stdio.h>
#include<conio.h>
struct employee
{
    char name[20];
    int id;
    float salary;
};
int main()
{
    int i;
    struct employee e[10];
    printf("Enter the Information of 10 employee\n");
    for(i=0;i<10;i++)
    {
        printf("Enter the Employee id\n");
        scanf("%d",&e[i].id);
        printf("Enter the name of employee\n");
        gets(e[i].name);
        printf("Enter the salary of employee\n");
        scanf("%f",&e[i].salary);
    }
    printf("\nEmployee Information List:");
    for(i=0;i<10;i++)
    {
        printf(" \nId:%d\tName:%s\tSalary:%f",e[i].id,e[i].name,e[i].salary);
    }
    getch();
    return 0;
}
```

How to declare and initialize array of structure variables? [PU: 2017 fall]

Suppose we want declare a structure to store 5 employee details. Array of structure to store 5 employee details is can be declared as follows:

```
struct employee
{
    char name[20];
    int id;
    float salary;
};

int main()
{
    struct employee e[5];

}
```

Structure array initialization follows same syntax as we initialize single structure variable. The only difference is here we can initialize more than one structure variable at a time.

```
struct employee e[5] =
{
    { "Ramesh", 12, 15000.35 },
    { "Hari", 15, 25000.5 },
    { "Gopal", 17, 30000.25 },
    { "Sita", 2, 45000.56 },
    { "Roshan", 9, 32000.25 }
};
```

Program to illustrate accessing array of structure. (*Not necessary for this question*)

```
#include<stdio.h>
#include<conio.h>
struct employee
{
    char name[20];
    int id;
    float salary;
};
int main()
{
    int i;
    struct employee e[5] =
    {
        { "Ramesh",12,15000.35},
        { "Hari",15, 25000.5 },
        { "Gopal", 17, 30000.25 },
        { "Sita", 2,45000.56 },
        { "Roshan",9, 32000.25}
    };

    printf("\nEmployee Information List:");
    for(i=0;i<5;i++)
    {
        printf(" \nId:%d\tName:%s\tsalary:%f",e[i].id,e[i].name,e[i].salary);
    }
    getch();
    return 0;
}
```

Output

```
Employee Information List:
Id:12  Name:Ramesh      salary:15000.349609
Id:15  Name:Hari       salary:25000.500000
Id:17  Name:Gopal      salary:30000.250000
Id:2   Name:Sita       salary:45000.558594
Id:9   Name:Roshan     salary:32000.250000
```

Array within Structure

We can use array of any type as the member of structure according to our programming requirement. Array within structure can be of single structure or of an array of structures .The following program shows a structure definition having an float type array to read marks of 5 subjects of student and display the average marks.

```
#include<stdio.h>
#include<conio.h>
struct student
{
    char name[20];
    int roll;
    float marks[5];
};
int main()
{
    struct student st;
    int i;
    float avg,sum=0;
    printf("Enter the name of student\n");
    gets(st.name);
    printf("Enter the roll no of student\n");
    scanf("%d",&st.roll);
    for(i=0;i<5;i++)
    {
        printf("Enter marks of subject %d",i+1);
        scanf("%f",&st.marks[i]);
        sum=sum+st.marks[i];
    }
    avg=sum/5;
    printf("Informatin of student are\n");
    printf("Name:%s\tRoll:%d\tAverage marks:%f",st.name,st.roll,avg);
    getch();
    return 0;
}
```

Pointer to structure

Pointers can also be used pointing to a structure.

We can define pointers to structures in the following way

```
struct structure_name *pointer_name;
```

We can store the address of a structure variable in following way:

```
pointer_name = &structure_variable name;
```

To access the members of a structure using a pointer to that structure, you must use the → operator as follows –

```
pointer_name->member;
```

Program

```
#include<stdio.h>
#include<conio.h>
struct book
{
    char name[30];
    int pages;
    float price;
};
int main()
{
    struct book b={"C programming",300,550.30};
    struct book *ptr;
    ptr=&b;
    printf("\nBook name\tNo.of pages\tprice\n");
    printf("\n%s\t%d\t%f",ptr->name,ptr->pages,ptr->price);
    getch();
    return 0;
}
```

Passing structure to function

In C , structure can be passed to functions by two methods.

- 1) Passing by value(Passing actual value as argument)
- 2) Passing by reference(passing address of an argument)

Passing by value

This methods involves passing a copy of the entire structure to the called function. Any changes to structure members within the function are not reflected in original structure in the calling function.

```
#include<stdio.h>
#include<conio.h>
struct employee
{
    char name[50];
    float salary;
};
void display(struct employee e);
void addbonus(struct employee ee);
int main()
{
    struct employee emp;
    printf("Enter the name of employee");
    scanf("%s",emp.name);
    printf("Enter the salary of employee");
    scanf("%f",&emp.salary);
    addbonus(emp);
    display(emp);
    getch();
}

void addbonus(struct employee ee)
{
    ee.salary=ee.salary+5000;
}
void display(struct employee e)
{
    printf("\nInformation of Employee");
    printf("\nName:%s",e.name);
    printf("\nSalary:%f",e.salary);
}
```

Output

```
Enter the name of employee: Ramesh
Enter the salary of employee: 25000
Information of employee
Name: Ramesh
Salary: 25000
```

Passing by reference

In this method the address location of the structure is passed to the called function. The function can access indirectly the entire structure and work on it. Any changes made in structure variable in function definition reflects in original structure variable in calling function.

```
#include<stdio.h>
#include<conio.h>
struct employee
{
    char name[50];
    float salary;
};
void display(struct employee *e);
void addbonus(struct employee *ee);
int main()
{
    struct employee emp;
    printf("Enter the name of employee");
    scanf("%s",emp.name);
    printf("Enter the salary of employee");
    scanf("%f",&emp.salary);
    addbonus(&emp);
    display(&emp);
    getch();
}

void addbonus(struct employee *ee)
{
    ee->salary=ee->salary+5000;
}
void display(struct employee *e)
{
    printf("\nInformation of Employee");
    printf("\nName:%s",e->name);
    printf("\nSalary:%f",e->salary);
}
```

```
Enter the name of employee: Ramesh
Enter the salary of employee: 25000
Information of employee
Name: Ramesh
Salary: 30000
```

Self-referential structure

Write a short notes on: Self-referential structure [PU: 2015 Spring, 2017 fall]

The self-referential structure is a structure that contain pointer to a structure of the same type.

General form:	Example
struct structure_name { datatype member1; datatype member2; struct structure_name *pointer_name; };	struct node { int data; struct node * next; };

Here the structure node is referencing to self as it has a pointer of its own type 'struct node', named 'next'.

Such self-referential structures are very useful in applications that involve linked data structures, such as lists and trees.

```
#include<stdio.h>
#include<conio.h>
struct node
{
    int data;
    struct node *next;
};
int main()
{
    struct node a,b,c;
    a.data=10;
    b.data=20;
    c.data=30;
    a.next=&b;
    b.next=&c;
    c.next=NULL;
    printf("a=%d",a.data);
    printf("\nb=%d",a.next->data);
    printf("\nc=%d",b.next->data);
    getch();
    return 0;
}
```

Output:

```
a=10
b=20
c=30
```

Union

Union is a derived data type which allows to store number of variables of different data types in same memory location.

Unions are similar to structure but there is only difference in terms of storage. In structures, each member has its own memory location but all member of the union use the same storage location. Due to sharing of a common memory location, all the members of union cannot accessed at a time.

The memory occupied by the union will be the memory sized occupied by a member which requires largest memory space among members.

Eg.

```
union student
{
    char name[20];
    int roll;
    float marks;
};
```

Here, memory required for member name is 20 bytes, roll is 2 bytes and marks is 4 bytes. Here, largest memory space (ie.20 bytes) will be allocated for union.

Declaration of union

Method I	
<p>Syntax:</p> <pre>union union_name { datatype member1; datatype member2; datatype memebn; }; int main() { union union_name variable1, variable2,.....variable n; }</pre>	<p>Example:</p> <pre>union student { char name[20]; int roll; float marks; }; int main() { union student st1, st2, st3; }</pre>

Method II	
<p>Syntax:</p> <pre>union union_name { datatype member1; datatype member2; datatype memebn; } variable1, variable2,.....variable n;</pre>	<p>Example:</p> <pre>union student { char name[20]; int roll; float marks; } st1,st2, st3;</pre>

Example:

```
#include<stdio.h>
#include<conio.h>
union student
{
    char name[20];
    int roll;
    float marks;
};
int main()
{
    union student st;
    printf("Enter the Name");
    scanf("%s",st.name);
    printf("Name is:%s",st.name);
    printf("Enter the Rollno");
    scanf("%d",&st.roll);
    printf("Rollno is %d",st.roll);
    printf("Enter the marks");
    scanf("%f",&st.marks);
    printf("Marks is %f",st.marks);
    getch();
    return 0;
}
```

How is structure different from union? Give examples codes. [PU: 2017 Fall]

Both structure and union are used to store the number of variables of different data types under a common variable name.

The main difference among them is in terms of storage. In structure each member has its own storage location but all members of union use the common memory location. Due to sharing of common memory location all members cannot be accessed at a time.

Program to illustrate Structure	Program to illustrate Union
<pre>#include<stdio.h> struct student { int roll; float marks; }; int main() { struct student st; st.roll=15; st.marks=67.5; printf("Roll= %d",st.roll); printf("\nMarks=%f",st.marks); return 0; }</pre>	<pre>#include<stdio.h> union student { int roll; float marks; }; int main() { union student st; st.roll=15; st.marks=67.5; printf("Roll=%d",st.roll); printf("\nMarks=%f",st.marks); return 0; }</pre>
<p><u>Output:</u></p> <p>Roll=15 Marks=67.5</p>	<p><u>Output:</u></p> <p>Roll=(Garbage value) Marks=67.5</p>
<p>Description: Here the memory reserved for structure will be (2 bytes+4 bytes=6 bytes) 2 bytes is reserved for integer roll and 4 bytes is reserved for float marks As well all members can be used at the same time.</p>	<p>Description: Here the memory reserved for union will be 4 bytes because data type of marks is float which is the largest member of union. But, all members cannot be used at the same time because all members in union share the common memory location.</p>

Write a short notes on:

Differences between structure and union [PU: 2016 spring]

Structure	Union
Keyword struct is used.	Keyword union is used.
Syntax <pre>struct structure_name { datatype member1; datatype member2; datatype membern; };</pre>	Syntax: <pre>union union_name { datatype member1; datatype member2; datatype membern; };</pre>
The separate memory location is allocated to each member of the structure .	All members of the union share the same memory location.
All the members of structure can be accessed at a time.	Only one members of union can be accessed at a time.
Size of the structure is equal to the sum of size of the each member.	Size of the union is equal to the size of the largest member.
Eg. <pre>struct employee { char name[20]; int age; float salary; }; Memory reserved=(20+2+4)=24 bytes</pre>	Eg. <pre>struct union { char name[20]; int age; float salary; }; Memory reserved=20 bytes</pre>

Some Important Questions:

1. Differentiate structure and union. [PU: 2016 fall]
2. Define structure and union. Explain way of declaring and accessing member of them with suitable example. [PU: 2015 fall]
3. Compare and contrast structure with union. [PU: 2012 Fall]

Create a structure called employee having name, address, salary and age. Input n records of employee and display information of employee whose address is Kathmandu.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
struct employee
{
    char name[20];
    char address[20];
    float salary;
    int age;
};
int main()
{
    struct employee e[100];
    int i,n;
    printf("Enter how many records you want to enter");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        printf("Enter the records of employee %d\n",i+1);
        printf("Enter the name");
        gets(e[i].name);
        printf("Enter the address");
        gets(e[i].address);
        printf("Enter the salary");
        scanf("%f",&e[i].salary);
        printf("Enter the age");
        scanf("%d",&e[i].age);
    }
    printf("\nInformation of employee whose address is kathmandu are\n");
    printf("\nName\tAddress\tSalary\tAge");
    for(i=0;i<n;i++)
    {
        if(strcmp(e[i].address,"kathmandu")==0)
        {
            printf("\n%s\t%s\t%f\t%d",e[i].name,e[i].address, e[i].salary,e[i].age);
        }
    }
    getch();
    return 0;
}
```

Perform the following operations for above question:

Display information of employee whose name is raju:

Hint:

```
printf("\nInformation of student whose name is raju are\n");
printf("\nName\tAddress\tSalary\tAge");
for(i=0;i<n;i++)
{
    if(strcmp(e[i].name,"raju")==0)
    {
        printf("\n%s\t%s\t%f\t%d",e[i].name,e[i].address, e[i].salary,e[i].age);
    }
}
```

Display information of employee whose salary is greater than 10, 000

Hint:

```
printf("\nInformation of employee whose salary is greater than 10,000 are\n");
printf("\nName\tAddress\tSalary\tAge");
for(i=0;i<n;i++)
{
    if(e[i].salary>10000)
    {
        printf("\n%s\t%s\t%f\t%d",e[i].name,e[i].address, e[i].salary,e[i].age);
    }
}
```

Display information of employee whose age is greater than 30

Hint:

```
printf("\nInformation of employee whose age is greater than 30 are\n");
printf("\nName\tAddress\tSalary\tAge");
for(i=0;i<n;i++)
{
    if(e[i].age>30)
    {
        printf("\n%s\t%s\t%f\t%d",e[i].name,e[i].address, e[i].salary,e[i].age);
    }
}
```

Display the information of Employee who has highest salary

Hint:

```
float high;
.....
.....
high=e[0].salary;
for(i=0;i<n;i++)
{
    if(e[i].salary>high)
    {
        high=e[i].salary;
    }
}
printf("\nInformation of employee who have highest salary are\n");
printf("\nName\tAddress\tSalary\tAge");
for(i=0;i<n;i++)
{
    if(e[i].salary==high)
    {
        printf("\n%s\t%s\t%f\t%d",e[i].name,e[i].address,e[i].salary,e[i].age);
    }
}
```

Display all the information of employee

Hint:

```
printf("\nInformation of employee are\n");
printf("\nName\tAddress\tSalary\tAge");
for(i=0;i<n;i++)
{
    printf("\n%s\t%s\t%f\t%d",e[i].name,e[i].address,e[i].salary,e[i].age);
}
```

Assignment:

- 1) Create a structure called customer having name, address, account number and balance.
Input n records and
 - Display information of customer whose balance is greater than 10000
 - Ask for account number from customer and display their information*[Note: Data type for accountno will be long int and format specifier %ld]*
- 2) Create a structure called employee having name, address, post salary and age. Input n records of employee and display information of those employee whose post is accountant.
- 3) Create a structure called university having name, college, and number of faculties. Input n records of college and display the name of those colleges whose address is Kathmandu.
- 4) Create a structure to specify students data rollno,name,address,marks1,mark2,mark3 to read 20 students data into array of structures and print the information of those students who are failed.
(Note: Pass mark for any subject is 45)

1) Create a structure for the following data.

Roll.no	Name	Address	Faculty	Data of birth		
				mm	dd	yy

Write a program to input 100 students and Display the records of student of “computer” faculty.[PU:2013 fall]

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
struct dob
{
    int dd;
    int mm;
    int yy;
};
struct student
{
    int roll;
    char name[20];
    char address[20];
    char faculty[20];
    struct dob d;
};
```

```

int main()
{
    int i;
    struct student st[100];
    for(i=0;i<100;i++)
    {
        printf("Enter the records of student %d\n",i+1);
        printf("Enter the roll");
        scanf("%d",&st[i].roll);
        printf("Enter the name");
        gets(st[i].name);
        printf("Enter the address");
        gets(st[i].address);
        printf("Enter the faculty");
        gets(st[i].faculty);
        printf("Enter year of birthday");
        scanf("%d",&st[i].d.yy);
        printf("Enter month of birthday");
        scanf("%d",&st[i].d.mm);
        printf("Enter day of birthday");
        scanf("%d",&st[i].d.dd);
    }
    printf("\nThe records of student whose faculty is computer are\n");
    printf("\nRollno\tName\tAddress\tFaculty\tDate of Birth");
    for(i=0;i<100;i++)
    {
        if(strcmp(st[i].faculty,"computer")==0)
        {
            printf("\n%d\t%s\t%s\t%s\t%d\t%d\t%d",st[i].roll,st[i].name,st[i].address,
            st[i].faculty,st[i].d.mm,st[i].d.dd,st[i].d.yy);
        }
    }
    getch();
    return 0;
}

```

Perform the following operations for above questions.

Display the records of student whose address is “pokhara”

Hint:

```
printf("\nThe records of student whose address is pokhara are\n");
for(i=0;i<100;i++)
{
    if(strcmp(st[i].address,"pokhara")==0)
    {
        printf("\n%d\t%s\t%s\t%s\t%d\t%d\t%d",st[i].roll,st[i].name,st[i].address,
        st[i].faculty,st[i].d.mm,st[i].d.dd,st[i].d.yy);
    }
}
```

Display the records of student who are not from “kathmandu”

Hint:

```
printf("\nThe records of student who are not from kathmandu are\n");
for(i=0;i<100;i++)
{
    if(strcmp(st[i].address,"kathmandu")!=0)
    {
        printf("\n%d\t%s\t%s\t%s\t%d\t%d\t%d",st[i].roll,st[i].name,st[i].address,
        st[i].faculty,st[i].d.mm,st[i].d.dd,st[i].d.yy);
    }
}
```

Display the records of student whose name is “raju”

Hint:

```
printf("\nThe records of student whose name is raju are\n");
for(i=0;i<100;i++)
{
    if(strcmp(st[i].name,"raju")==0)
    {
        printf("\n%d\t%s\t%s\t%s\t%d\t%d\t%d",st[i].roll,st[i].name,st[i].address,
        st[i].faculty,st[i].d.mm,st[i].d.dd,st[i].d.yy);
    }
}
```

Display records of student who born in july month

Hint:

```
printf("\nThe records of student who born in july are\n");
printf("\nRollno\tName\tAddress\tFaculty\tDate of Birth");
for(i=0;i<100;i++)
{
    if(st[i].d.mm==7)
    {
        printf("\n%d\t%s\t%s\t%s\t%d\t%d\t%d",st[i].roll,st[i].name,st[i].address,
        st[i].faculty,st[i].d.mm,st[i].d.dd,st[i].d.yy);
    }
}
```

Display records of student who was born in 1995

Hint:

```
printf("\nThe records of student who was born in 2016 are\n");
printf("\nRollno\tName\tAddress\tFaculty\tDate of Birth");
for(i=0;i<100;i++)
{
    if(st[i].d.yy==1995)
    {
        printf("\n%d\t%s\t%s\t%s\t%d\t%d\t%d",st[i].roll,st[i].name,st[i].address,
        st[i].faculty,st[i].d.mm,st[i].d.dd,st[i].d.yy);
    }
}
```

Display all the records

Hint:

```
printf("\nThe records of all student are\n");
printf("\nRollno\tName\tAddress\tFaculty\tDate of Birth");
for(i=0;i<100;i++)
{
    printf("\n%d\t%s\t%s\t%s\t%d\t%d\t%d",st[i].roll,st[i].name,st[i].address,
    st[i].faculty,st[i].d.mm,st[i].d.dd,st[i].d.yy);
}
```

Create a structure called book, member name, price, author, and published date in day, month, and year. Write a program to read 100 books information from the user and displays those records having price greater than 250. [2014 spring]

```
#include<stdio.h>
#include<conio.h>
struct date
{
    int dd;
    int mm;
    int yy;
};
struct book
{
    char name[20];
    float price;
    char author[20];
    struct date d;
};

int main()
{
    int i;
    struct book b[100];
    for(i=0;i<100;i++)
    {
        printf("Enter the information of book %d\n",i+1);
        printf("Enter the book name");
        gets(b[i].name);
        printf("Enter the author name");
        gets(b[i].author);
        printf("Enter the book price");
        scanf("%f",&b[i].price);
        printf("Enter book published year");
        scanf("%d",&b[i].d.yy);
        printf("Enter book published month");
        scanf("%d",&b[i].d.mm);
        printf("Enter book published date");
        scanf("%d",&b[i].d.dd);
    }
}
```

```

printf("\nInformation of Books having price greater than 250 are\n");
printf("\nName\tPrice\tAuthor\tPublished date");
for(i=0;i<100;i++)
{
    if(b[i].price>250)
    {
        printf("\n%s\t%f\t%s\t%d\t%d\t%d",b[i].name,b[i].price,b[i].author,b[i].d.dd,b[i].d.mm,
        b[i].d.yy);
    }
}
getch();
return 0;
}

```

Assignment.

WAP to input the following records of 50 employees using structure and display them properly.

Name	Address	Post	Salary	Data of Appointment		
				mm	dd	Yr

[PU: 2015 spring]

Given a structure of employee

Name	Address	telephone	Salary	Year of joining		
				mm	dd	Yr

Write a program to input data of 100 employee and display the record of those employees living in pokhara.

*[Note: datatype for telephone must be **long int** and format specifier is **%ld**]*

[PU: 2013 spring]

Create a structure for the following data.

Emp_id	Emp_name	Address	Department	Date of Birth		
				mm	dd	Yr

Also write a program to input 100 employee records and display whose Department is “sales”.

[PU:2019 spring]

Define a structure called ‘football’ that will describe the following information:

Player name

Country name

Number of goal scored.

Using football, declare an array player with 50 elements and write a program to read the information about all the information about all the 50 players and print a country wise list containing names of players with their number of goal scored. [PU: 2014 fall]

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#define N 50
struct football
{
    char name[20];
    char country[20];
    int goal;
};
int main()
{
    struct football p[50],temp;
    int i, j;
    for (i = 0; i < N; i++)
    {
        printf("Enter the infomation of player %d\n",i+1);
        printf("Enter player name");
        gets(p[i].name);
        printf("Enter player country");
        gets(p[i].country);
        printf("Enter Number of goal scored : ");
        scanf("%d", &p[i].goal);
    }
}
```

```

for (i = 0; i < N-1; i++)
{
    for (j = 0; j < N - 1-i; j++)
    {
        if (strcmp(p[j].country, p[j + 1].country) > 0)
        {
            temp = p[j];
            p[j] = p[j + 1];
            p[j + 1] = temp;
        }
    }
}
printf("Country wise list of players are:\n");
printf("Player name\tCountry\tNo of goal scored\n");
for (i = 0;i < N;i++)
{
    printf("\n%s\t%s\t%d",p[i].name,p[i].country,p[i].goal);
}
getch();
}

```

Assignment:

Define a structure called cricket that will describe the following information:

player name,
team name,
batting average.

Using cricket, declare an array player with 50 elements and write a program to read the information about all the 50 players and print a team-wise list containing names of player with their batting average.

In a bank there are n customers with attributes name, account no and balance. WAP to find out information of customer who has highest balance in bank. [PU: 2006 spring]

```

#include<stdio.h>
#include<conio.h>
struct customer
{
    char name[20];
    long int accno;
    float balance;
};

```

```

int main()
{
    int i,n;
    float high;
    struct customer c[100];
    printf("Enter the number of customer");
    scanf("%d",&n);
    printf("Enter records of %d customer\n",n);
    for(i=0;i<n;i++)
    {
        printf("Enter Name");
        gets(c[i].name);
        printf("Enter account no");
        scanf("%ld",&c[i].accno);
        printf("Enter balance");
        scanf("%f",&c[i].balance);
    }
    high=c[0].balance;
    for(i=0;i<n;i++)
    {
        if(c[i].balance>high)
        {
            high=c[i].balance;
        }
    }
    printf("\nInformation of customer who have highest balance are\n");
    printf("\nName\tAccount no\tBalance");
    for(i=0;i<n;i++)
    {
        if(c[i].balance==high)
        {
            printf("\n%s\t%ld\t%f",c[i].name,c[i].accno,c[i].balance);
        }
    }
    getch();
    return 0;
}

```

Assignment:

Write a C program to accept details of 'n' employee (eno, ename, salary) and display the details of employee having highest salary. Use array of structure.

Create a structure called student with data members name, rollno, and marks of three subjects for 100 students. Display the name of students with average marks greater than 80.

[PU: 2007 spring]

```
#include<stdio.h>
#include<conio.h>
struct student
{
    char name[20];
    int rollno;
    float sub1,sub2,sub3;
    float avg;
};
int main()
{
    int i;
    struct student st[100];
    printf("Enter the information of 100 students\n");
    for(i=0;i<100;i++)
    {
        printf("Enter Name");
        gets(st[i].name);
        printf("Enter Rollno");
        scanf("%d",&st[i].rollno);
        printf("Enter marks of sub1");
        scanf("%f",&st[i].sub1);
        printf("Enter marks of sub2");
        scanf("%f",&st[i].sub2);
        printf("Enter marks of sub3");
        scanf("%f",&st[i].sub3);
        st[i].avg=(st[i].sub1+st[i].sub2+st[i].sub3)/3;
    }
    printf("Name of student with average marks greater than 80\n");
    for(i=0;i<100;i++)
    {
        if(st[i].avg>80)
        {
            puts(st[i].name);
        }
    }
    getch();
    return 0;
}
```

CHAPTER-9

Files and File Handling

The input/output function, Like printf(),scanf(),getchar(),putchar(),gets(),puts(),are known as console oriented I/O functions which always use keyboard for input device. While using these library function the entire data is lost when either the program is terminated or computer is turned off. Again it becomes cumbersome and time consuming to handle large volume of data through keyboard. It takes lot of time to enter the entire data. If user makes a mistake while entering the data he/she has to start from beginning again. If the same data is to be entered again at some later age, again we have to enter the same data. These problems invite the concept of data file in which data can be stored on disks and read whenever necessary, without destroying data.

What is file?

File is a place on the disk where a group of related data is stored. The data file allow us to store information permanently and to access and alter the information whenever necessary.

Why file handling is needed in c program? [PU: 2013 spring]

When problem involves large volume of data and in such situations the console oriented i/o operations pose two major problems.

- It becomes cumbersome and time consuming to handle large volume of data through terminals.
- The entire data is lost when either a program is terminated or the computer is turned off.

To solve these problems the concept of data file is introduced in which data can be stored on disks and read whenever necessary, without destroying data. However, if we have a file containing all the data, we can easily access the contents of the file using few commands in C.

Types of Data files:

Programming language in C has various library functions creating and processing data files.

Mainly there are two types of data files.

- 1) High level(standard or stream oriented) files
- 2) Low level(system oriented) files

Stream-oriented data files are easier to work with and therefore more commonly used. It can be subdivided into two categories.

- I. **Text files:** Text file is a human readable sequence of characters and the words they form that can be encoded into computer-readable format such as ASCII. A text file is also known as ASCII file and can be read by any word processor .Text files stores information in consecutive characters. These characters can be interpreted as individual data items or as a component of strings or numbers.
- II. **Binary files:** A binary files is made up of machine readable symbols that is made up of 0's and 1's.The binary file must be interpreted by the program that understand in advance exactly how it is formatted. Binary files are organized into blocks containing contiguous bytes of information. These blocks represents more complex data structures, such as array and structures.

System-oriented data files:

They are more closely related to the computer's operating system than stream-oriented data files. A separate set of procedures with accompanying library functions is required to process system oriented data files.

Differences between text file and binary file

Text file	Binary file
1. Text file is human readable because everything is stored in terms of text.	1. In binary file everything is written in terms of 0 and 1, therefore binary file is not human readable.
2. Every text file is terminated with EOF(End of File),whose ASCII value is 26.	2. There is no such special character present in the binary mode files to mark the end of file.
3. A newline (\n) character is converted into the carriage return-linefeed combination before being written to the disk.	3. In binary file, these conversions will not take place.
4. In text file, the text and characters are stored one character per byte. For example, the integer value 12345 will occupy 2 bytes in memory but it will occupy 5 bytes in text file.	4. In binary file, the integer value 12345 will occupy 2 bytes in memory as well as in file.
5. The extension for text file is .txt.	5. The extension for binary file is .dat.
6. File opening modes for text file are r, w, a, r+, w+ and a+.	6. File opening modes for binary file are rb,wb, ab, rb+,wb+ and ab+.

Operations involved in files

What are basic file operations that can be performed in C?

The basic file operations in C are:

- Naming a file
- Opening a file
- Reading data from file
- Writing data to file and
- Closing the file

Opening and closing a file

- Before a program can write to a file or read from a file, the program must open it.
- Opening a file establishes a link between program and the operating system. This provides the operating system the name of a file and the mode in which file is to be opened.
- While working with high level data file, we need buffer area where information is stored temporarily in the course of transferring data between computer memory and data file.
- The process of establishing connection between the program and the file is called opening a file.
- A structure named FILE is defined in the file stdio.h that contains all the information about file like name, status, buffer size, current position and of file status etc. A file pointer is a pointer to a structure of type FILE.
- Whenever a file is opened, a structure of type FILE associated with it and the file pointer that points to this structure identifies this file.

The function **fopen()** is used to open the file. The buffer area is established by:

FILE *ptr_variable;

And file is opened by using the following syntax:

ptr_variable = fopen("file_name", "file_opening_mode");

Here, file_name is the name of the file we intend to open and opening mode specifies the purpose of opening file.

For example:

```
FILE *fptr1,*fptr2;  
fptr1=fopen("myfile.txt","w");  
fptr2=fopen("testfile.dat","rb");
```

The file that was opened using **fopen()** function must be closed when no more operations are to be performed on it. After closing the file the connection between file and the program is broken. On closing the file, all buffers associated with it are flushed and can be available for other files.

Its syntax is:

```
fclose(ptr_variable);
```

for example:

```
fclose(fp1);  
fclose(fp2);
```

File Pointer

What is the significance of file pointer in file handling? [PU: 2018 spring, 2014 spring]

A file pointer is a pointer to a structure, which contains information about the file, including its name, current position of the file, whether the file is being read or written, and whether errors or end of the file have occurred. The user does not need to know the details, because the definitions obtained from stdio.h include a structure declaration called FILE. The only declaration needed for a file pointer is symbolized by

```
FILE *fp;
```

This says that fp is the file pointer that points to a FILE structure.

File opening modes

Describe the different file opening modes in c. [PU: 2013 spring, 2014 spring]

Write a short notes on:

File opening modes: [PU: 2012 fall, 2015 fall]

File opening modes specifies the way in which a file should be opened In other words it specifies the purpose of opening a file. Let us discuss the file opening mode for text file.

File mode	Meaning of mode	During Inexistence of file
“r”	Opens an existing file for reading purpose only	If the file does not exist, fopen() returns NULL.
“w”	Opens a file for writing purpose	If the file exists, its contents are overwritten(contents are deleted first and written). If the file does not exist, it will be created.
“a”	Opens an existing file for appending purpose (i.e. addition of new content at the end of the existing file)	If the file does not exists, it will be created.
“r+”	Opens an existing file for both reading and writing purpose.	If the file does not exist, fopen() returns NULL.
“w+”	Opens file for reading and writing purpose.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
“a+”	Opens an existing file for both reading and appending purpose.	If the file does not exists, it will be created.

Note: The file opening modes in binary files are similar to text mode .Character b is added to each mode to indicate the binary mode. eg. rb, wb,ab,rb+,wb+,ab+.

Eg.

```
FILE *fptr,;
fptr=fopen("sample.txt", r);
```

Here, file named sample.txt is opening for reading mode only.

Similarly,

```
FILE *fptr,;
fptr=fopen("hello.dat",wb+);
```

Here, file named hello.dat is opening for both reading and writing.

1) Character I/O functions

Using character I/O functions data can be read from a file or written to a file one character at a time.

fgetc(): It is used to read a character from a file.

Syntax: **char_variable=fgetc(file_ptr_variable);**

fputc(): It is used to write a character to file.

Syntax: **fputc(char_variable,file_ptr_variable);**

WAP to read a characters from keyboard and write character to a file.

OR

WAP to create a file named sample.txt and write some characters to a file until user hits the enter key.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int main()
{
    char ch;
    FILE *fptr;
    fptr=fopen("sample.txt","w");
    if(fptr==NULL)
    {
        printf("File cannot be openend");
        exit(1);
    }
    printf("Enter some characters");
    while((ch=getchar())!='\n')
    {
        fputc(ch,fptr);
    }
    fclose(fptr);
    getch();
    return 0;
}
```

WAP to open the file created in above example and read character stored in it and display it to the screen.

OR

WAP to read a character from the file name sample.txt and display it on screen.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int main()
{
char ch;
FILE *fptr;
fptr=fopen("sample.txt","r");
printf("The character from file are\n");
if(fptr==NULL)
{
printf("File cannot be openend");
exit(1);
}
while((ch=fgetc(fptr))!=EOF)
{
putchar(ch);
}
fclose(fptr);
getch();
return 0;
}
```

WAP to read a characters from a keyboard, store in a file and display it.

OR

WAP to read a characters from a keyboard and store in a file named 'sample.txt' and display it.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int main()
{
char ch;
FILE *fptr;
fptr=fopen("sample.txt","w+");
if(fptr==NULL)
{
printf("File cannot be opened");
exit(1);
}
```

```
printf("Enter some character");
while((ch=getchar())!='\n')
{
fputc(ch,fptr);
}
rewind(fptr);
printf("The characters from file are");
while((ch=fgetc(fptr))!=EOF)
{
putchar(ch);
}
fclose(fptr);
getch();
return 0;
}
```

2) String Input/ Output functions

Using string I/O functions, data can be read from a file or written to a file in the form of array of characters.

fgets(): It is used to read a string from a file

syntax: fgets(string_variable,int_value,file_ptr_variable);

Here, int_value denotes the number of characters in string .The functions read a string from a file representing file_ptr_variable and stores in a variable string variable.

fputs():It is used to write a string to a file

syntax: fputs(string_variable,file_ptr_variable);

Here, contents in the string variable is written to a file representing a file_ptr_variable.

Write a program to create a file named “info.txt” and write “Welcome to Nepal” and display it.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int main()
{
FILE *fptr;
fptr=fopen("info.txt","w");
if(fptr==NULL)
{
printf("File cannot be opened");
exit(1);
}
fputs("Welcome to Nepal",fptr);
fclose(fptr);
getch();
return 0;
}
```

WAP to write a string data to a file

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<stdlib.h>
int main()
{
FILE *fptr;
char str[50];
fptr=fopen("sample.txt","w");
if(fptr==NULL)
{
printf("File cannot be opened");
exit(1);
}
printf("Enter the strings");
while(strlen(gets(str))!=0)
{
fputs(str,fptr);
}
fclose(fptr);
getch();
return 0;
}
```

Program to read string data from file

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int main()
{
FILE *fptr;
char str[50];
fptr=fopen("sample.txt","r");
if(fptr==NULL)
{
printf("File cannot be opened");
exit(1);
}
printf("The information from string is\n");
while(fgets(str,50,fptr)!=NULL)
{
puts(str);
}
fclose(fptr);
getch();
return 0;
}
```

WAP to input strings to a file and display it.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<stdlib.h>
int main()
{
char str[50];
FILE *fptr;
fptr=fopen("text.txt","w+");
if(fptr==NULL)
{
printf("File cannot be opened");
exit(1);
}
printf("Enter the string!\n");
while((strlen(gets(str))!=0))
{
fputs(str,fptr);
}
```

```

rewind(fp);
printf("The information in file are\n");
while(fgets(str,50,fp)!=NULL)
{
    puts(str);
}
fclose(fp);
getch();
return 0;
}

```

3) Formatted Input /Output

These functions are used to read numbers, characters or string from a file or write them to a file in a format as per requirement.

fprintf() It is formatted output function which is used to write integer ,float ,char or string data to a file.

Syntax: `fprintf(file_ptr_variable,"control_string",list_variables);`

fscanf() It is formatted input function which is used to read integer ,float, char or string data from a file.

Syntax: `fscanf(file_ptr_variable,"format _specifier",&list_variables);`

WAP to input age, gender, marks of a student in a file called “student.txt”. Now read these information from the file display them.

```

#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int main()
{
    char gender;
    int age;
    float marks;
    FILE *fp;
    fp=fopen("student.txt","w+");
    if(fp==NULL)
    {
        printf("File cannot be opened");
        exit(1);
    }

```

```

printf("Enter the gender\n");
scanf("%c",&gender);
printf("Enter the age\n");
scanf("%d",&age);
printf("Enter the marks\n");
scanf("%f",&marks);
fprintf(fp,"%c\t%d\t%f",gender,age,marks);
rewind(fp);
fscanf(fp,"%s%d%f",&gender,&age,&marks);
printf("Gender=%c\tAge=%d\tMarks=%f",gender,age,marks);
fclose(fp);
getch();
return 0;
}

```

Write a program to create a file “hello.txt”, write data info to the file and finally read from the file.

[PU: 2017 fall]

Solution:

(just change the file name of above program to “hello.txt”)

WAP to open a new file and read name, address and telephone number of 10 employees from a user and write to a file.[PU: 2013 fall]

```

#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int main()
{
char name[20],address[20];
long int telno;
int i;
FILE *fp;
fp=fopen("employee.txt","w");
if(fp==NULL)
{
printf("File cannot be opened");
exit(1);
}

```

```

for(i=0;i<10;i++)
{
printf("Enter the name,address and telno of employee %d\n",i+1);
scanf("%s%s%ld",name,address,&telno);
fprintf(fp,"%s\t%s\t%ld\n",name,address,telno);
}
fclose(fp);
getch();
return 0;
}

```

Program to display the information stored in above file “employee .txt
(Note: This solution is not necessary for above question)

```

#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int main()
{
char name[20],address[20];
long int telno;
int i;
FILE *fp;
fp=fopen("employee.txt","r");
if(fp==NULL)
{
printf("Unable to open file");
exit(1);
}
printf("Name\tAddress\tTel.no\n");
for(i=0;i<3;i++)
{
fscanf(fp,"%s%s%ld",name,address,&telno);
printf("\n%s\t%s\t%ld",name,address,telno);
}
fclose(fp);
getch();
return 0;
}

```

Write a program to open a file named inventory.txt and store the following data to the file.

Product Name	Quantity	Rate
AAA	3	50
BBB	2	100
CCC	4	40

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int main()
{
    char items[20];
    int qty,rate,i;
    FILE *fptr;
    fptr=fopen("inventory.txt","w");
    if(fptr==NULL)
    {
        printf("Unable to open file");
        exit(1);
    }
    for(i=0;i<3;i++)
    {
        printf("Enter Product Name, Quantity and Rate");
        scanf("%s%d%d",items,&qty,&rate);
        fprintf(fptr,"%s\t%d\t%d\n",items,qty,rate);
    }
    fclose(fptr);
    getch();
    return 0;
}
```

Write a program to open a file named inventory.txt created in above program and read data from file and display inventory table with total amount [i.e rate*qty] of each item.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int main()
{
    char items[20];
    int qty,rate,i;
    FILE *fptr;
    fptr=fopen("inventory.txt","r");
    if(fptr==NULL)
    {
        printf("Unable to open file");
        exit(1);
    }

    printf("Product Name\tQuantity\tRate\tTotal Amount\n");
    for(i=0;i<3;i++)
    {
        fscanf(fptr,"%s%d%d",items,&qty,&rate);
        printf("\n%s\t%d\t%d\t%d",items,qty,rate,qty*rate);
    }
    fclose(fptr);
    getch();
    return 0;
}
```

Output

Product Name	Quantity	Rate	Total Amount
AAA	3	50	150
BBB	2	100	200
CCC	4	40	160

WAP to read name, age and height of person from the keyboard until user want to enter the information and stored them in a file named 'info.txt' and read information from display them on screen.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int main()
{
char name[20];
char choice;
int age;
float height;
FILE *fptr;
fptr=fopen("hello.txt","w+");
if(fptr==NULL)
{
printf("Unable to open file");
exit(1);
}
do
{
printf("Enter the name");
scanf("%s",name);
printf("Enter the age");
scanf("%d",&age);
printf("Enter the height");
scanf("%f",&height);
printf("Do you want to continue?");
scanf("%c",&choice);
fprintf(fptr,"%s\t%d\t%f\n",name,age,height);
}while(choice=='Y' || choice=='y');
rewind(fptr);
printf("\nName\tAge\tHeight");
while(fscanf(fptr,"%s%d%f",name,&age,&height)!=EOF)
{
printf("\n%s\t%d\t%f",name,age,height);
}
fclose(fptr);
getch();
return 0;
}
```

Record I/O

- Record i/o writes numbers to files in binary format.so that integers are stored in 2 bytes ,floating point numbers are stored in 4 bytes and so on.
- It permits reading or writing multiple data values in the form of arrays, structures and array of structures once.
- Thus the arrays, structures, array of structures etc. can be read from a file or written to a file as a single unit using record i/o.

There are functions fwrite and fread for writing and reading structure (record) in a file on a disk.

fwrite () used for writing entire block to a given file.

Syntax:

```
fwrite(&ptr,size_of_array_or_structure,number_of_structure_or_array,fptr);
```

fread() is used to read an entire block from a given file.

Syntax:

```
fread(&ptr, size_of_array_or_structure, number_of_structure_or_array, fptr);
```

where,

- i) ptr is the address of an array or structure to be written
- ii) Size_of_array_or_structure is an integer value that shows the size of structure or size of array which is being read or written.
- iii) number_of_structure_or_array is an integer value that indicates number of arrays or structures to be written to file or read from file.
- iv) fptr is a file pointer of a file opened in binary mode.

<i>Illustration of fwrite() function</i>	<i>Illustration of fread() function</i>
<pre>#include<stdio.h> #include<conio.h> #include<stdlib.h> struct student { int roll; char name[25]; float marks; }; int main() { FILE *fptr; char ch; struct student st; fptr = fopen("test.dat","wb"); if(fptr == NULL) { printf("file cannot be opened"); exit(1); } do { printf("Enter the Rollno\n"); scanf("%d",&st.roll); printf("Enter the Name\n"); scanf("%s",st.name); printf("Enter the Marks\n"); scanf("%f",&st.marks); fwrite(&st,sizeof(st),1,fptr); printf("Do you want to add another data (y/n):"); ch = getche(); }while(ch=='y' ch=='Y'); printf("\nData written successfully"); fclose(fptr); getch(); return 0; }</pre>	<pre>#include<stdio.h> #include<conio.h> #include<stdlib.h> struct student { int roll; char name[25]; float marks; }; int main() { FILE *fptr; char ch; struct student st; fptr = fopen("test.dat","rb"); if(fptr == NULL) { printf("File cannot be opened"); exit(1); } printf("\nRoll.no\tName\tMarks\n"); while(fread(&st,sizeof(st),1,fptr)==1) { printf("\n%d\t%s\t%f",st.roll,st.name,st.marks); } fclose(fptr); getch(); return 0; }</pre>

Illustration of fread/fwrite

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
struct student
{
    int roll;
    char name[25];
    float marks;
};
int main()
{
    FILE *fptr;
    char ch;
    struct student st;
    fptr = fopen("test.dat","wb+");
    if(fptr == NULL)
    {
        printf("File cannot be opened");
        exit(1);
    }
    do
    {
        printf("Enter the Rollno:\n");
        scanf("%d",&st.roll);
        printf("Enter the Name:\n");
        scanf("%s",st.name);
        printf("Enter the Marks:\n");
        scanf("%f",&st.marks);
        fwrite(&st,sizeof(st),1,fptr);
        printf("Do you want to add another data (y/n):\n");
        ch = getche();
    }while(ch=='y' || ch=='Y');
    printf("Data written successfully\n");
    rewind(fptr);
    printf("\nRoll\tName\tMarks\n");
    while(fread(&st,sizeof(st),1,fptr)==1)
    {
        printf("\n%d\t%s\t%f",st.roll,st.name,st.marks);
    }
    fclose(fptr);
    getch();
    return 0; }
```

- 1)** Create a file called “university .dat”. Input n records of colleges in a structure having collegename, location, and no of faculties of Pokhara University. Now display name of colleges whose location is Kathmandu.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<string.h>
struct university
{
    char name[20];
    char location[20];
    int nooffaculties;
};
int main()
{
    struct university u[100];
    int i,n;
    FILE *fptr;
    fptr=fopen("university.dat","wb+");
    if(fptr==NULL)
    {
        printf("File cannot be opened");
        exit(1);
    }
    printf("Enter number of colleges\n");
    scanf("%d",&n);
    printf("Enter records of %d college\n",n);
    for(i=0;i<n;i++)
    {
        printf("Enter college name\n");
        gets(u[i].name);
        printf("Enter location\n");
        gets(u[i].location);
        printf("Enter number of faculties\n");
        scanf("%d",&u[i].nooffaculties);
    }
    fwrite(&u,sizeof(u),n,fptr);
    rewind(fptr);
```

```

printf("The name of colleges whose location is kathmandu are:\n");
fread(&u,sizeof(u),n,fptr);
for(i=0;i<n;i++)
{
    if(strcmp(u[i].location,"kathmandu")==0)
    {
        printf("\n%s",u[i].name);
    }
}
fclose(fptr);
getch();
return 0;
}

```

Alternative solution-1

```

#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<string.h>
struct university
{
    char name[20];
    char location[20];
    int nooffaculties;
};
int main()
{
    struct university u[100];
    int i,n;
    FILE *fptr;
    fptr=fopen("university.dat","wb+");
    if(fptr==NULL)
    {
        printf("File cannot be opened");
        exit(1);
    }
    printf("Enter number of colleges\n");
    scanf("%d",&n);
    printf("Enter records of %d college\n",n);
    for(i=0;i<n;i++)
    {
        printf("Enter college name\n");
        gets(u[i].name);
        printf("Enter location\n");

```

```

gets(u[i].location);
printf("Enter number of faculties\n");
scanf("%d",&u[i].nooffaculties);
fwrite(&u[i],sizeof(u[i]),1,fptr);
}
rewind(fptr);
printf("The name of colleges whose location is kathmandu are:\n");

for(i=0;i<n;i++)
{
    fread(&u,sizeof(u[i]),1,fptr);
    if(strcmp(u[i].location,"kathmandu")==0)
    {
        printf("\n%s",u[i].name);
    }
}
fclose(fptr);
getch();
return 0;
}

```

Alternative solution-2

```

#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<string.h>
struct university
{
    char name[20];
    char location[20];
    int nooffaculties;
};
int main()
{
    struct university u;
    int i,n;
    FILE *fptr;
    fptr=fopen("university.dat","wb+");
    if(fptr==NULL)
    {
        printf("File cannot be opened");
        getch();
        exit(1);
    }

```

```

printf("Enter number of colleges\n");
scanf("%d",&n);
printf("Enter records of %d college\n",n);
for(i=0;i<n;i++)
{
    printf("Enter college name\n");
    gets(u.name);
    printf("Enter location\n");
    gets(u.location);
    printf("Enter number of faculties\n");
    scanf("%d",&u.nooffaculties);
    fwrite(&u,sizeof(u),1,fptr);
}
rewind(fptr);
printf("The name of colleges whose location is kathmandu are:\n");
while(fread(&u,sizeof(u),1,fptr)==1)
{
    if(strcmp(u.location,"kathmandu")==0)
    {
        printf("\n%s",u.name);
    }
}
fclose(fptr);
getch();
return 0;
}

```

Assignment:

Create a structure with data members college name, location and number of faculties. Now read the information of 50 colleges affiliated to Pokhara university and write them into the file named university.dat and while retrieving ,display the information of colleges whose address is Kathmandu.

- 2) Write a program by using structure that includes using structure that includes AccNO, CName, CAddress, CTelno and balance for 100 customers of ABC financial institution and store in account.txt file and finally display the information of the customers who have balance greater than Rs.10,000 [PU:2012 fall]**

```

#include<stdio.h>
#include<conio.h>
#include<stdlib.h>

```

```

struct customer
{
    int long accno;
    int long ctelno;
    char cname[20];
    char caddress[20];
    float balance;
};
int main()
{
    struct customer c[100];
    int i;
    FILE *fptr;
    fptr=fopen("account.txt","wb+");
    if(fptr==NULL)
    {
        printf("File cannot be opened");
        exit(1);
    }
    printf("Enter records of 100 customers\n");
    for(i=0;i<100;i++)
    {
        printf("Enter account no\n");
        scanf("%ld",&c[i].accno);
        printf("Enter the customer name\n");
        gets(c[i].cname);
        printf("Enter customer telephone no\n");
        scanf("%ld",&c[i].ctelno);
        printf("Enter customer address\n");
        gets(c[i].caddress);
        printf("Enter balance\n");
        scanf("%f",&c[i].balance);
    }
    fwrite(&c,sizeof(c),100,fptr);
    rewind(fptr);
    printf("Information customers who have balance greater than 10000 are:\n");
    printf("AccNo\tCName\tCtelNo\tCAddress\tBalance\n");
    fread(&c,sizeof(c),100,fptr);
}

```

```

for(i=0;i<100;i++)
{
if(c[i].balance>10000)
{
printf("\n%ld\t%s\t%ld\t%s\t%f",c[i].accno,c[i].cname,c[i].ctelno,c[i].caddress,c[i].balance);
}
}
fclose(fptr);
getch();
return 0;
}

```

- 3) Write a program to create structure for the following data for student (RN, Name, phone, address and semester). Read the 10 students by user and write only those students whose semester is 1 in file “student.txt”.**[PU:2016 fall]**

```

#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
struct student
{
char name[20];
int rollno;
long int phone;
char address[20];
int semester;
};
int main()
{
int i;
struct student st[10];
FILE *fptr;
fptr=fopen("student.txt","wb");
if(fptr==NULL)
{
    printf("File cannot be opened");
    exit(1);
}

```

```

printf("Enter records of 10 students\n");
for(i=0;i<10;i++)
{
    printf("Enter the name\n");
    gets(st[i].name);
    printf("Enter rollno\n");
    scanf("%d",&st[i].rollno);
    printf("Enter Phone number\n");
    scanf("%ld",&st[i].phone);
    printf("Enter the Address\n");
    gets(st[i].address);
    printf("Enter semester\n");
    scanf("%d",&st[i].semester);
    if(st[i].semester==1)
    {
        fwrite(&st[i],sizeof(st[i]),1,fptr);
    }
}
printf("Data written successfully");
fclose(fptr);
getch();
return 0;
}

```

- 4) Write a program to input name, address, faculty, program and GPA(in maximum 4.0) of 500 students and store them in 'RESULT.DAT' data file and display the records of those student whose faculty is 'Engineering' and GPA>3.5. [PU: 2016 spring]

```

#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<stdlib.h>
struct student
{
    char name[20];
    char address[20];
    char faculty[20];
    char program[20];
    float gpa;
};

```

```

int main()
{
struct student st[500];
int i;
FILE *fptr;
fptr=fopen("result.dat","wb+");
if(fptr==NULL)
{
printf("File cannot be opened");
exit(1);
}
printf("Enter records of 500 students");
for(i=0;i<500;i++)
{
printf("Enter the records of student %d\n",i+1);
printf("Enter student name\n");
gets(st[i].name);
printf("Enter the address\n");
gets(st[i].address);
printf("Enter the faculty\n");
gets(st[i].faculty);
printf("Enter the Program\n");
gets(st[i].program);
do
{
printf("Enter the gpa\n");
scanf("%f",&st[i].gpa);
}while(st[i].gpa>4);
}
fwrite(&st,sizeof(st),500,fptr);
rewind(fptr);
printf("Information student of faculty Engineering with GPA greater than 3.5 are:\n");
printf("Name\tAddress\tFaculty\tProgram\tGPA\n");
fread(&st,sizeof(st),500,fptr);
for(i=0;i<500;i++)
{
if(st[i].gpa>3.5&&strcmp(st[i].faculty,"Engineering")==0)
{
printf("\n%s\t%s\t%s\t%s\t%f",st[i].name,st[i].address,st[i].faculty,st[i].program,st[i].gpa);
}
}
getch();
return 0;
}

```

- 5) Write a program to create structure for the following data for cricket game.(Country name, Player name, playing type(eg. batting ,balling or both),Number of matches played by player and salary).Save the information in a fie named “cricket.txt” and display the information of those players who had played more than 10 matches. [PU: 2018 fall]

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
struct cricketplayer
{
    char pname[20];
    char pcountry[20];
    char ptype[20];
    int noofmatches;
    float salary;
};
int main()
{
    struct cricketplayer c[100];
    int i,n;
    FILE *fptr;
    fptr=fopen("cricket.txt","wb+");
    if(fptr==NULL)
    {
        printf("File cannot be opened");
        exit(1);
    }
    printf("Enter the number of cricket players\n");
    scanf("%d",&n);
    printf("Enter records of %d cricket players",n);
    for(i=0;i<n;i++)
    {
        printf("Enter player name");
        gets(c[i].pname);
        printf("Enter country");
        gets(c[i].pcountry);
        printf("Enter playing type");
        gets(c[i].ptype);
        printf("Enter number of matches played");
        scanf("%d",&c[i].noofmatches);
        printf("Enter salary");
        scanf("%f",&c[i].salary);
    }
}
```

```

fwrite(&c,sizeof(c),n,fptr);
rewind(fptr);
printf("Information of those player who had played more than 10 matches are\n");
printf("\nName\tcountry\tPlaying type\tNo of matches\tSalary\n");
fread(&c,sizeof(c),n,fptr);
for(i=0;i<n;i++)
{
if(c[i].noofmatches>10)
{
printf("\n%s\t%s\t%s\t%d\t%f",c[i].pname,c[i].pcountry,c[i].ptype,c[i].noofmatches,c[i].salary);
}
}
fclose(fptr)
getch();
return 0;
}

```

- 6) Write a program to create structure named football for the following data for football game(country name, player name, team name, number of matches played by player and salary).Save the information in a file named “football.txt” and display the information of those players whose country name is Portugal and had played more than 10 matches.

```

#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<stdlib.h>
struct football
{
char pname[20];
char pcountry[20];
char tname[20];
int noofmatches;
float salary;
};
int main()
{
struct football f[100];
int i,n;
FILE *fptr;
fptr=fopen("football.txt","wb+");
if(fptr==NULL)
{
printf("File cannot be opened");
exit(1);
}

```

```

printf("Enter the number of football players\n");
scanf("%d",&n);
printf("Enter records of %d football players",n);
for(i=0;i<n;i++)
{
    printf("Enter player name\n");
    gets(f[i].pname);
    printf("Enter country\n");
    gets(f[i].pcountry);
    printf("Enter team name\n");
    gets(f[i].tname);
    printf("Enter number of matches played\n");
    scanf("%d",&f[i].noofmatches);
    printf("Enter salary\n");
    scanf("%f",&f[i].salary);
}
fwrite(&f,sizeof(f),n,fptr);
rewind(fptr);
printf("Player whose country name is portugal and had played more than 10 matches are\n");
printf("\nName\tcountry\tTeam name\tNo of matches\tSalary\n");
fread(&f,sizeof(f),n,fptr);
for(i=0;i<n;i++)
{
    if(strcmp(f[i].pcountry,"portugal")==0&&f[i].noofmatches>10)
    {
        printf("\n%s\t%s\t%s\t%d\t%f",f[i].pname,f[i].pcountry,f[i].tname,f[i].noofmatches,f[i].salary);
    }
}
fclose(fptr);
getch();
return 0;
}

```

7) Create a structure for the following data:

ID, name, address, salary and joining date (dd/mm/yy) and WAP to input 100 employee and information and store it in file employee.dat and now read records from the file and display records of those employee whose address is pokhara.

```

#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<string.h>

```

```

struct date
{
    int dd;
    int mm;
    int yy;
};

struct employee
{
    int id;
    char name[20];
    char address[20];
    float salary;
    struct date d;
};

int main()
{
    int i;
    struct employee e[100];
    FILE *fptr;
    fptr=fopen("employee.dat","wb+");
    if(fptr=NULL)
    {
        printf("file cannot be opened");
        exit(1);
    }
    printf("Enter the records of 100 employees\n");
    for(i=0;i<100;i++)
    {
        printf("Enter records of employee %d\n",i+1);
        printf("Enter the Employee id\n");
        scanf("%d",&e[i].id);
        printf("Enter the name\n");
        gets(e[i].name);
        printf("Enter the address\n");
        gets(e[i].address);
        printf("Enter the salary\n");
        scanf("%f",&e[i].salary);
        printf("Enter the joining year\n");
        scanf("%d",&e[i].d.yy);
        printf("Enter the joining month\n");
        scanf("%d",&e[i].d.mm);
        printf("Enter the joining day\n");
        scanf("%d",&e[i].d.dd);
    }
}

```

```

fwrite(&e,sizeof(e),100,fptr);
rewind(fptr);
printf("\nThe information of employees having in pokhara");
printf("\nID\tName\tAddress\tSalary\tDate of Joining");
fread(&e,sizeof(e),100,fptr);
for(i=0;i<100;i++)
{
if(strcmp(e[i].address,"pokhara")==0)
{
printf("\n%d\t%s\t%s\t%f\t%d\t%d\t%d",e[i].id,e[i].name,
e[i].address,e[i].salary,e[i].d.dd,e[i].d.mm,e[i].d.yy);
}
}
fclose(fptr);
getch();
return 0;
}

```

Assignment:

Create a structure named Employee with structure members name, eid, address and gender. Structure need to read information for 50 employees. Write all content into the file info.dat and while retrieving display the information of those employees whose address is “Kathmandu”.

[PU: 2019 fall]

- 8) Create a structure called goods that stores number, price, purchase date and quantity. WAP to store information of 100 goods in the file “good.dat” [PU: 2011 fall]**

```

#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
struct date
{
int yy;
int mm;
int dd;
};
struct goods
{
int number;
float price;
struct date d;
int quantity;
};

```

```

int main()
{
    int i;
    struct goods g[100];
    FILE *fptr;
    fptr=fopen("goods.dat","wb");
    if(fptr==NULL)
    {
        printf("File cannot be opened");
        exit(1);
    }
    printf("Enter records of 100 goods\n");
    for(i=0;i<100;i++)
    {
        printf("Enter number\n");
        scanf("%d",&g[i].number);
        printf("Enter the price\n");
        scanf("%f",&g[i].price);
        printf("Enter the quantity\n");
        scanf("%d",&g[i].quantity);
        printf("Enter year of purchase\n");
        scanf("%d",&g[i].d.yy);
        printf("Enter month of purchase\n");
        scanf("%d",&g[i].d.mm);
        printf("Enter day of purchase\n");
        scanf("%d",&g[i].d.dd);
    }
    fwrite(&g,sizeof(g),100,fptr);
    printf("Data written successfully");
    fclose(fptr);
    getch();
    return 0;
}

```

Program to read the data stored in above file goods.dat

```

#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
struct date
{
    int yy;
    int dd;
    int mm;
};

```

```

struct goods
{
    int number;
    float price;
    struct date d;
    int quantity;
};

int main()
{
    int i;
    struct goods g[100];
    FILE *fptr;
    fptr=fopen("goods.dat","rb");
    if(fptr==NULL)
    {
        printf("file cannot be opened");
        exit(1);
    }
    printf("\n\tNumber\tPrice\tQuantity\tPurchase Date\n");
    fread(&g,sizeof(g),100,fptr);
    for(i=0;i<100;i++)
    {
        printf("\n\t%d\t%f\t%d\t%d\t%d",g[i].number,g[i].price,g[i].quantity,g[i].d.yy,g[i].d.mm,
g[i].d.dd);
    }
    fclose(fptr);
    getch();
    return 0;
}

```

9) Consider the following structure:

Roll.No	Name	Address	Faculty	Date of Birth		
				mm	dd	yy

Write a program to create “student.txt” file to store the above records for 100 students and display the records of student who are not from pokhara. [PU: 2014 fall]

```

#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<string.h>

```

```

struct dob
{
    int mm;
    int dd;
    int yy;
};

struct student
{
    int rollno;
    char name[20];
    char address[20];
    char faculty[20];
    struct dob d;
};

int main()
{
    int i;
    struct student st[100];
    FILE *fptr;
    fptr=fopen("student.txt","wb+");
    if(fptr=NULL)
    {
        printf("File cannot be opened\n");
        exit(1);
    }
    printf("Enter the records of 100 students\n");
    for(i=0;i<100;i++)
    {
        printf("Enter records of student %d\n",i+1);
        printf("Enter the rollno\n");
        scanf("%d",&st[i].rollno);
        printf("Enter the name\n");
        gets(st[i].name);
        printf("Enter the address\n");
        gets(st[i].address);
        printf("Enter the faculty\n");
        gets(st[i].faculty);
        printf("Enter the year of birthday\n");
        scanf("%d",&st[i].d.yy);
        printf("Enter the month of birthday\n");
        scanf("%d",&st[i].d.mm);
        printf("Enter the day of birthday\n");
        scanf("%d",&st[i].d.dd);
    }
}

```

```

fwrite(&st,sizeof(st),100,fptr);
rewind(fptr);
printf("\nThe information of students who are not from pokhara");
printf("\nRollno\tName\tAddress\tFaculty\tDate of birth");
fread(&st,sizeof(st),100,fptr);
for(i=0;i<100;i++)
{
if(strcmp(st[i].address,"pokhara")!=0)
{
printf("\n%d\t%s\t%s\t%s\t%d\t%d\t%d",st[i].rollno,st[i].name,st[i].address,st[i].faculty,
st[i].d.mm,st[i].d.dd,st[i].d.yy);
}
}
fclose(fptr);
getch();
return 0;
}

```

Assignment:

i) Consider the following structure:

Roll.No	Name	Address	Faculty	Date of Birth		
				mm	dd	yy

[PU: 2018 spring]

Write a program to create “student.txt” file to store the above records for 100 students and display the records of student who are not from Kathmandu.

ii) Consider the following structure

Roll.No	Name	Address	Faculty	Date of Birth		
				mm	dd	yy

Write a program to create “student.txt” file to store the above records for 100 students. [PU:2019 spring]

10) Write a program to read the name ,author and price of 500 books in a library from the file "libarary.dat".Now print the name and price of those books whose price is above Rs.300.[PU:
2015 fall]

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
struct book
{
    char name[20];
    char author[20];
    float price;
};
int main()
{
    int i;
    struct book b[500];
    FILE *fptr;
    fptr=fopen("library.dat","rb");
    if(fptr==NULL)
    {
        printf("file cannot be opened");
        exit(1);
    }
    printf("Book which price is above 300 are");
    printf("\nName\tprice");
    fread(&b,sizeof(b),500,fptr);
    for(i=0;i<500;i++)
    {
        if(b[i].price>300)
        {
            printf("\n%s\t%f",b[i].name,b[i].price);
        }
    }
    fclose(fptr);
    getch();
    return 0;
}
```

Alternative solution

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
struct book
{
    char name[20];
    char author[20];
    float price;
};
int main()
{
    int i;
    struct book b;
    FILE *fptr;
    fptr=fopen("library.dat","rb");
    if(fptr==NULL)
    {
        printf("file cannot be opened");
        exit(1);
    }
    printf("Book which price is above 300 are");
    printf("\nName\tprice");
    while(fread(&b,sizeof(b),1,fptr)==1)
    {
        if(b.price>300)
        {
            printf("\n%s\t%f",b.name,b.price);
        }
    }
    fclose(fptr);
    getch();
    return 0;
}
```

- 11) Write a program to input name, address, registration no, faculty and academic year of admission in university of 'n' number of students of Pokhara University and append them in data file called 'STUDENT.DAT'. Then display the records of those students by reading the records from 'STUDENT.DAT' data file who got admission in 2016. [PU: 2015 spring]

```

#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
struct student
{
    char name[20];
    char address[20];
    long int regno;
    char faculty[20];
    int year;
};
int main()
{
    struct student st;
    int i,n;
    FILE *fptr;
    fptr=fopen("student.dat","ab+");

    if(fptr==NULL)
    {
        printf("File cannot be opened");
        exit(1);
    }
    printf("Enter number of students");
    scanf("%d",&n);
    printf("Enter records of %d students of pokhara university\n",n);
    for(i=0;i<n;i++)
    {
        printf("Enter name\n");
        gets(st.name);
        printf("Enter address\n");
        gets(st.address);
        printf("Enter registration number\n");
        scanf("%ld",&st.regno);
        printf("Enter faculty\n");
        gets(st.faculty);
        printf("Enter admission year\n");
        scanf("%d",&st.year);
        fwrite(&st,sizeof(st),1,fptr);
    }
    rewind(fptr);
    printf("The records of student who got admission in 2016 are:\n");
    printf("\nName\tAddress\tReg.no.\tFaculty\tAdmission Year");
}

```

```

while(fread(&st,sizeof(st),1,fptr)==1)
{
    if(st.year==2016)
    {
        printf("\n%s\t%s\t%ld\t%s\t%d",st.name,st.address,st.regno,st.faculty,st.year);
    }
}
fclose(fptr)
getch();
return 0;
}

```

Alternative solution:

```

#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
struct student
{
    char name[20];
    char address[20];
    long int regno;
    char faculty[20];
    int year;
};
int main()
{
    struct student st[100],s;
    int i,n;
    FILE *fptr;
    fptr=fopen("student.dat","ab+");

    if(fptr==NULL)
    {
        printf("File cannot be opened\n");
        exit(1);
    }
    printf("Enter number of students");
    scanf("%d",&n);
    printf("Enter records of %d students of pokhara university\n",n);

```

```

for(i=0;i<n;i++)
{
printf("Enter name\n");
gets(st[i].name);
printf("Enter address\n");
gets(st[i].address);
printf("Enter registration number\n");
scanf("%ld",&st[i].regno);
printf("Enter faculty\n");
gets(st[i].faculty);
printf("Enter admission year\n");
scanf("%d",&st[i].year);
}
fwrite(&st,sizeof(st),n,fptr);
rewind(fptr);
printf("The records of student who got admission in 2016 are:\n");
printf("\nName\tAddress\tReg.no.\tFaculty\tAdmission Year");
while(fread(&s,sizeof(s),1,fptr)==1)
{
if(s.year==2016)
{
    printf("\n%s\t%s\t%ld\t%s\t%d",s.name,s.address,s.regno,s.faculty,s.year);
}
}
fclose(fptr);
getch();
return 0;
}

```