Pokhara University

TECH CLUB

C PROGRAMMING

EXAM SOLUTIONS

ROSHAN LAMICHHANE
MAHESHWOR ACHARYA
VISHAL KANU
SAROJ ARYAL
KUSAL LAMSAL

fb.com/putechclub
ESTD. 2076

About PU Tech Club

Being the first batch of B.E Computer and B.E Software at School of Engineering, Pokhara University, we wanted to create a club to promote skill learning activities.

This exam reference is supposed to be helpful for those who are having troubles while preparing for the exam of C language.

The current committee members of Pu Tech Club are :

- President/Chairperson: Roshan Lamichhane, B.E Computer/075
- Vice-President/Vice-Chairperson: Maheshwor Acharya, B.E Computer/075
- Secretary: Kusal Lamsal, B.E Software/075
- Treasurer: Vishal Kanu, B.E Computer/075
- Committee members:
  - Saroj Aryal, B.E Software/075
  - Susmita Thapa, B.E Computer/075
  - Nikash Subedi, B.E Computer/075
  - Ashish Adhikari, B.E Computer/075

Connect with Tech club:

Facebook: https://www.facebook.com/putechclub

CONTENTS

CHAPTER 1: INTRODUCTION

Q.1) Define programming language. Differentiate between high level programming language and low-level programming language. [2013 Fall] [2015Spring]

→ A programming language is a set of commands, instructions, and other syntax use to create a software program.

The main difference between **high level language** and **low level language** is that, Programmers can easily understand or interpret or compile the high level language in comparison of machine. On the other hand, Machine can easily understand the low level language in comparison of human beings. Examples of high level languages are C, C++, Python, Java, etc.

S.N	High Level Language	Low Level Language
1.	It is programmer friendly language.	Low level language is machine readable form of program.
2.	High level language is less memory efficient.	Low level language is high memory efficient.
3.	It is easy to understand.	It is tough to understand.
4.	In high level language debugging (troubleshooting) i.e. finding and correcting errors are easier	Debugging in the low level language is quite difficult.
5.	It needs compiler or interpreter for translation.	It needs assembler for translation.
6.	It can run on any platform.	It is machine dependent.
7.	Low level language coding and compiling is time consuming process	High level language coding and compiling is much easy and takes very less time to compile

Q.2) What do you mean by programming language? Explain different types of programming language. [2013 Spring] [2014Fall]

→ A programming language is a set of commands, instructions, and other syntax use to create a software program.

Types of Programming Languages

There are two types of programming languages, which can be categorized into the following ways:

1. Low level language
   a) Machine language (1GL)
   b) Assembly language (2GL)
2. High level language
   a) Procedural-Oriented language (3GL)
   b) Problem-Oriented language (4GL)
   c) Natural language (5GL)

1. **Low level language**

This language is the most understandable language used by computer to perform its operations. It can be further categorized into:

a) **Machine Language (1GL)**

Machine language consists of strings of binary numbers (i.e. 0s and 1s) and it is the only one language, the processor directly understands. Machine language has a Merit of very fast execution speed and efficient use of primary memory.

Merits:

1. It is directly understood by the processor so has faster execution time since the programs written in this language need not to be translated.
2. It doesn't need larger memory.

Demerits:

1. It is very difficult to program using 1GL since all the instructions are to be represented by 0s and 1s.
2. Use of this language makes programming time consuming.
3. It is difficult to find error and to debug.
4. It can be used by experts only.

b) **Assembly Language**

Assembly language is also known as low-level language because to design a program programmer requires detailed knowledge of hardware specification. This language uses mnemonics code (symbolic operation code like 'ADD' for addition) in place of 0s and 1s. The program is converted into machine code by assembler. The resulting program is referred to as an object code.

Merits:

1. It is makes programming easier than 1GL since it uses mnemonics code for programming. Eg: ADD for addition, SUB for subtraction, DIV for division, etc.
2. It makes programming process faster.
3. Error can be identified much easily compared to 1GL.
4. It is easier to debug than machine language.

Demerits:

1. Programs written in this language is not directly understandable by computer so translators should be used.
2. It is hardware dependent language so programmers are forced to think in terms of computer's architecture rather than to the problem being solved.
3. Being machine dependent language, programs written in this language are very less or not portable.
4. Programmers must know its mnemonics codes to perform any task.

2. **High level language**

Instructions of this language closely resemble to human language or English like words. It uses mathematical notations to perform the task. The high level language is easier to learn. It requires less time to write and is easier to maintain the errors. The high level language is converted into machine language by one of the two different languages translator programs; interpreter or compiler.

High level language can be further categorized as:

a) **Procedural-Oriented language (3GL)** : Procedural Programming is a methodology for modeling the problem being solved, by determining the steps and the order of those steps that must be followed in order to reach a desired outcome or specific program state. These languages are designed to express the logic and the procedure of a problem to be solved. It includes languages such as Pascal, COBOL, C, FORTAN, etc.

Merits:

1. Because of their flexibility, procedural languages are able to solve a variety of problems.
2. Programmer does not need to think in term of computer architecture which makes them focused on the problem.
3. Programs written in this language are portable.

Demerits:

1. It is easier but needs higher processor and larger memory.
2. It needs to be translated therefore its execution time is more.

b) **Problem-Oriented language (4GL)** : It allows the users to specify what the output should be, without describing all the details of how the data should be manipulated to produce the result. This is one step ahead from 3GL. These are result oriented and include database query language.
E.g. Visual Basic, C#, PHP, etc.
The objectives of 4GL are to:

1. Increase the speed of developing programs.
2. Minimize user's effort to obtain information from computer.
3. Reduce errors while writing programs.

Merits: Programmer need not to think about the procedure of the program. So, programming is much easier.

Demerits:

1. It is easier but needs higher processor and larger memory.
2. It needs to be translated therefore its execution time is more.

c**) Natural language (5GL)**

Natural language is still in developing stage where we could write statements that would look like normal sentences.

Merits:

1. Easy to program.
2. Since, the program uses normal sentences, they are easy to understand.
3. The programs designed using 5GL will have artificial intelligence (AI).
4. The programs would be much more interactive and interesting.

Demerits:

1. It is slower than previous generation language as it should be completely translated into binary code which is a tedious task.
2. Highly advanced and expensive electronic devices are required to run programs developed in 5GL. Therefore, it is an expensive approach.

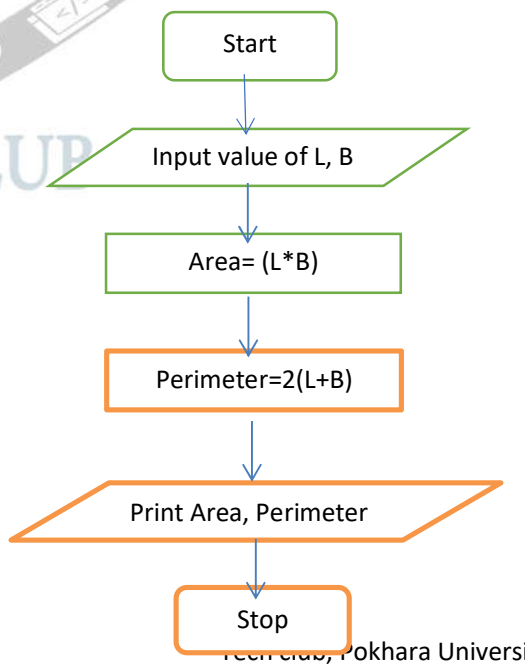
Q.3) What do you mean by algorithm and flowchart? Explain the C compilation process in brief.[2013Fall]

→ To write a logical step-by-step method to solve the problem is called algorithm, in other words, an algorithm is a procedure for solving problems. In order to solve a mathematical or computer problem, this is the first step of the procedure. An algorithm includes calculations, reasoning and data processing. Algorithms can be presented by natural languages, pseudo code and flowcharts, etc.

A flowchart is the graphical or pictorial representation of an algorithm with the help of different symbols, shapes and arrows in order to demonstrate a process or a program. With algorithms, we can easily understand a program. The main purpose of a flowchart is to analyze different processes. Several standard graphics are applied in a flowchart.

Algorithm & Flowchart to find Area and Perimeter of Rectangle

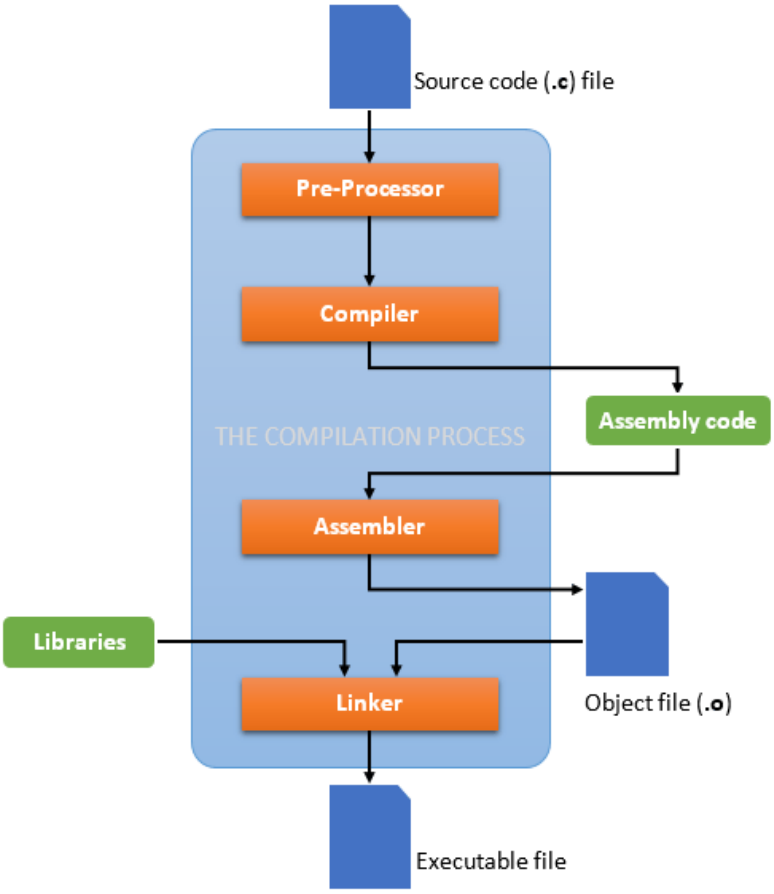
L: Length of Rectangle
B: Breadth of Rectangle
AREA: Area of Rectangle
PERIMETER: Perimeter of Rectangle
Algorithm
Step-1 Start
Step-2 Input Side Length & Breadth say L, B
Step-3 Area = L x B
Step-4 PERIMETER = 2 x (L + B)
Step-5 Display AREA, PERIMETER
Step-6 Stop



Q. Explain process of compilation in C in brief.

→ The process of translating source code written in high level to low level machine code is called as Compilation. The compilation is done by special software known as compiler. The compiler checks source code for any syntactical or structural errors and generates object code.

C is a high-level language and it needs a compiler to convert it into an executable code so that the program can be run on our machine. Compiler converts a C program into an executable. There are four phases for a C program to become an executable: Preprocessing, Compiling, Assembling, and Linking.



1. Preprocessing: The first stage of compilation is called preprocessing. In this stage, lines starting with a # character are interpreted by the preprocessor as preprocessor commands. These commands form a simple macro language with its own syntax and semantics. This language is used to reduce repetition in source code by providing functionality to inline files, define macros, and to conditionally omit code.

2. Compilation: The second stage of compilation is confusingly enough called compilation. In this stage, the preprocessed code is translated to assembly instructions specific to the target processor architecture. These form an intermediate human readable language.

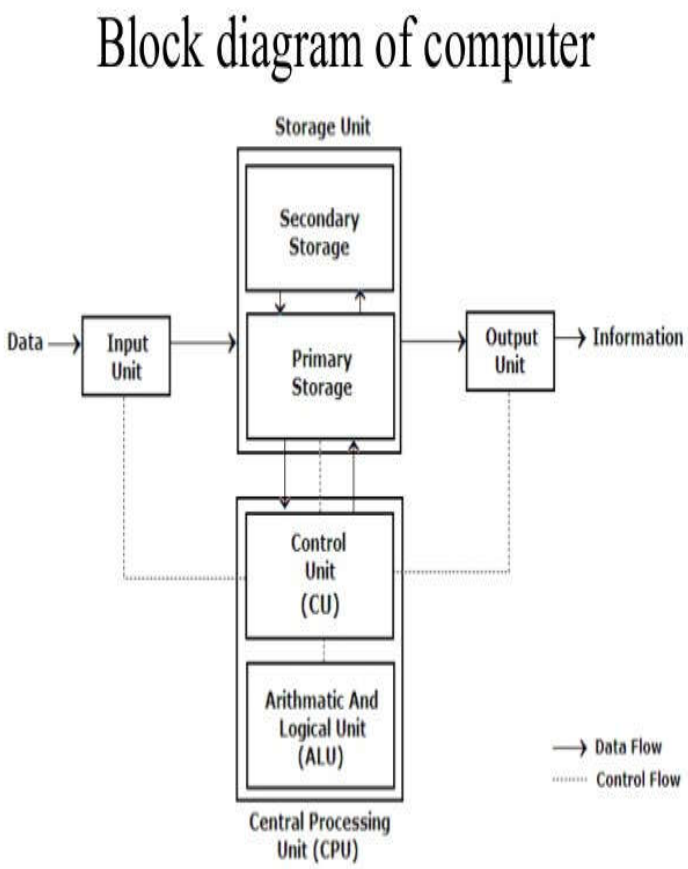
3. Assembly: During this stage, an assembler is used to translate the assembly instructions to object code. The output consists of actual instructions to be run by the target processor.

4. Linking: The object code generated in the assembly stage is composed of machine instructions that the processor understands but some pieces of the program are out of order or missing. To produce an executable program, the existing pieces have to be rearranged and the missing ones filled in. This process is called linking.
It links all the function calls with their original definition. Which means the function printf() gets linked to its original definition. Linker generates the final executable file.

Q.4) With the help of block diagram of digital computer explain the function of control unit and memory unit.[2014Spring]

→ Computer is an electronic device which performs tasks given by user with extremely fast speed and accuracy. Like any other device or machine, a computer system has also a number of parts. A computer system can be blocked into mainly three parts:

1. Input Unit
2. Central Processing Unit :
   i .Control unit (CU)
   ii. Arithmetic & Logic unit (ALU)
   iii. Memory Unit (MU)
3. Output Unit



Control Unit: A control unit (CU) is a piece of hardware that manages the activities of peripherals (separate devices attached to the computer, such as monitors, hard drives, printers, etc.) Although the CPU (central processing unit-the "big boss" in the computer) gives instructions to the controller, it is the control unit itself that performs the actual physical transfer of data.

Functions of control Unit

1. It is responsible for controlling the transfer of data and instructions among other units of a computer.
2. It manages and coordinates all the units of the computer.
3. It obtains the instructions from the memory, interprets them, and directs the operation of the computer.
4. It communicates with Input/Output devices for transfer of data or results from storage.
5. It does not process or store data.

Memory or Storage Unit : This unit can store instructions, data, and intermediate results. This unit supplies information to other units of the computer when needed. It is also known as internal storage unit or the main memory or the primary storage or Random Access Memory (RAM).

Its size affects speed, power, and capability. Primary memory and secondary memory are two types of memories in the computer. Functions of the memory unit are −

1. It stores all the data and the instructions required for processing.
2. It stores intermediate results of processing.
3. It stores the final results of processing before these results are released to an output device.
4. All inputs and outputs are transmitted through the main memory

Q.5) What is the importance of documentation in programming? Write an algorithm and draw a flowchart to find and output all the roots of the quadratic equations, for non-zero coefficients. [2014Fall]

Ans: For a programmer, reliable documentation is always a must. The presence of documentation helps keep track of all aspects of an application and it improves on the quality of a software product. Its main focuses are development, maintenance and knowledge transfer to other developers. Successful documentation will make information easily accessible, provide a limited number of user entry points, help new users learn quickly, simplify the product and help cut support costs.

The phases of software development documentation are:

1. Implementation Plan, Design Document, Test Plan and Test Cases, and End User Documentation.

2. To recognize what we coded, if the code gets a bit older.

3. Documentation is important for others to understand our code.

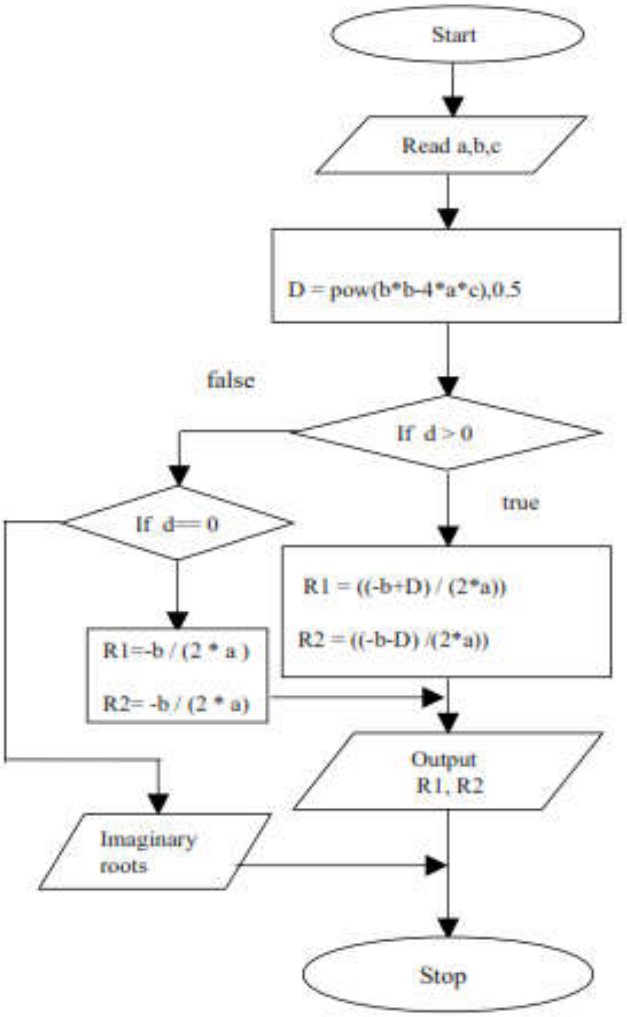
4. Documentation makes our code simple and easy to read.

5. Documentation offers immense support for maintenance.

Algorithm:

Step 1: Start
Step 2: Read A, B, C as integer
Step 3: Declare disc(discriminant), deno(denominator), x1, x2 as float
Step 4: Assign disc = (B * B) - (4 * A * C)
Step 5: Assign deno = 2 * A;
Step 6: if(disc > 0)
begin
Print “THE ROOTS ARE REAL ROOTS”
Assign x1 ← (-B / deno) + (sqrt(disc) / deno)
Assign x2 ← (-B / deno) - (sqrt(disc) / deno)
Print x1, x2
end
else if(disc = 0)
begin
Print “ THE ROOTS ARE REPEATED ROOTS"
Assign x1 ← -B / deno
Print x1
end
else Print “THE ROOTS ARE IMAGINARY ROOTS”
Step7: Stop

Flowchart:



Q.6) Discuss the significance of Algorithm and flowchart in programming. Draw a flowchart for finding greatest digit for the supplied number by the user.[2014Fall]

→ An algorithm is a set of instructions, sometimes called a procedure or a function that is used to perform a certain task. This can be a simple process, such as adding two numbers together, or a complex function, such as adding effects to an image.

Advantages of algorithm

- 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.
- Development of the procedure itself, which involves identification of the processes, major decision points, and variables necessary to solve the problem.
- Problems that would be difficult or impossible to solve wholesale can be approached as a series of small, solvable sub problems.
- Separation of the procedure steps facilitates division of labor and development of expertise.

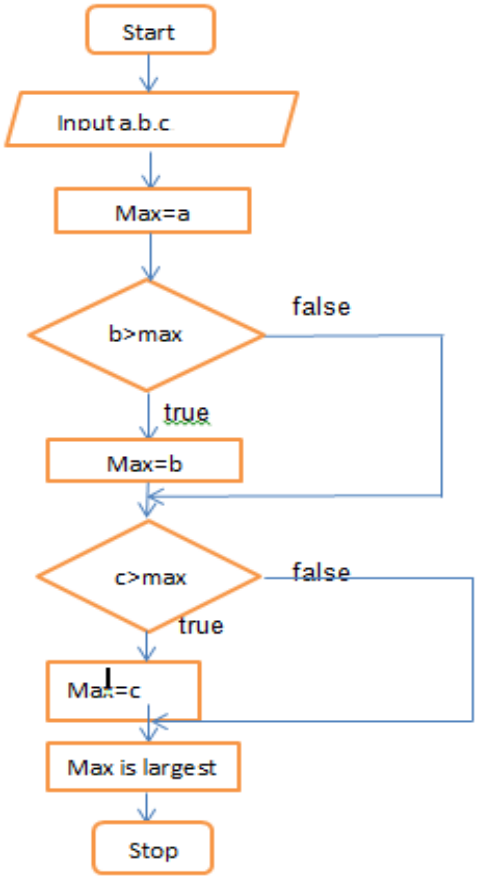
FLOWCHART:

Unlike an algorithm, Flowchart uses different symbols to design a solution to a problem. It is another commonly used programming tool. By looking at a Flowchart one can understand the operations and

sequence of operations performed in a system. Flowchart is often considered as a blueprint of a design used for solving a specific problem.

Advantages of flowchart:

1. Flowchart is an excellent way of communicating the logic of a program.
2. Easy and efficient to analyze problem using flowchart.
3. During program development cycle, the flowchart plays the role of a blueprint, which makes program development process easier.
4. After successful development of a program, it needs continuous timely maintenance during the course of its operation. The flowchart makes program or system maintenance easier.
5. It is easy to convert the flowchart into any programming language code. Algorithm & Flowchart



Q.7) What is a flowchart? Write an algorithm and draw a flowchart to display weather a number is prime or not. [2015 Fall]

→ A flowchart is the graphical or pictorial representation of an algorithm with the help of different symbols, shapes and arrows in order to demonstrate a process or a program. With algorithms, we can easily understand a program. The main purpose of a flowchart is to analyze different processes. Several standard graphics are applied in a flowchart.

Algorithm to check prime or not:

Step1: Start
Step2: Declare variables n,I,flag
Step3: Initialize variables flag=1,i-2
Step4: Read n from user
Step5: If n<1
Display Not prime(any number less than 1 is not prime)
Step6: Repeat the steps until i<[(n/2)+1]
If remainder of n/i=0, set flag =0
Goto step7
Step7: If flag ==0 ,
Display "'n' is not prime"
Else Display "'n' is prime"
Step8: Stop

Q.8) Write short notes on Pseudo code. [2015Spring]

→ Pseudo code is an informal way of programming description that does not require any strict programming language syntax or underlying technology considerations. It is used for creating an outline or a rough draft of a program.
Often, algorithms are represented with the help of pseudo codes as they can be interpreted by programmers no matter what their programming background or knowledge is.

Advantages of pseudo code

• It's one of the best approaches to start implementation of an algorithm.

•Acts as a bridge between the program and the algorithm or flowchart. Also works as a rough documentation, so the program of one developer can be understood easily when a pseudo code is written out.
•The main goal of a pseudo code is to explain what exactly each line of a program should do, hence making the code construction phase easier for the programmer.
• Pseudo code is understood by the programmers of all types.
•It enables the programmer to concentrate only on the algorithm part of the code development.

Q.9) Write the significance of algorithm and flowchart in programming. Draw a flowchart to input a number and check if it is prime or not.[2015 Spring]

Ans: First part[2014Fall] Qno.6, Second part[2015Fall] Qno.7

Q.10) Why C is called structured programming language? Compare and contrast High Level Programming Language and Low Level Programming Language.[2017Fall]

→ Firstly, let us know what structure programming is.
Structured programming (modular 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. Certain languages such as Ada, Pascal, and C are designed with features that encourage or enforce a logical program structure.
Structured programming frequently employs a top-down design model, in which developers map out the overall program structure into separate subsections. A defined function or set of similar functions is coded in a separate module or submodule, which means that code can be loaded into memory more efficiently and that modules can be reused in other programs. After a module has been tested individually, it is then integrated with other modules into the overall program structure.

Program flow follows a simple hierarchical model that employs looping constructs such as "for," "repeat," and "while." Use of the "Go To" statement is discouraged.

- C is called structured programming language as a program in C language can be divided into small logical functional modules or structures with the help of function procedure.
- 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, so, it is called structured programming language.

Similarities:

At some point both HLL and LLL are compiled down to and executed as a really low-level command set that a particular CPU understands.

Contrasting (Differences) chart

See [2013Fall] Qno.1

Q.11)What are the differences between primary and secondary memory? Explain the function of control unit with the help of block diagram of digital computer.[2016Fall]

→

S.N	Primary Memory	Secondary Memory
1.	It is the main memory where the data and information are stored temporarily.	It is the external memory where the data is stored permanently.
2.	Data is directly accessed by processing unit.	Data cannot be directly processed by processor
3.	It is a volatile memory meaning data cannot be retained in case of power failure.	It's a nonvolatile memory so data can be retained by even after power failure.

4.	Memory is stored in semiconductor chips which are relatively expensive.	Memory is stored in external storage device such as hard disk , flash drives, etc.
5.	It can be categorized into cache memory and Random Access Memory (RAM).	They are permanent storage devices such as CD,DVD,HDD,Floppy disk,etc.
6.	It is relatively faster than secondary memory because of its volatile nature.	It is usually slower than primary memory. It is like a backup memory.
7.	It holds data/information that is currently being used by the processing unit.	It stores substantial amount of data and information ranging from gigabytes to terabytes.

Q.12)Define the role of flowchart in efficient program maintenance with its character. Also develop a flow chart to print the even numbers between 150 to 500. [2016Fall]

→ First part [2014Fall] Qno.6

Second part:

Q.13)What is programming language? Why is High Level Language (HLL) preferred to Low level Language (LLL)?[2016Spring]

Ans: A high-level language (HLL) is a programming language such as C, FORTRAN, or Pascal that enables a programmer to write programs that are more or less independent of a particular type of computer.

Such languages are considered high-level because they are closer to human languages and further from machine languages.

A low-level language is a type of programming language that contains basic instructions recognized by a computer. Unlike high-level languages used by software developers, low-level code is often cryptic and not human-readable. Two common types of low-level programming languages are assembly language and machine language.

- The main advantage of high-level languages over low-level languages is that they are easier to read, write, and maintain.
- HLL is programmer friendly whereas LLL is machine friendly, which makes HLL understandable to programmers than LLL.
- HLL are portable and runnable in any platforms whereas LLL are not portable and machine dependent.
- We don't need hardware knowledge to write programs in HLL whereas hardware knowledge is must for to write program in LLL.
- Debugging and maintenance is simple in HLL whereas it is quite complex in LLL.

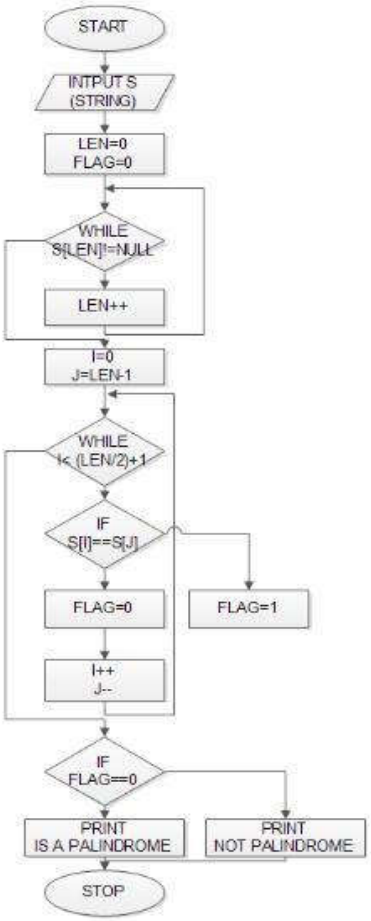
Q.14) What is the significance of algorithm and flowchart in programming? Write an algorithm and draw a neat flowchart to input a number and check it is palindrome or not.[Note: Palindrome number is same even after its reverse such as 989].[2016Spring]

Ans: [2014Fall] Qno.6

Second Part:

Algorithm:

Step 1: Input S (string)
Step 2: Len = 0 , Flag =0
Step 3: While (S[Len] != NULL)
Len++
Step 4: I = 0 , J = Len-1
Step 5: While (I < (Len/2)+1)
If (S[I] == S[J])
Flag=0
else
Flag=1
I++ , J–
Step 6: If (Flag == 0)
Print Key Is a Palindrome
else
Print Key Is Not a Palindrome
Step 7: End



Q.15) What do you mean by algorithm and flowchart? Write an algorithm and flowchart to find palindrome of given number.[2017Spring]

Ans: First part page …[2013Fall] Qno.3, Second Part [2017Fall] Qno.14

Q.16What do you mean by programming language? Discuss on machine language, assembly language and high level language.[2017Fall]

Ans: A programming language is a set of commands, instructions, and other syntax use to create a software program.

Machine Language :

A machine language is the language written as strings of binary 1's and 0's. It is the only language which a computer understands without using a translation program.

Disadvantages

1. It is machine dependent i.e. differs from computer to computer.
2. It is difficult to program and write.
3. It is prone to errors.
4. It is difficult to modify.

Assembly Language: It is a low level programming language that allows a user to write a program using alphanumeric mnemonics codes, instead of numeric codes for a set of instructions.

It requires a translator known as assembler to convert assembly language into machine language so that it can be understood by the computer. It is easier to remember and write than machine language.

Assembler-It is a computer program which converts or translates assembly language into machine language. It assembles the machine language program in the main memory of tha computer and makes it ready for execution.

Advantages:

1. It is easier to understand.
2. It is easier to locate and correct errors.
3. It is easier to modify.

Disadvantage: It is machine dependent.

High Level Language:

It is a machine dependent language. It enables a user to write program in a language which resembles English words and familiar mathematical symbols.

Each statement in high level language is a micro instruction which is translated into several machine language instructions.

A compiler is a translator program which translates a high level programming language into equivalent machine language program. It compiles a set of machine language instructions for every high level language program.

Source Code: It is the input (or the programming instructor) of a procedural language.

The compiler translates the source code into machine level language which is known as object code.

Source code-->Language Translator Program-->Object code

Linker: A program used with a compiler to provide links to the libraries needed for an executable program. It takes one or more object code generated by a compiler and combines them into a single executable program.

Interpreter: It is a language translator used for translating high level language into the desired output. It takes one statement, translates it into machine language instructions and then immediately executes the result .Its output is the result of program execution.

High level languages are programmer friendly. They are easy to write, debug and maintain.

Advantages:
- It provides higher level of abstraction from machine languages.
- It is machine independent language.
- Easy to learn.
- Less error prone, easy to find and debug errors.
- High level programming results in better programming productivity.

Disadvantages of High level language
1. It takes additional translation times to translate the source to machine code.
2. High level programs are comparatively slower than low level programs.
3. Compared to low level programs, they are generally less memory efficient.
4. Cannot communicate directly with the hardware.

Q.17)Write short notes on documentation.[2018]

Ans: Any written text, illustrations or video that describe a software or program to its users is called program document.
In modular programming documentation becomes even more important because different modules of the software are developed by different teams. If anyone other than the development team wants to or needs to understand a module, good and detailed documentation will make the task easier.

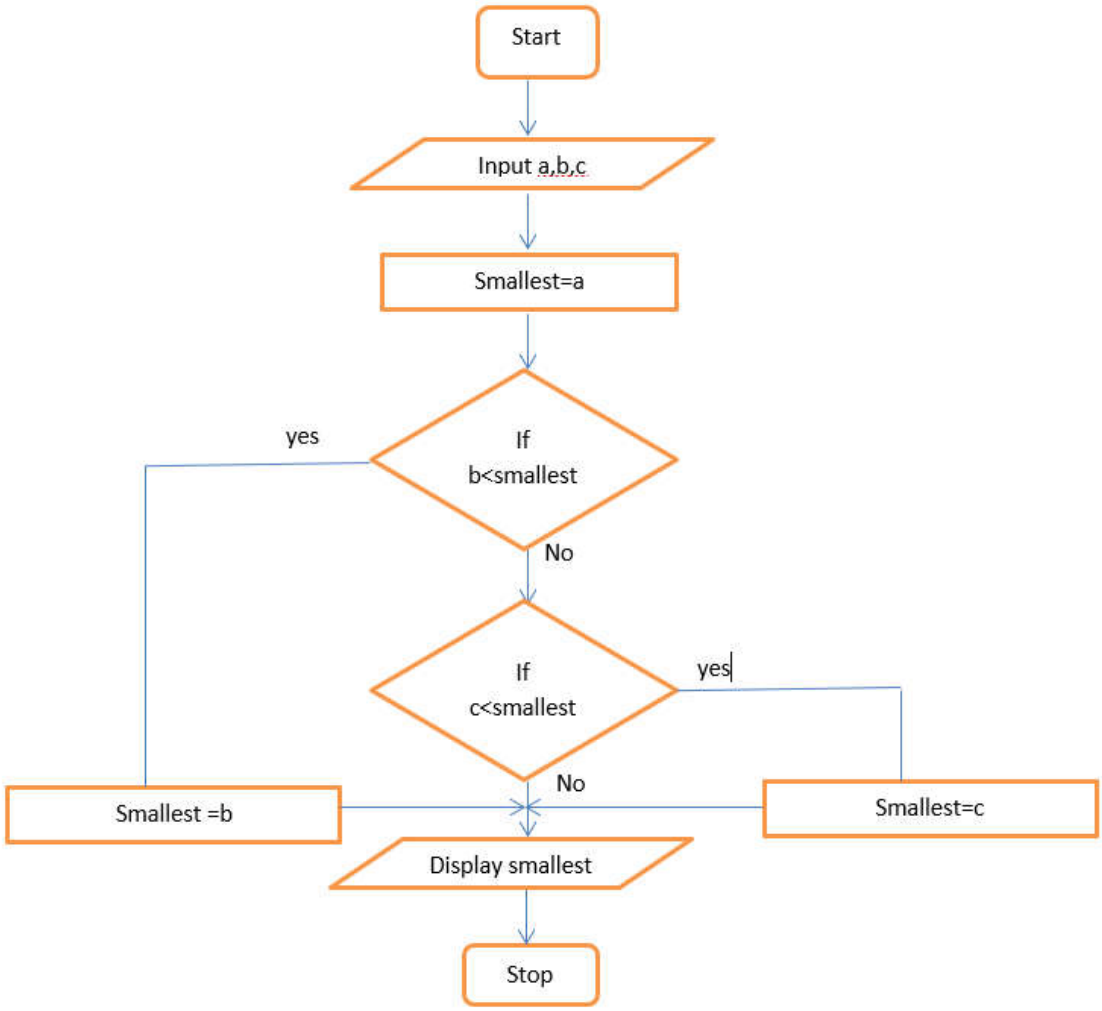
These are some guidelines for creating the documents −
1. Documentation should be from the point of view of the reader
2. Document should be unambiguous.
3. There should be no repetition.
4. Standards should be used.
5. Documents should always be updated.
6. Any outdated document should be phased out after due recording of the phase out.

Advantages of Documentation:
These are some of the advantages of providing program documentation:
1.Keeps track of all parts of a software or program
2.Maintenance is easier
3.Programmers other than the developer can understand all aspects of software.
4.Improves overall quality of the software
5.Assists in user training
6.Ensures knowledge de-centralization, cutting costs and effort if people leave the system abruptly

Q.18) Define algorithm and flowchart. Draw a flowchart to read three numbers from user and find the smallest one.[2017 Fall]

Ans:

Note: The questions of this chapter were solved by Maheshwor Acharya, Vice-president, Tech club



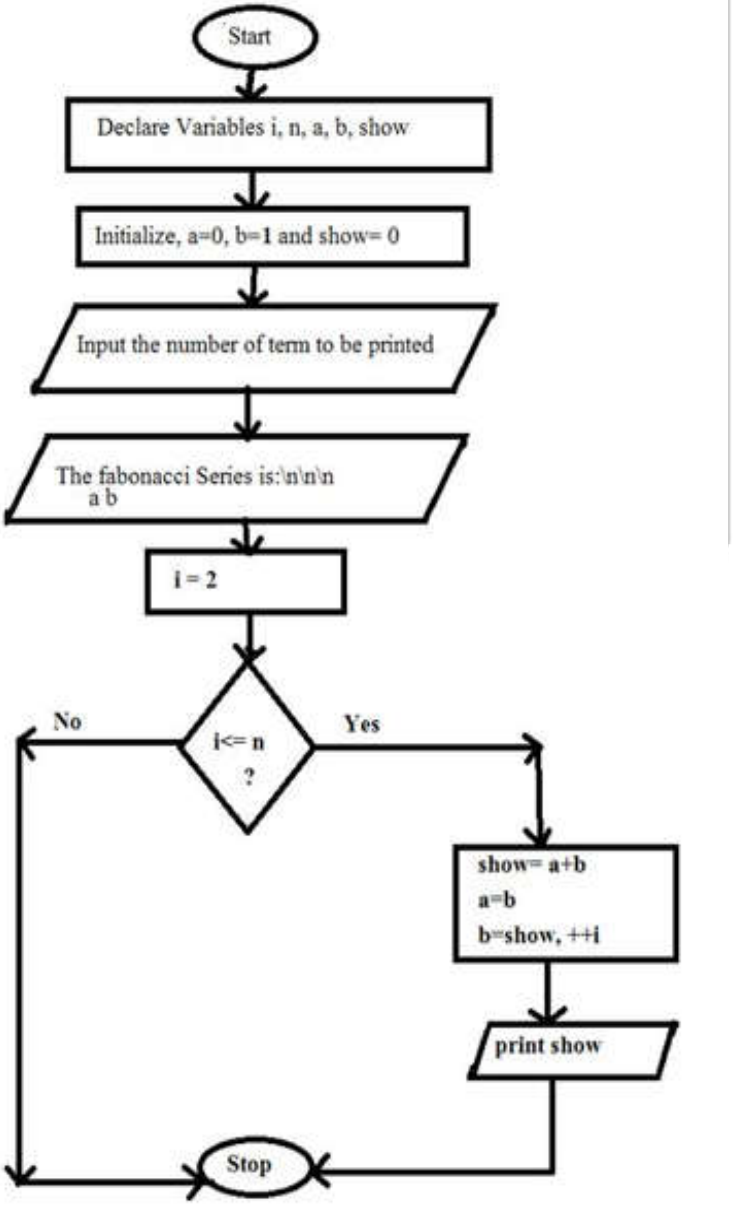
Q.19) Difference between compiler and interpreter:

S.N	Compiler	Interpreter
1.	Scans the entire program and translates it as a whole into machine code.	Translates program one statement at a time
2.	It takes large amount of time to analyze the source code but the overall execution time is comparatively faster	It takes less amount of time to analyze the source code but the overall execution time is slower.
3.	Generates intermediate object code which further requires linking, hence requires more memory	No intermediate object code is generated, hence are memory efficient.
4.	It generates the error message only after scanning the whole program. Hence debugging is comparatively hard.	Continues translating the program until the first error is met, in which case it stops. Hence debugging is easy.
5.	Program need not to be compiled every time.	Every time higher level program is converted to lower level program.
6.	Compiler takes entire program as input.	Interpreter takes single instruction as input.
7.	Programming language like C, C++ use compilers.	Programming language like Python, Ruby use interpreters.

Q.20)Write algorithm and draw flowchart to generate Fibonacci sequence of eight terms.[2018Fall]

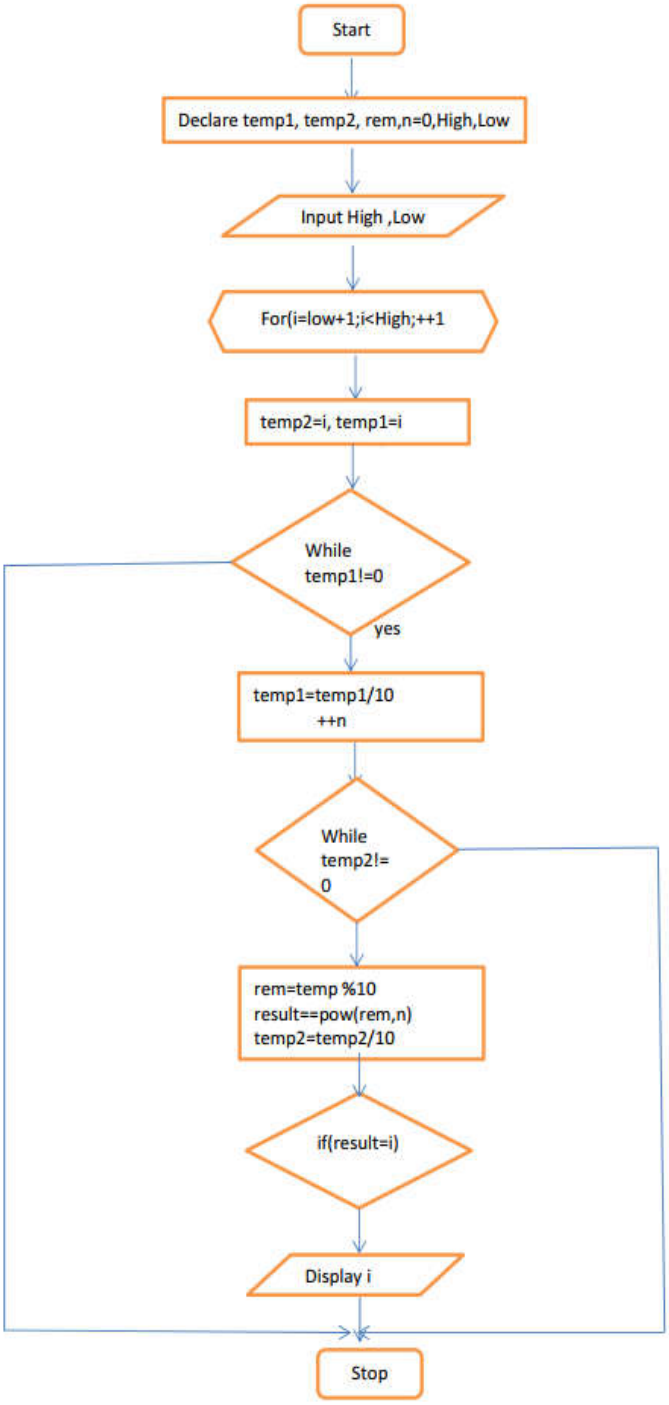
Ans: **Fibonacci Series Algorithm:**

Step1: Start
Step2: Declare variables i, a,b , show
Step3: Initialize the variables, a=0, b=1, and show =0
Step4: Enter the number of terms of Fibonacci series to be printed
Step5: Print first two terms of series
Step6: Use loop for the following steps
-> show=a+b
-> a=b
-> b=show
-> increase value of i each time by 1
-> print the value of show
Step7: End



Q.21Define the role of flowchart in efficient program maintenance with its character. Also develop a flowchart to print the Armstrong numbers between 150 and 500.[Spring2018]

Ans: First Part[Qno.6]



Note: The questions of this chapter were solved by **Maheshwor Acharya**[B.E. Computer], Vice-president, Tech club

Chapter 2: Variable & Data Types.

Q.1 Which of the following are invalid variable name and why? Fall (2019)

Minimum	First.name	Row Total	&name
Doubles	3rd_row	Column-total	float

Describe the four basic data-types along with their size and range.

➔A variable is nothing but a name given to a storage area that our programs can manipulate. Each variable in C has a specific type, which determines the size and layout of the variable's memory; the range of values that can be stored within that memory; and the set of operations that can be applied to the variable. The invalid variables are:

Row Total: It is invalid variable because in programming with c, there is a rule to avoid space while naming a variable.

Column-total: while naming a variable, you can only add a special character i.e. underscore (-).

&name: In programming with c, we cannot use ampersand (&) but a underscore is valid.

float: A float is a data type in C so, a variable named float is invalid in c programming.

Data types define the size and type of values to be stored in the computer memory, **Basic Data Types** are also known as **"primitive data types"** here are the few basic data types with their sizes in C language:

char: The most basic data type in C. It stores a single character and requires a single byte of memory in almost all compilers.

int: As the name suggests, an int variable is used to store an integer.

float: It is used to store decimal numbers (numbers with floating point value) with single precision.

double: It is used to store decimal numbers (numbers with floating point value) with double precision.

Different data types also have different ranges up to which they can store numbers. These ranges may vary from compiler to compiler. Below is list of ranges along with the memory requirement and format.

DATA TYPE	MEMORY (BYTES)	RANGE	FORMAT SPECIFIER
short int	2	-32,768 to 32,767	%d
unsigned short int	2	0 to 65,535	%u
unsigned int	4	0 to 4,294,967,295	%u
int	4	-2,147,483,648 to 2,147,483,647	%d
long int	4	-2,147,483,648 to 2,147,483,647	%ld
unsigned long int	4	0 to 4,294,967,295	%lu
signed char	1	-128 to 127	%c

DATA TYPE	MEMORY (BYTES)	RANGE	FORMAT SPECIFIER
unsigned char	1	0 to 255	%c
float	4	$3.4x10^{-38}$ to $3.4x10^{+38}$	%f
Double	8	$1.7x10^{-308}$ to $1.7x10^{+308}$	%lf
Long double	10	$3.4x10^{-4932}$ to $1.1x10^{+4932}$	%Lf

Q.2 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. [Spring 2018]

➔The following variables can be declared by following data types:

Mobile phone numbers: integer(int)

Address: character(char)

Body temperature: float

Salary: double

And rest are same as above question Q.1.

Q.3 What are different data types available in C? Explain their type's qualifier, conversion character, range of value and storage size in memory occupied by each type.

➔There are two types of qualifiers available in C language. They are:

1. CONST KEYWORD:

Constants are also like normal variables. But, only difference is, their values can't be modified by the program once they are defined. They refer to fixed values. They are also called as literals. They may be belonging to any of the data type.

Syntax:

```
const data_type variable_name; (or) const data_type *variable_name;
```

2. VOLATILE KEYWORD:

When a variable is defined as volatile, the program may not change the value of the variable explicitly.

But, these variable values might keep on changing without any explicit assignment by the program. These types of qualifiers are called volatile.

For example, if global variable's address is passed to clock routine of the operating system to store the system time, the value in this address keep on changing without any assignment by the program. These variables are named as volatile variable.

Syntax:

*volatile data_type variable_name; (or) volatile data_type *variable_name;*

And rest are same as above question fall 2019.

Q.4 What is an operator? Explain the conditional operator with suitable example. Fall 2017

➔Operators are the symbols that operates on certain data items to perform certain mathematical or logical manipulations.

Conditional operators: The operator pair **"?:"** is known as conditional operator. As, it takes 3 operands so, it general is called ternary operator. The general form of conditional operator is, Expression1? Expression2: Expression3

Example: C program to find largest among two numbers.

```
#include <stdio.h>
int main()
{
    int n1 = 5, n2 = 10, max; // variable declaration

    max = (n1 > n2) ? n1 : n2;  // Largest among n1 and n2

    printf("Largest number between"" %d and %d is %d. ", n1, n2, max);

    return 0;
}
```

Output: Largest number between 5 and 10 is 10.

Q.5 Why it is necessary to have a knowledge of data type in programming. Explain all types of datatype available in c. spring 2017

➔A data type, in programming, is a classification that specifies which type of value a variable has and what type of mathematical, relational or logical operations can be applied to it without causing an error. A string, for example, is a data type that is used to classify text and an integer is a data type used to classify whole numbers. The data type defines which operations can safely be performed to create, transform and use the variable in another computation.

And rest are same as above question fall Q.1.

Q.6 What is an operator? Explain the arithmetic, relational, logical and assignment operators in C language. Fall 2016

➔Operators are the symbols that operates on certain data items to perform certain mathematical or logical manipulations.

1. Arithmetic Operators **(+, -, *, /, %)**

An arithmetic operator performs mathematical operations such as addition, subtraction, multiplication, division etc on numerical values (constants and variables).

Operator	Description
+	adds two operands
-	subtract second operands from first
*	multiply two operand
/	divide numerator by denominator
%	remainder of division
++	Increment operator - increases integer value by one
--	Decrement operator - decreases integer value by one

2. Relational Operators **(<, >, <=, >=, ==, !=)**

A relational operator checks the relationship between two operands. If the relation is true, it returns 1; if the relation is false, it returns value 0.

Operator	Description
==	Check if two operand are equal
!=	Check if two operand are not equal.
>	Check if operand on the left is greater than operand on the right
<	Check operand on the left is smaller than right operand
>=	check left operand is greater than or equal to right operand
<=	Check if operand on left is smaller than or equal to right operand

3. Logical operators

C language supports following 3 logical operators. Suppose a = 1 and b = 0,

Operator	Description	Example
&&	Logical AND	(a && b) is false
||	Logical OR	(a || b) is true
!	Logical NOT	(!a) is false

4.Assignment operators **(+=, -=, *=, /=, %=)**
An assignment operator is used for assigning a value to a variable. The most common assignment operator is =.

Note: The questions of this chapter were Solved by Vishal Kanu[B.E. Computer], Treasurer, Tech Club

Operator	Description	Example
=	assigns values from right side operands to left side operand	a=b
+=	adds right operand to the left operand and assign the result to left	a+=b is same as a=a+b
-=	subtracts right operand from the left operand and assign the result to left operand	a-=b is same as a=a-b
*=	mutiply left operand with the right operand and assign the result to left operand	a*=b is same as a=a*b
/=	divides left operand with the right operand and assign the result to left operand	a/=b is same as a=a/b
%=	calculate modulus using two operands and assign the result to left operand	a%=b is same as a=a%b

Q.7 What is operator? Describe about the unary operator, binary operator and ternary operator with examples.

➔Operators are the symbols that operates on certain data items to perform certain mathematical or logical manipulations.

1.Unary Operators

Unary Operators are the operators which require single operand to perform any action.

Types of unary operators:

A. unary minus(-)

B. increment(++)

C .decrement(- -)

2.Binary Operators

Binary operators are the operators which require two operands to perform any action.
C offers different types of binary operators. they are given as follows

1.Arithmetic Operators

2.Logical Operators

3.Relational Operators

4.Bitwise Operators

5.Assignment Operators

6.Special Operators

3.Ternary or conditional Operator (? :)

There is one ternary operator supported by C language.The ternary operator is also called as "Conditional Operator".

Syntax: `Condition ? expression1 : expression2`
If condition is true expression1 gets executed else expression2.A ternary operator is a short form of if-else.

Q.8 Define operator in C. List out different types of operators used in C. Explain three of them with examples. Fall 2015

➔ same as question no.5

```
Example of arithmetic operator
#include <stdio.h>
int main()
{
    int a = 9,b = 4, c;
    c = a+b;
    printf("a+b = %d \n",c);
    c = a-b;
    printf("a-b = %d \n",c);
    c = a*b;
    printf("a*b = %d \n",c);
    c = a/b;
    printf("a/b = %d \n",c);
    c = a%b;
    printf("Remainder when a divided by b = %d \n",c);
    return 0;
}
```

Output:

```
a+b = 13
a-b = 5a*b = 36
a/b = 2
Remainder when a divided by b=1
```

Example of assignment operators

```
#include <stdio.h>
int main()
{
    int a = 5, c;
    c = a;                              // c is 5
    printf("c = %d\n", c);
    c += a;                            // c is 10
    printf("c = %d\n", c);
    c -= a;                           // c is 5
    printf("c = %d\n", c);
    c *= a;                          // c is 25
    printf("c = %d\n", c);
    c /= a;                         // c is 5
    printf("c = %d\n", c);
    c %= a;                        // c = 0
    printf("c = %d\n", c);
    return 0;
}
```

Output:

```
c = 5
c = 10
c = 5
c = 25
c = 5
c = 0
```

Example of relational operators

```
#include <stdio.h>
int main()
{
    int a = 5, b = 5, c = 10;
    printf("%d == %d is %d \n", a, b, a == b);
    printf("%d == %d is %d \n", a, c, a == c);
    printf("%d > %d is %d \n", a, b, a > b);
    printf("%d > %d is %d \n", a, c, a > c);
    printf("%d < %d is %d \n", a, b, a < b);
    printf("%d < %d is %d \n", a, c, a < c);
    printf("%d != %d is %d \n", a, b, a != b);
    printf("%d != %d is %d \n", a, c, a != c);
    printf("%d >= %d is %d \n", a, b, a >= b);
    printf("%d >= %d is %d \n", a, c, a >= c);
    printf("%d <= %d is %d \n", a, b, a <= b);
    printf("%d <= %d is %d \n", a, c, a <= c);

    return 0;
}
```

Output

```
5 == 5 is 1
5 == 10 is 0
5 > 5 is 0
5 > 10 is 0
5 < 5 is 0
```

```
5 < 10 is 1
5 != 5 is 0
5 != 10 is 1
5 >= 5 is 1
5 >= 10 is 0
5 <= 5 is 1
5 <= 10 is 1
```

Q.9 What are rules for naming identifiers? Why we need different data types in programming? Differentiate between local and global variable with suitable examples. Spring 2014

➔Variables are examples of identifiers. *Identifiers* are names given to identify *something*. There are some rules you have to follow for naming identifiers:

The first character of the identifier must be a letter of the alphabet (upper or lowercase) or an underscore ('_').

The rest of the identifier name can consist of letters (upper or lowercase), underscores ('_') or digits (0-9).

Identifier names are case-sensitive. For example, myname and myName are **not** the same. Note the lowercase n in the former and the uppercase N in the latter.

Examples of *valid* identifier names are i, __my_name, name_23 and a1b2_c3.

Examples of *invalid* identifier names are 2things, this is spaced out and my-name.

We need different data types in programming because programming is a classification that specifies which type of value a variable has and what type of mathematical, relational or logical operations can be applied to it without causing an error. A string, for example, is a data type that is used to classify text and an integer is a data type used to classify whole numbers. The data type defines which operations can safely be performed to create, transform and use the variable in another computation.

Parameter	Local	Global
Scope	It is declared inside a function.	It is declared outside the function.
Value	If it is not initialized, a garbage value is stored	If it is not initialized zero is stored as default.
Lifetime	It is created when the function starts execution and lost when the functions terminate.	It is created before the program's global execution starts and lost when the program terminates.
Data sharing	Data sharing is not possible as data of the local variable can be accessed by only one function.	Data sharing is possible as multiple functions can access the same global variable.
Parameters	Parameters passing is required for local variables to access the value in other function	Parameters passing is not necessary for a global variable as it is visible throughout the program

Q.10 What is the purpose of qualifiers register and volatile? Describe four basic data types. How could you extend the range of values they represent? Spring 2013

➔ C's volatile keyword is a qualifier that is applied to a variable when it is declared. It tells the compiler that the value of the variable may change at any time--without any action being taken by the code the compiler finds nearby.

Four basic data types: see question no.1

The range of values of can be extended by changing the data types. Such as if a variable declared by int data type It's range between -32768 to 32766. We can extend its range by change the type long or other. If we change long, then its range between -2147483468 to 2147483467.

Q.8 Mention the appropriate data type for storing following data. Also justify your answer in brief. Fall 2013

Distance jumped by a frog

A prime number between 5 and 555.

Weight of your body.

The examination symbol number of a student.

➔ There are basically four type of basic data types in C. i.e. integer(int), float, char, double.

char: The most basic data type in C. It stores a single character and requires a single byte of memory in almost all compilers.

int: As the name suggests, an int variable is used to store an integer.

float: It is used to store decimal numbers (numbers with floating point value) with single precision.

double: It is used to store decimal numbers (numbers with floating point value) with double precision.

The appropriate data type for storing following data are:

Distance jumped by a frog: This type of data can be stored in float because distance measured could be in decimal and could be very long so to store such type of data float is used in C programming. E.g. 22.73 meter.

A prime number between 5 and 555: This type of data can be stored in int(integer) because the prime numbers are of no decimal point and it ranges from 2 to 499. So a int data type is used.

Weight of your body: This type of data can be stored in float because weight of body consists of two unit's kilogram(kg) and gram(gm) where gram is represented after decimal. E.g. 55.23 kgThe examination symbol number of a student: This type of data can be stored in long int because the value could be a lengthy one and in C such data are stored in long int. E.g. 85624795631

Q.11 What do you mean by local and global variables? Give examples of each.

→ Local variable are those variable which only appears or works under the defined function and global variable is common for all functions which is defined in header section. One function alters the value of global variable then this change appears all of the functions under where it is used. But in the local variable case, does not affects other function variables.

Example:

```
#include<stdio.h>
float pi=3.14 //global variable defined
int main()
```

```
{
      int x=10; //local variable defined
      printf("\n %d",x);
      printf("\n%f",pi);
      myfunction();
      return 0;
}
myfunction()
{
      int x=20; //local variable defined
      printf("\n %d",x);
      printf("\n%f",pi);
  }
```

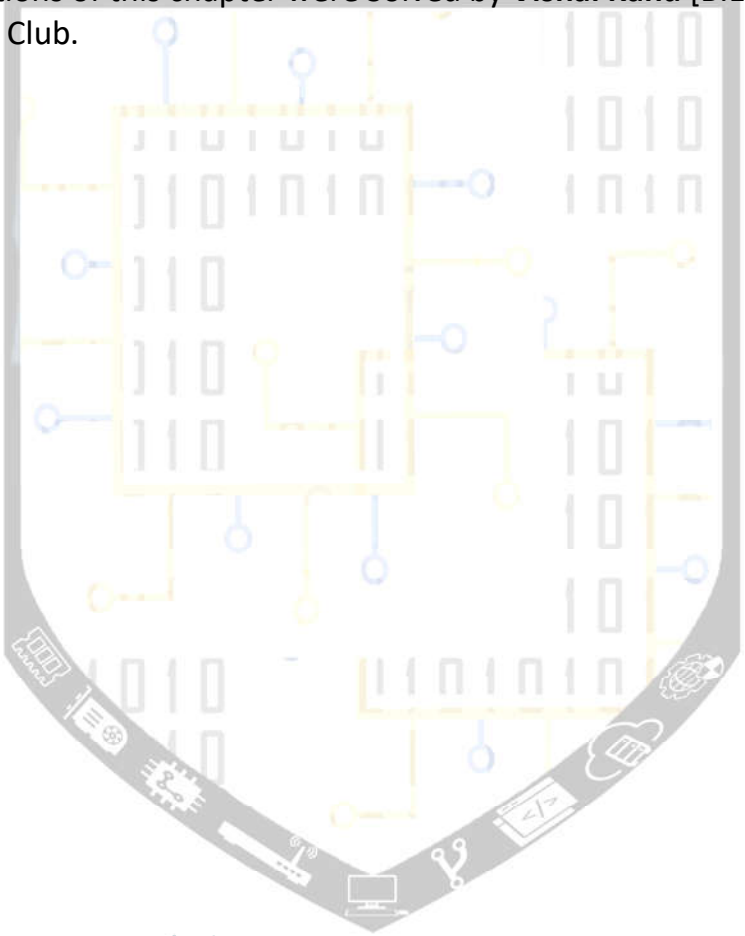

Q.12 What is operator precedence and associativity in C ?

➔ The following table lists the precedence and associativity of C operators. Operators are listed top to bottom, in descending precedence.

Precedence	Operator	Description	Associativity
1	++ --	Suffix/postfix increment and decrement	Left-to-right
	()	Function call	
	[]	Array subscripting	
	.	Structure and union member access	
	->	Structure and union member access through pointer	
	(*type*){*list*}	Compound literal(C99)	
2	++ --	Prefix increment and decrement	Right-to-left
	+ -	Unary plus and minus	
	! ~	Logical NOT and bitwise NOT	
	(*type*)	Type cast	
	*	Indirection (dereference)	
	&	Address-of	
	sizeof	Size-of	
	_Alignof	Alignment requirement(C11)	
3	* / %	Multiplication, division, and remainder	Left-to-right
4	+ -	Addition and subtraction	
5	<< >>	Bitwise left shift and right shift	
6	< <=	For relational operators < and ≤ respectively	
	> >=	For relational operators > and ≥ respectively	
7	== !=	For relational = and ≠ respectively	
8	&	Bitwise AND	
9	^	Bitwise XOR (exclusive or)	
10	|	Bitwise OR (inclusive or)	
11	&&	Logical AND	
12	||	Logical OR	

13	?:	Ternary conditional	Right-to-Left
14	= += -= *= /= %= <<= >>= &= ^= |=	Simple assignment Assignment by sum and difference Assignment by product, quotient, and remainder Assignment by bitwise left shift and right shift Assignment by bitwise AND, XOR, and OR	
15	,	Comma	Left-to-right

Note: The questions of this chapter were Solved by **Vishal Kanu** [B.E. Computer], Treasurer, Tech Club.



CHAPTER 3: CONTROL STRUCTURES

Qno1:Describe the working of for loop and while loop with flowchart and examples.[2013Fall]

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

Syntax:

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

Different Components of for loop

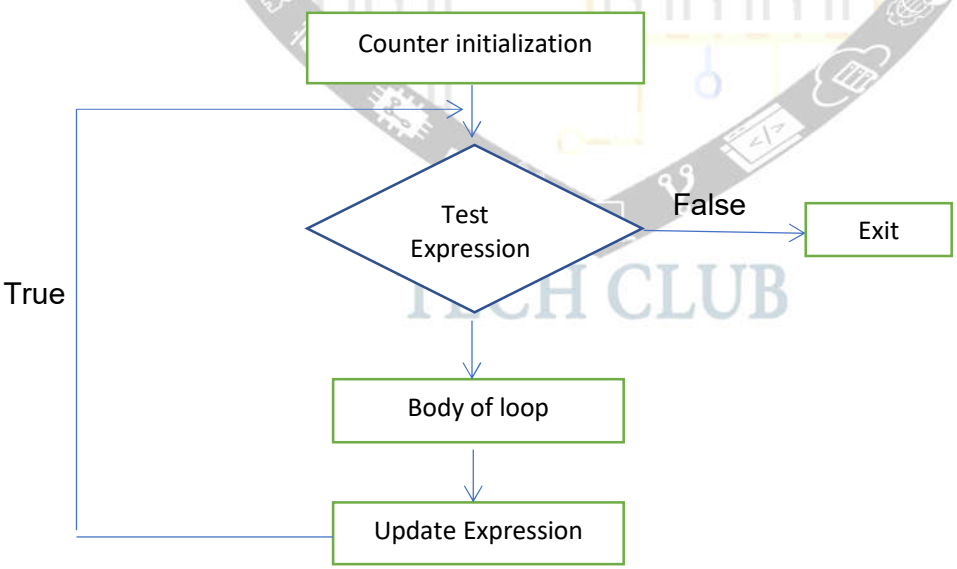
Counter initialization: The counter variable is initialized using assignment statement such as i=0 , j=1. Here, the variables I , j are is known as counter or loop controlled variables whose value controls the loop execution.

Test control: The value of counter variable is tested using test condition. If the condition is true, the body of the loop is executed, otherwise the loop is terminated and the execution continues with the statement that immediately follows the loop.

Update expression or increment/decrement expression : When the body of the loop is executed, the control is transferred back to the for statement after evaluating the last statement in the body of loop. The counter variable is updated either incremented or decremented and then only it is to be tested with test condition again and same process is repeated.

At first; counter is initialized to some value (i.e. step 1).Then counter variable is tested with test condition (i.e. step 2). If test is true, body of loop is executed (i.e. step 3). After finishing body, the counter variable is incremented or decremented (i.e. step 4) and then again updated counter variable is tested with test conditions (i.e. step 2). The same step is repeated as long as the condition is true. If the result with test condition is false, the control passes outside the loop i.e. the statement following the loop.

Flowchart



While Loop

A while loop in C programming repeatedly executes a target statement as long as a given condition is true.

Syntax:

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

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

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

Flowchart

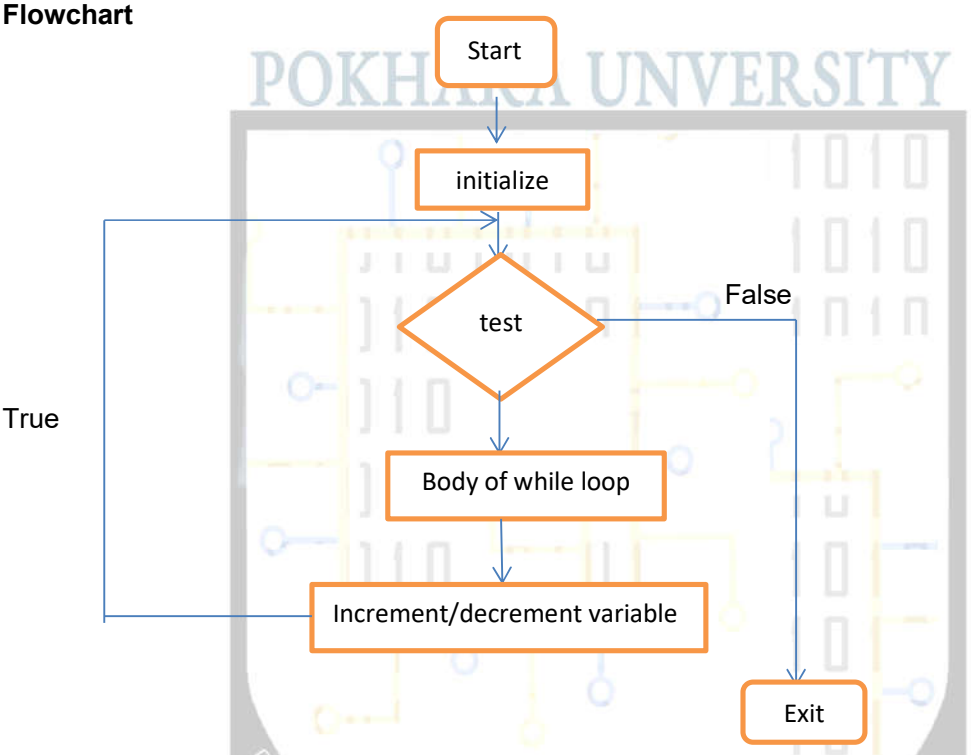


Fig: flowchart of while loop

Comparison of for **and** while **loop**

When the frequency of repetition is known in advance, the for loop is used and when it is not known, while loop is suitable.

```
int i;
for(i=0;i<10;i++)
{
//for loop body
}
```

The equivalent while loop is :

```
int i;
while(i<10)
{
//while loop body
i++
}
```

Q.n.2 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

Beyond 300 units……..Rs. 60 per unit
All users are also charged 'meter charge', which is equal to Rs. 50.
Write a complete C program to read the number of units consumed and print out the total charges.

➔

```
#include<stdio.h>
#include<conio.h>
#define metercharge 50
void main()
{
int amount=0,unit;
printf("\n enter total unit consumed");
scanf("%d",&unit);
if(unit<=100)
{
amount=metercharge+unit*40;
}
else if(unit<=300)
{
amount=metercharge+100*40+(unit-100)*50;
}
else
{
amount=metercharge+100*40+200*50+(unit-300)*60;
}
printf("\n your total amount is %d",amount);
getch();
}
```

Q.N.3 Explain with examples the decision control mechanism and loop control mechanisms available in c? 2008 Spring, 2008 Fall

OR

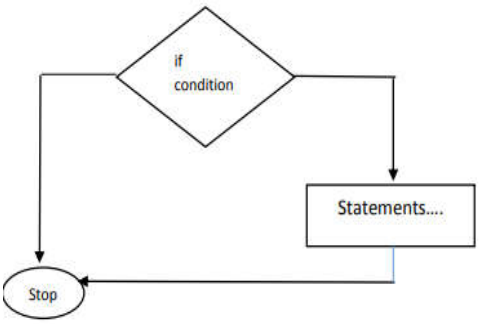
Explain entry control and exit control loop with examples? 2010,16,17 Fall, 2011, 16,18 Spring

Entry control is while loop & for loop and Exit control loop is do-while loop which is described below. Decision control mechanism involves making statement that control the flow of execution. Different decision control mechanism is:

1)* If statement*:

```
syntax:
if(test condition)
{
statement1}
……..
```

Note: The questions of this chapter were solved by Saroj Aryal[B.E. Software], Committee Member, Tech Club.



Example:

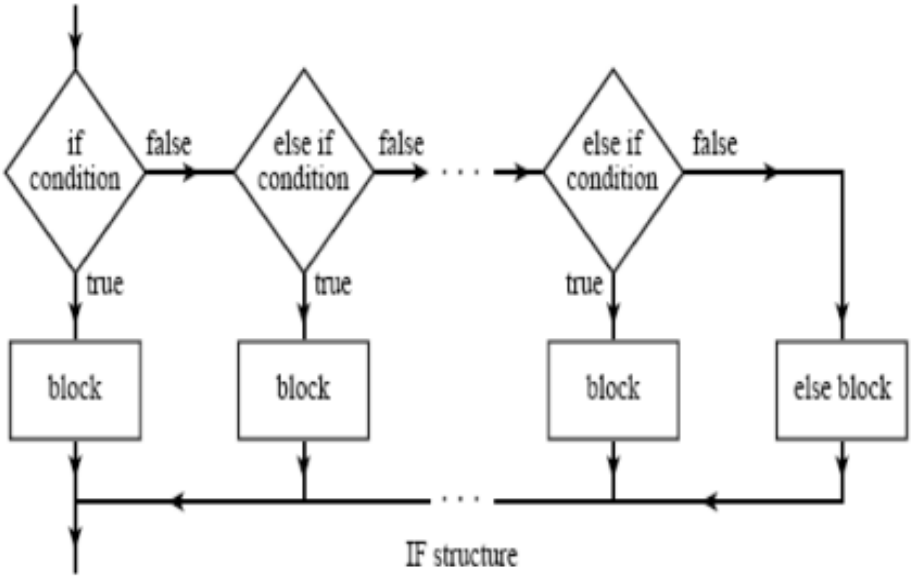
```
if ( 6 < 10 )
{
int x=1;
x+=1;
printf("\n %d",x);
}
```

2) If… else statement:

if...else statement is used if the programmer wants to execute some statement/s when the test expression is true and execute some other statement/s if the test expression is false.

Syntax:

```
if(test expression)
{
      True –block statement ;
}
else
{
      False-block statement;
}
Statement-X;
```

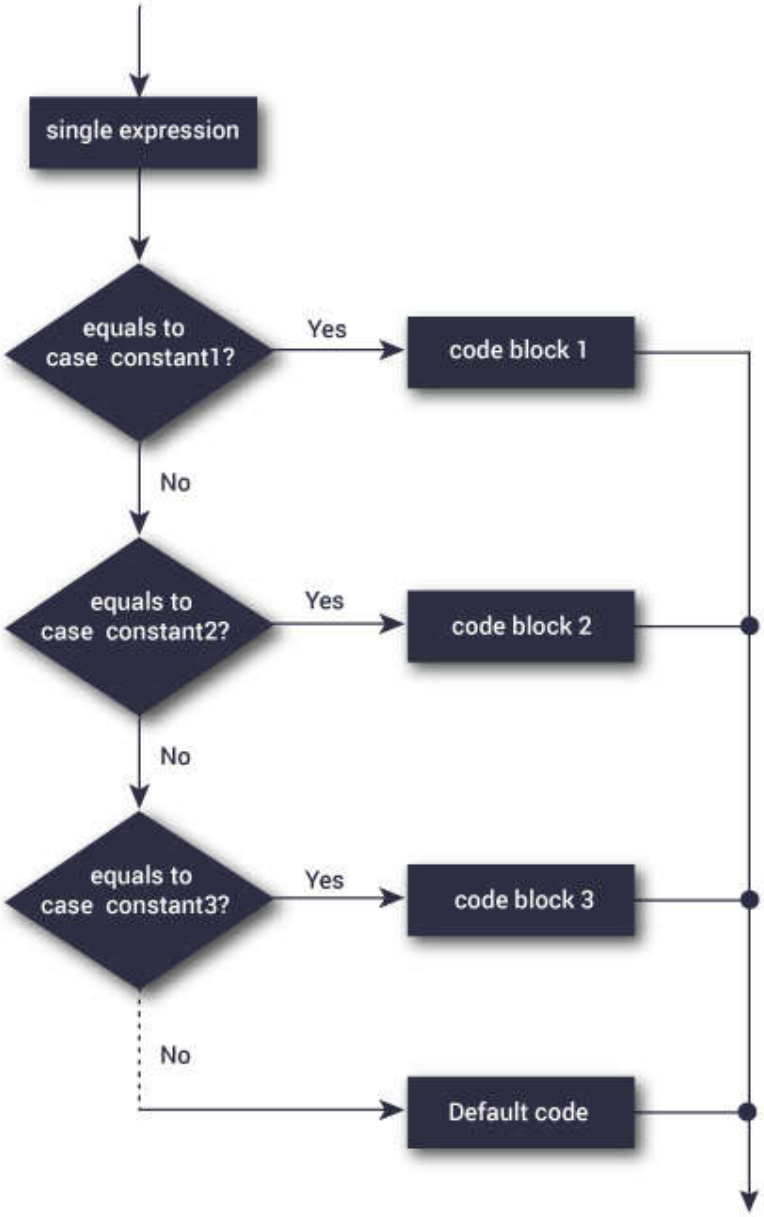


3) Switch statement :

The switch and case statements help control complex conditional and branching operations. The switch statement transfers control to a statement within its body.

```
switch (expression)
{
    case constant1:
      // statements
      break;

    case constant2:
      // statements
      break;
       .
       .
       .
      default:
        // default statements
  }
```

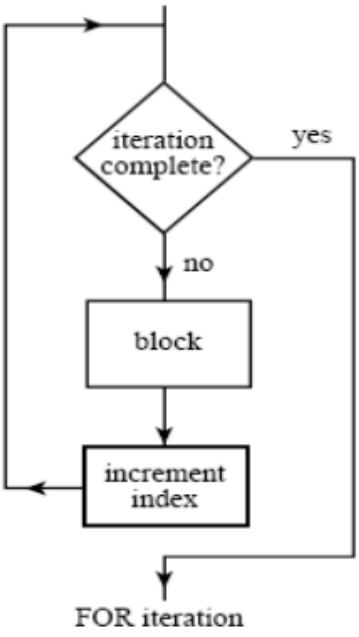


Loop control mechanism involves block of statements which are repeatedly executed for a certain no of times or until a particular condition is specified. Different loop control mechanisms are as follows.

4) For loop : 2009 fall

Execute a sequence of statements multiple times and abbreviates the code that manages the loop variable.
Syntax:

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

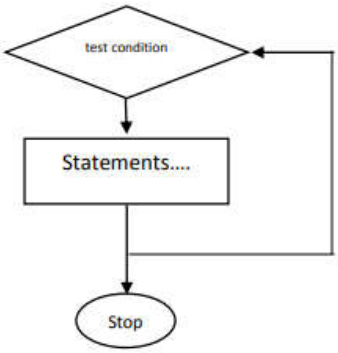


5) While loop(Entry Control): 2010 Spring

The while loop checks whether the test expression is true or not. If it is true, code/s inside the body of while loop is executed, that is, code/s inside the braces { } are executed. Then again the test expression is checked whether test expression is true or not. This process continues until the test expression becomes false.

Syntax:

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

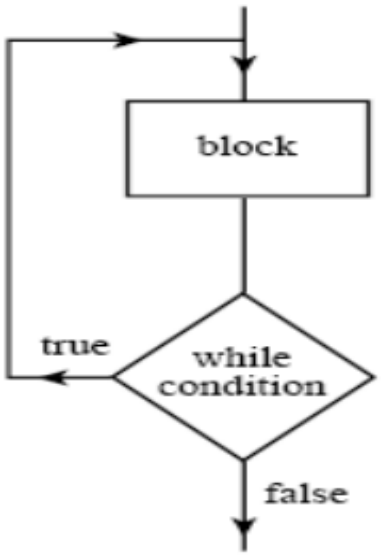


6) do while loop(Exit Control): 2010 Spring

In C, do...while loop is very similar to while loop. Only difference between these two loops is that, in while loops, test expression is checked at first but, in do...while loop code is executed at first then the condition is checked. So, the code are executed at least once in do...while loops.

Syntax:

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



Qno.4 Write a program to enter numbers until user press zero (0) and find the sum of all supplied numbers. 2010 Spring

```
#include<stdio.h>
#include<conio.h>
void main()
{
int sum=0,x;
printf("Enter the numbers:");
while(x!=0)
{
scanf("%d",&x);
sum+=x;
}
printf("Sum of nos:%d",sum);
getch();
}
```

Qno.5 Write a program to find the average of all the elements of m x n matrix. 2010 Spring

```
#include<stdio.h>
#include<conio.h>
void main()
{
int mat[50][50],m,n,i,j,sumEle=0,avg;
printf("Enter the row and colum of matrix");
```

```
scanf("%d%d",&m,&n);
for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
{
printf("\n Enter %d row and %d column element",i+1,j+1);
scanf("%d",&mat[i][j]);
}}
for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
{
sumEle+=mat[i][j];
}}
avg=sumEle/(m*n);
printf("\n Sum of the element of matrix:%d",avg);
getch();
}
```

Q.N.6 Write a program to find the terms in the given series till the term value is less than 250. (12+22)/3, (22+32)/4, (32+42)/5,..... 2011 Fall

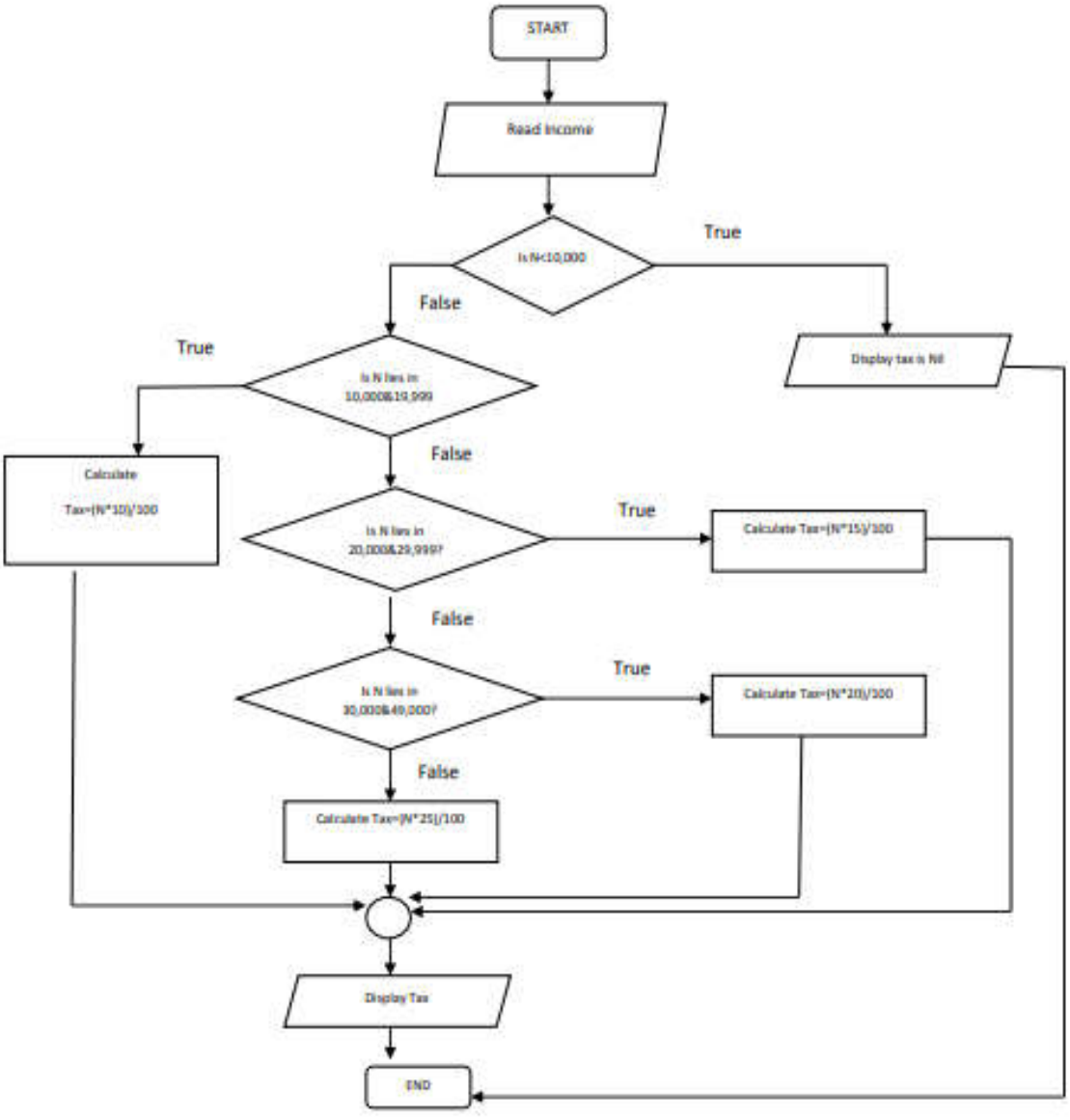
```
#include<stdio.h>
#include<conio.h>
#define expectedTerms 200
#include<math.h>
void main()
{
int i,sum;
clrscr();
for(i=1;i<expectedTerms;i++)
{
sum = (pow(i,2)+pow(i+1,2))/(i+2);
if(sum>250)
{
goto stop;
}
else
{
printf("%d term: %d \n",i,sum);
}
}
stop:
printf("\n Stopped after sum of trem %d reached 250",i);
getch();
}
```

Qno.7 Write a flow chart and program to compute tax amount when income is given considering the following tax table: 2006 Fall, 2010 Spring

Table of Tax rates

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

FlowChart:-



Code:

```
#include<stdio.h>
#include<conio.h>
void main()
{
float income,incomeAftrTax;
printf("Please enter your income:");
scanf("%f",&income);
if(income<10000)
{
printf("\n You do not have to pay Tax");
}
if(income>=10000&&income<20000)
{
incomeAftrTax=income+income*0.1;
printf("\n Your income after tax paying in %.2f",incomeAftrTax);
}
if(income>=20000&&income<30000)
{
incomeAftrTax=income+income*0.15;
printf("\n Your income after tax paying in %.2f",incomeAftrTax);
}
if(income>=30000&&income<50000)
{
incomeAftrTax=income+income*0.2;
printf("\n Your income after tax paying in %.2f",incomeAftrTax);
}
else
{
incomeAftrTax=income+income*0.25;
printf("\n Your income after tax paying in %.2f",incomeAftrTax);
}
getch();
}
```

Qno.8 Write a menu driven program having the given menu
Main Menu
1. To print factorial of a number
2. To reverse an integer arrary
3. Exit * Please select your choice (1 to 3):.....Note: The program should continue the execution till 3 is not selected* 2010 spring

Code:

```
#include<stdio.h>
#include<conio.h>
#define MAX 30
void main()
{
long int fac(long int n);
```

```
void reva();
int ch;
long int num;
clrscr();
while(1)
{
printf("1:fibonacci series\n");
printf("2:reverse an integer array\n");
printf("3:exit");
printf("\nenter the choice\n");
scanf("%d",&ch);
switch(ch)
{
case 1:printf("\nEnter number to calculate factorial");
scanf("%ld",&num);
printf("\n factorial of number%ld is %ld \t",num,fac(num));
beak;
case 2:asci();
break;
case 3:exit(0);
break;
default:printf("wrong choice\n");
break;
}
}
getch();
}
long int fac(long int n)
{
if(n<=0)
{
printf("please enter value greather than 0");
exit(0);
}
else if(n==1)
return 1;
else
return(n*fac(n-1));
}
void reva
 {
 int size, i, arr[MAX];
 int *ptr;
 clrscr();
 ptr = &arr[0];
 printf("\nEnter the size of array : ");
 scanf("%d", &size);
 printf("\nEnter %d integers into array: ", size);
 for (i = 0; i < size; i++) {
 scanf("%d", ptr);
 ptr++;
 }
 ptr = &arr[size - 1];
 printf("\nElements of array in reverse order are :");
```

```
for (i = size - 1; i >= 0; i--) {
printf("\nElement%d is %d : ", i, *ptr);
ptr--;
}
```

Qno.9 Write a program to calculate the sum of given series, up to the term given by user

$$Y = \frac{1}{2} - \frac{2}{x^2} + \frac{3}{x^3} - \ldots\ldots\ldots$$

2008 Fall

Code:

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
int n,x;
float sum=0;
printf("\n how many terms");
scanf("%d",&n);
printf("\n enter the value of x");
scanf("%d",&x);
if(n<2)
sum=0.5;
else
{
for(int i=2;i<=n;i++)
{
sum=float(sum+pow((-1),(i+1))*i/(pow(x,i)));
}

sum=float(sum+0.5);
}
printf("\n the sum of the upto %dth term is %f",n,sum);
getch();
}
```

Qno.10 Write a program to display the following menu:
MENU
i. To check if the given number is odd or even
ii. To find the sum of N numbers
iii. To find the sum of squares of two numbers
iv. Exit from program
And perform tasks as per user's choice until the user wants to exit. 2008 Spring

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#include<stdlib.h>
void main()
{
int ch,no,choice,n,sum=0,nsq1,nsq2,sqr,i;
```

```
clrscr();
while(1)
{
printf("\n -------MENU------");
printf("\n \t 1.Find Even and Odd \n\t 2.Find Sum \n\t 3.Find Sum of
square \n\t 4.Exit");
printf("\n Please Enter your choice:");
scanf("%d",&choice);
switch(choice)
{
case 1:
printf("\n Enter a no find even or odd");
scanf("%d",&no);
if(no%2==0)
{
printf("\n Given no is even");
}
else
printf("\n Given no is odd");
break;
case 2:
printf("\n please enter no to find sum up to that no:");
scanf("%d",&n);
for(i=0;i<=n;i++)
{
sum+=i;
}
printf("\n Sum up to %d: %d",n,sum);
break;
case 3:
printf("\n Enter two no to find their sum of square:");
scanf("%d %d",&nsq1,&nsq2);
sqr=pow(nsq1,2)+pow(nsq2,2);
printf("\n Sum of square:%d",sqr);
break;
case 4:
exit(0);
}
}
getch();
}
```


Qno.11 What do you mean by selective and repetitive statement? Why do we need break and continue statement. **2011 Fall, 2018 Spring**

A selection statement is a control statement that allows choosing between two or more execution paths in program. Most modern programming languages include one-way, two-way and n-way (multiple) selectors.

- An if statement with no else is a one-way selector
- An if-else statement is a two-way selector
- A switch or case statement is an n-way selector

Repetitive statement are those which are used to execute one statement or multiple statements more than one times. For, while and do… while loop are the repetitive statements in C.The break statement allows us to exit a loop from any point within its body, bypassing its normal termination expression. When the break statement is encountered inside a loop, the loop is immediately terminated, and program control resumes at the next statement following the loop. Break also can be used in conjunction with functions and case statements.
Continue statement forces the next iteration of the loop to take place, skipping any code in between itself and the test condition of the loop. In a while and do-while loops, a continue statement will cause control to go directly to the test condition and then continue the looping process. In the case of the for loop, the increment part of the loop continues. One good use of continue is to restart a statement sequence when an error occurs. By these reasons we need break and continue statements.

Q.N.12 Write program for armstrong number between 100 to 999. 2014 Fall (BCA)
2018 Spring

Code:

```
#include<stdio.h>
#include<conio.h>
void main()
{
int i,n,b,s;
clrscr();
printf("Armstrong number are :=\n");
for(i=100;i<=999 ;i++)
{
s=0;
n=i;
for(;n>0;n/=10)
{
b=n%10;
s=s+b*b*b;
}
if(s==i)
{
printf("%d\n",i);
}
}
getch();
}
```


Qno.13 Differentiate between *break* and *continue* statements with example.2018 Fall

Both “break” and “continue” are the ‘jump’ statements, that transfer control of the program to another part of the program. The main difference between break and continue is that break is used for immediate termination of loop. On the other hand, ‘continue’ terminate the current iteration and resumes the control to the next iteration of the loop.

break	continue
It terminates the current loop and executes the remaining statement outside the loop.	It terminates the current iteration and transfer the control to the next iteration in the loop.
syntax: break	syntax: continue

for i in "welcome":
  if(i=="c"):
  break
  print(i)
	for i in "welcome":
	  if(i=="c"):
	  continue
	  print(i)

w e l	w e l o m e

Qno.14 WAP to print Fibonacci Sequence of given terms. 2018 Fall

```
#include <stdio.h>
int main()
{
 int i, n, t1 = 0, t2 = 1, nextTerm;
 printf("Enter the number of terms: ");
 scanf("%d", &n);
 printf("Fibonacci Series: ");
 for (i = 1; i <= n; ++i)
    {
        printf("%d, ", t1);
        nextTerm = t1 + t2;
        t1 = t2;
        t2 = nextTerm;
    }
    return 0;
}
```

Pattern Printing

1.

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

Code:

```
#include<stdio.h>
int main() {
    int i, j, rows;
    printf("Enter number of rows: ");
    scanf("%d", &rows);
```

```
    for (i=1; i<=rows; ++i) {
        for (j=1; j<=i; ++j)
        { printf("* "); }
        printf("\n");
    }
    return 0;
}
```

2.Program to print half pyramid using alphabets

A
B B
C C C
D D D D
E E E E E

```
#include<stdio.h>
int main() {
    int i, j;
    char input, alphabet='A';
    printf("Enter the uppercase character you want to print in last row: ");
    scanf("%c", &input);
    for (i=1; i<=(input-'A'+1); ++i) {
        for (j=1; j<=i; ++j)
        { printf("%c", alphabet); }
        ++alphabet;
        printf("\n");
    }
    return 0;
}
```

3.Programs to print inverted half pyramid using * .

```
* * * * *
* * * *
* * *
* *
*
#include<stdio.h>
int main() {
    int i, j, rows;
    printf("Enter number of rows: ");
    scanf("%d", &rows);
    for (i=rows; i>=1; --i) {
        for (j=1; j<=i; ++j)
        { printf("* "); }
        printf("\n");
    }
    return 0;
}
```

4.Program to print full pyramid using *.

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

```
#include<stdio.h>
int main() {
    int i, space, rows, k=0;
    printf("Enter number of rows: ");
    scanf("%d", &rows);
    for (i=1; i<=rows; ++i,k=0) {
        for (space=1; space<=rows-i; ++space)
        { printf("  "); }
        while (k!=2*i-1) {
            printf("* ");
            ++k;
        }
        printf("\n");
    }
    return 0;
}
```

Using Numbers.

```
    1
  2 3 2
 3 4 5 4 3
4 5 6 7 6 5 4
5 6 7 8 9 8 7 6 5
```

```
#include<stdio.h>
int main() {
    int i, space, rows, k=0, count=0, count1=0;
    printf("Enter number of rows: ");
    scanf("%d", &rows);
    for (i=1; i<=rows; ++i) {
        for (space=1; space<=rows-i; ++space) {
          printf("  ");
          ++count;
        }
        while (k!=2*i-1) {
            if (count <= rows-1)
         { printf("%d ", i+k);
              ++count;
            }
            else {
              ++count1;
```

```
            printf("%d ", (i+k-2*count1));
          }
          ++k;
        }
        count1=count=k=0;
        printf("\n");
    }
    return 0;
}
```

5.Print Pascal's triangle

```
          1
        1   1
      1   2   1
    1   3   3   1
  1  4    6   4   1
1  5   10   10  5   1
```

```
#include<stdio.h>
int main() {
    int rows, coef=1, space, i, j;
    printf("Enter number of rows: ");
    scanf("%d", &rows);
    for (i=0; i<rows; i++) {
        for (space=1; space <= rows-i; space++)
            printf("  ");
        for (j=0; j<=i; j++) {
            if (j==0 || i==0)
                coef = 1;
            else
                coef=coef*(i-j+1)/j;
            printf("%4d", coef);
        }
        printf("\n");
    }
    return 0;
}
```

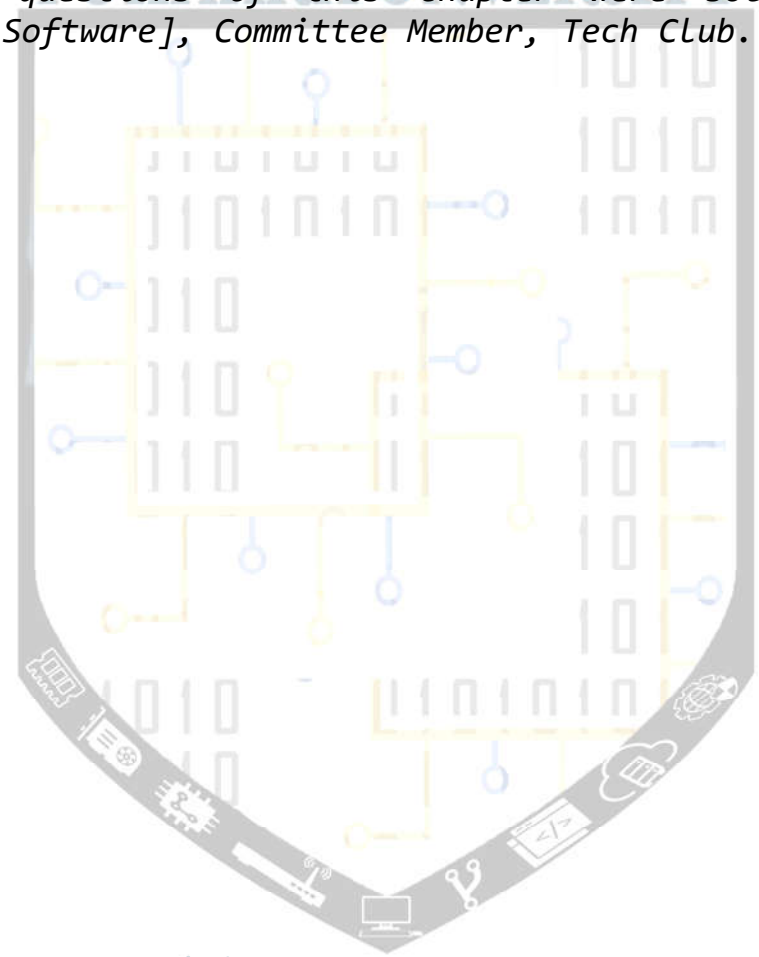
6.Print Floyd's Triangle.

```
1
2 3
4 5 6
7 8 9 10
```

```
#include<stdio.h>
int main() {
    int rows, i, j, number= 1;
    printf("Enter number of rows: ");
    scanf("%d", &rows);
    for (i=1; i<=rows; i++) {
```

```
            for (j=1; j<=i; ++j)
            { printf("%d ", number);
              ++number;
            }
            printf("\n");
        }
        return 0;
    }
```

Note: The questions of this chapter were solved by **Saroj Aryal***[B.E. Software], Committee Member, Tech Club.*



CHAPTER 4 : ARRAYS AND STRINGS

1. Write a program to read a one dimensional array, sort the number in ascending and decending order and display sorted number.

→

```
#include <stdio.h>
int main()
{
    int a[100],n,i,j;
    printf("Array size: ");
        scanf("%d",&n);
        printf("Elements: ");

      for(i=0;i<n;i++)
    {
        scanf("%d",&a[i]);
    }
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            if (a[j] > a[i])
            {
                int tmp = a[i];
                a[i] = a[j];
                a[j] = tmp;
            }
        }
    }
    printf("\n\nAscending : ");
    for (int i = 0; i < n; i++)
    {
        printf(" %d ", a[i]);
    }
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            if (a[j] < a[i])
            {
                int tmp = a[i];
                a[i] = a[j];
                a[j] = tmp;
            }
        }
    }
    printf("\n\nDescending : ");
    for (int i = 0; i < n; i++)
    {
        printf(" %d ", a[i]);
    }
    return 0;
}
```

2. WAP to add two 3x3 matrix. Display the sum stored in third matrix. [2018 spring]

→

```
#include<stdio.h>
int main()
{
 int matA[3][3],matB[3][3],matC[3][3];
 int r,c,k;
 for(r=0; r<3; r++)
 {
  for(c=0; c<3; c++)
  {
    printf("Enter first matrix : ");
    scanf("%d", &matA[r][c]);
  }
 }
 for(r=0; r<3; r++)
 {
  for(c=0; c<3; c++)
  {
    printf("Enter second matrix : ");
    scanf("%d", &matB[r][c]);
  }
 }
 for(r=0; r<3; r++)
 {
  for(c=0; c<3; c++)
  {
    matC[r][c]=0;
    for(k=0; k<3;k++)
       matC[r][c] = matA[r][c] + matB[r][c];
  }
 }
 printf("\n New addition matrix : \n");
 for(r=0; r<3; r++)
 {
  for(c=0; c<3; c++)
     printf(" %d",matC[r][c]);
  printf("\n");
 }
 return 0;
}
```

3. Describe string. Explain string handling function with examples. [2016 fall]

→

String is an array of characters i.e. characters are arranged one after another in memory.

The strings are used to manipulate text such as words and sentences. A string is always terminated by a NULL character i.e. \0. The terminating null character \0 is important because it is the only way for string handling functions to know the end of string. Normally each character is stored in one byte of memory, and successive characters of the string are stored in successive bytes.

String is initialized as:

char n[ ] = “Hello”;

n[0]	n[1]	n[2]	n[3]	n[4]	n[5]

H	e	l	l	o	\0

Here, the string is initialized to string variable n.

The characters of the string are enclosed within a pair of double quotes.

Example:

```
#include <stdio.h>
int main()
{
      char name[20];
      printf("Enter a string: );
      scanf("%s", name);
      printf("Your name is %s.", name);
      return 0;
}
```

The string handling function used in C programming are:

i) strlen(): It returns the length of the string (number of character in string).

Syntax:

strlen(string);

Example:

```
char str[] = "Tech Club";
int strLength;
strLength = strlen(str);
```

ii) strcpy(): It is used to copy one string to another. The Destination String should be a variable and Source_String can either be a string constant or a variable.

Syntax:

strcpy(Destination_String,Source_String);

Example:

```
char Destination_String[];
char Source_String[] = "Respect seniors";
strcpy(Destination_String,Source_String);
printf("%s", Destination_String);
```

iii) strcat(): It is used to concatenate two strings. The destination string should be a variable and source string can either be a string constant or a variable.

Syntax:

strcat(Destination_String, Source_String);

Example:

```
char Destination_String []="Respect ";
char Source_String[] = "Seniors";
strcat(Destination_String, Source_String);
puts( Destination_String);
```

iv) strcmp(): It is use two compare two strings. strcmp() function does a case sensitive comparison between two strings. The Destination_String and Source_String can either be a string constant or a variable.

Syntax:

strcmp(string1, string2);

Example:

```
char string1[] = "Kusal ";
char string2[] = "Kaushal";
int ret;
ret=strcmp(string1, string2);
printf("%d",ret);
```

Q. What is an array? Write a C programming using array to find largest and smallest number form a list of 100 given numbers.

→ An array is a data structure that store a number of data items as a single object. Array is used when multiple data items that have common characteristics are required in a program.

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

First element				**Last element(n^{th} element)**
n[0]	n[1]	n[2]	n[3]	…….

Syntax to declare array:

data_type array_name [array_size];

example:

```
int arr[48];
```

Program Solution:

```
#include<stdio.h>

int main()
{
    int a[100],i,large,small;

    printf("Enter the Array:");

    for(i=0;i<100;++i)
       {
            scanf("%d",&a[i]);
        }

    large=small=a[0];
    for(i=1;i<100;++i)
    {
        if(a[i]>large)
            large=a[i];
        if(a[i]<small)
            small=a[i];
    }

    printf("The largest element is %d",large);
    printf("\nThe smallest element is %d",small);

    return 0;
}
```

Q. Write a program to insert a given character at the 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". [2013 fall]

→

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

int main()
{
     char a[10];
     char b[10];
     char c[10];
     int p=0,r=0,i=0;
     int t=0;
     int x,g,s,n,o;

     puts("Enter String:");
     scanf("%s",&a);
     puts("Enter string to be inserted:");
     scanf("%s",&b);
```

```
        printf("Enter the position where the string has to be inserted: ");
        scanf("%d",&p);
        r = strlen(a);
        n = strlen(b);
        i=0;

        // Copying the input string into another array
        while(i <= r)
        {
                c[i]=a[i];
                i++;
        }
        s = n+r;
        o = p+n;

        // Adding the sub-string
        for(i=p;i<s;i++)
        {
                x = c[i];
                if(t<n)
                {
                        a[i] = b[t];
                        t=t+1;
                }
               a[o]=x;
               o=o+1;
        }

        printf("%s", a);

        return 0;
}
```

Q. Write a program to test, whether given two matrices are equal or not. [2013 fall]

→

```
    #include <stdio.h>

    int main()
    {
        int a[10][10], b[10][10];
        int i, j, row1, column1, row2, column2, flag = 1;

        printf("Enter the order of the matrix A \n");
        scanf("%d %d", &row1, &column1);


        printf("Enter the order of the matrix B \n");
        scanf("%d %d", &row2, &column2);

        printf("Enter the elements of matrix A \n");
        for (i = 0; i < row1; i++)
        {
```

```
    for (j = 0; j < column1; j++)
    {
        scanf("%d", &a[i][j]);
    }
}

printf("Enter the elements of matrix B \n");
for (i = 0; i < row2; i++)
{
    for (j = 0; j < column2; j++)
    {
        scanf("%d", &b[i][j]);
    }
}

printf("MATRIX A is \n");
for (i = 0; i < row1; i++)
{
    for (j = 0; j < column1; j++)
    {
        printf("%3d", a[i][j]);
    }
    printf("\n");
}

printf("MATRIX B is \n");
for (i = 0; i < row2; i++)
{
    for (j = 0; j < column2; j++)
    {
        printf("%3d", b[i][j]);
    }
    printf("\n");
}

if (row1 == row2 && column1 == column2)
{
    printf("Matrices can be compared \n");
    for (i = 0; i < row1; i++)
    {
        for (j = 0; j < column2; j++)
        {
            if (a[i][j] != b[i][j])
            {
                flag = 0;
                break;
            }
         }
    }
}

else
{
    printf(" Cannot be compared\n");}
```

```
        if (flag == 1)
            printf("Two matrices are equal \n");

        else
            printf("But, two matrices are not equal \n");

      return 0;
    }
```

Q. 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. [2013 fall]

→

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

   int arr[10], i, s, j, p, sum = 0;

   printf("Enter Size of An Array :");
   scanf("%d", &s);

   printf("\nEnter Array Elements :");
   for (i = 0; i < s; i++) {
       scanf("%d", &arr[i]);
   }

   for (i = 0; i < s; i++) {
       j = 2;
       p = 1;
       while (j < arr[i]) {
           if (arr[i] % j == 0) {
               p = 0;
               break;
           }
           j++;
       }
       if (p == 1)
           sum = sum + arr[i];
   }

   printf("\nSum of Prime Numbers is :%d", sum);

   return 0;
}
```

Q. 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. For e.g. LIRIL, MADAM etc) [2013 spring]

→

```
#include <stdio.h>
int main()
{
  char text[100];
  int begin, middle, end, length = 0;
  printf("Enter a string: ");
  scanf("%s", &text);

  while (text[length] != '\0')
    length++;

  end = length - 1;
  middle = length/2;

  for (begin = 0; begin < middle; begin++)
  {
    if (text[begin] != text[end])
    {
      printf("Not a palindrome.\n");
      break;
    }
    end--;
  }
  if (begin == middle)
    printf("Palindrome.\n");

  return 0;
}
```

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

```
#include<stdio.h>
#include<string.h>
int main()
{
   int i,j,count;
   char str[25][25],temp[25];
   puts("Enter total no of Employee: ");
   scanf("%d",&count);

   puts("Enter Employees name one by one: ");
   for(i=0;i<count;i++)
   {
      scanf("%s", &str[i]);
   }

   for(i=0;i<count;i++)
   {
```

```
      for(j=i+1;j<count;j++)
      {
         if(strcmp(str[i],str[j])>0)
         {
            strcpy(temp,str[i]);
            strcpy(str[i],str[j]);
            strcpy(str[j],temp);
         }
      }
    }

   printf("Employees name in alphabetical order: \n");
   for(i=0;i<count;i++)
   {
      puts(str[i]);
   }

   return 0;
}
```

Q. Why array is important in programming? How can you initialize different types of arrays? Explain 2-dimensional array in C. [2014 spring]

→

Arrays a kind of data structure that can store a fixed-size sequential collection of elements of the same type.

- An array is used to store a collection of data or as a collection of variables of the same type.
- It is used to represent multiple data items of the same type by using only a single name.
- Arrays are more efficient and beneficial when compared to linked lists and hash tables.
- Faster and can be utilized anywhere.
- Store data of similar data types together and can be used anywhere in the code.

There are two types of array in C programming:

Single Dimension array: It is used to store data in a linear form. It have only one subscript variable.

The array initialization may be of three types:

1. `int a[5] = {1,2,3,4,5};`

Here a is integer type array which has 5 elements and their values are assigned as a[0] = 1, a[1] = 2, a[3] =5 and a[4] = 8.

2. `int b[ ] = {2,3,4};`

Here size of array is not given, the compiler automatically sets it size according to the number of values given.

3. `int c[10] = {1,2,3,4,5};`

In this situation all individual elements that are not assigned contains zero as initial values. So, value of c[5], c[6], c[7], c[8] and c[9] is zero.

2-dimensional array: The 2-dimensional array is organized as matrices which can be represented as the collection of rows and columns. It provides ease of holding the bulk of data at once.

Syntax to declare two dimensional array:

data_type array_name [rows][columns];

example:

```
int respect[2][3];
```

In 2-dimesional arry, we must define at least second dimension of array. 2-dimesional array can be initialized as:

```
int respect[2][3] = { {2,4,6}, {8,10,12}};
```

Q. Write a program to read a matrix and find the sum of all the digits in its main diagonal. [2015 fall]

→

```
#include<stdio.h>
int main()
{
    int i, j, rows, columns, a[10][10], Sum = 0;
    printf("\n Please Enter Number of rows and columns  :  ");
    scanf("%d %d", &i, &j);
    printf("\n Please Enter the Matrix Elements \n");
    for(rows = 0; rows < i; rows++)
    {
        for(columns = 0;columns < j;columns++)
        {
            scanf("%d", &a[rows][columns]);
        }
    }
    for(rows = 0; rows < i; rows++)
    {
        Sum = Sum + a[rows][rows];
    }
    printf("\n The Sum of Diagonal Elements of a Matrix =  %d", Sum );
    return 0;
}
```

Q. Write a program to read the elements of a 3x4 matrix and find the biggest and smallest element of the matrix. [2015 spring]

→

```
#include <stdio.h>
int main()
{
  int  c, d, matrix[4][4], maximum, minimum;

  printf("Enter the elements of the matrix\n");

  for (c = 0; c < 3; c++)
    for(d = 0; d < 4; d++)
      scanf("%d", &matrix[c][d]);

  maximum = matrix[0][0];
  for (c = 0; c < 3; c++)
  {
    for (d = 0; d < 4; d++)
    {
```

```
      if (matrix[c][d] > maximum)
      {
        maximum = matrix[c][d];
      }
    }
  }

  minimum = matrix[0][0];
  for (c = 0; c < 3; c++)
  {
    for (d = 0; d < 4; d++)
    {
      if (matrix[c][d] < minimum)
      {
        minimum = matrix[c][d];
      }
    }
  }

  printf("Maximum element in the matrix is %d\n", maximum);
  printf("Minimum element in the matrix is %d\n", minimum);

  return 0;
}
```

Q. How can you initialize one dimensional array at compile time and at run time? Explain with suitable examples. [2016 spring]

→

Compile time initialization: We can initialize the elements of arrays in the same way as the ordinary varaibles when they are declared. The feneral form of initialization of array is:

data_type array_name [array_size] = {list of values} (values in the lists are separated by commas)

example:

```
int respect[3] = {0,1,2};
```

This will declare the variable “number” as an array of size 3 and will assign the values to each element. If the number of values in the list is less than the number of elements, then only that many elements will be initialized. The remaining elements will be set to zero automatically.

Run time initialization: Array can alson be explicitly initialized at run time. This approach is usually applied for initializing large arrays.

For example:

```
int  i, respect[5];
for (i=0; i<5; i++)
{
      printf("\n Enter a number: ");
      scanf("%d", &respect[i]);
}
```

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

→

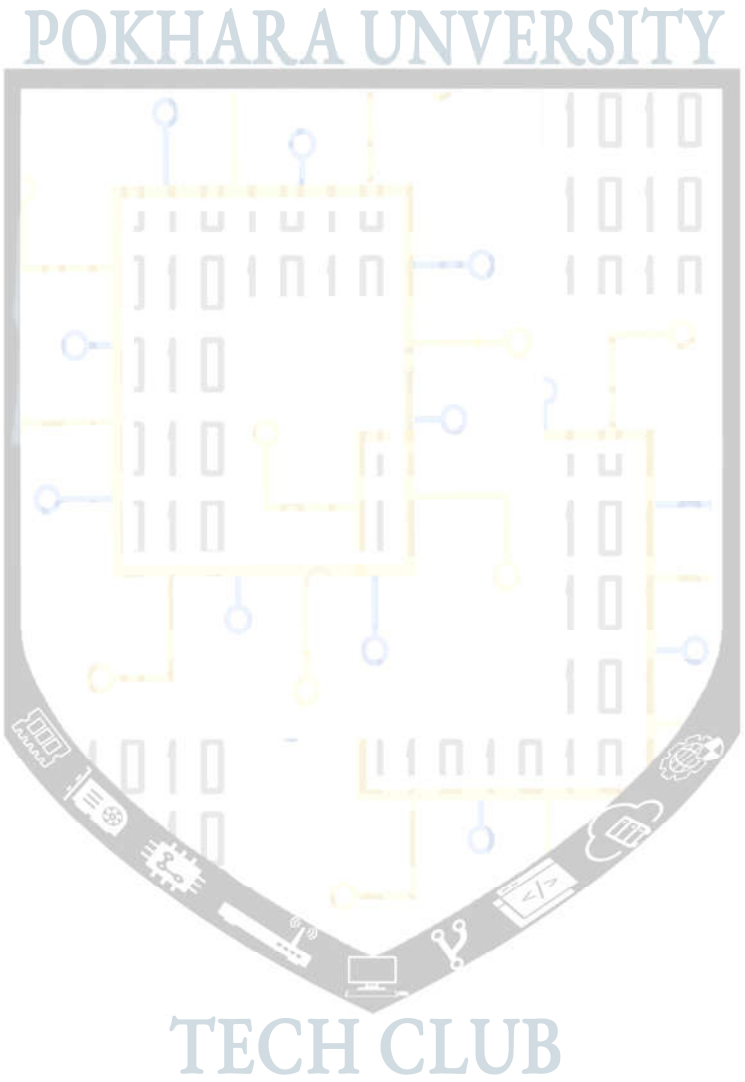
```
#include <stdio.h>
int main()
{
    int i, n;
    int num;
    int count;
    int arr[100];

    printf("Enter the number of elements in array: ");
    scanf("%d", &n);

    for (i = 0; i < n; ++i) {
        printf("Enter number%d: ", i + 1);
        scanf("%d", &arr[i]);
    }

    for (i = 0; i < n; ++i)
    {
         count=0;
         num = arr[i];
         while(arr[i] != 0)
         {
            count++;
            arr[i] /= 10;
         }
         if (count==5)
         {
             printf("%d \n", num);
         }
    }
    return 0;
}
```


Note: The questions of this chapter were solved by **Kusal Lamsal** [B.E. Software], Secretary, Tech Club.


POKHARA UNVERSITY
TECH CLUB

CHAPTER 5: FUNCTIONS

Q.1 Write a recursive program to generate the Fibonacci series up to N terms. Fall 2019

➔

```
#include<stdio.h>
int fibo(int);
int main()
{
    int n,i;
    printf("Enter no.of terms: ");
    scanf("%d",&n);
    for(i = 0;i<n; i++)
    {
        printf("%d\t", fibo(i));
    }
    return 0;
}
int fibo(int num)
{
    if(num == 0 || num == 1)       //base condition
    {
        return num;
    }
    else
    {
        return fibo(num-1) + fibo(num-2);  // recursive call
    }
}
```

Q. What are pre-processor directives?

Pre-processor directive is a collection of special statements that are executed at the beginning of a compilation process. They are placed in the source program before the main function and before the source code is passed through compiler, it is examined by the pre-processor for any processor directives. If there are any, appropriate actions are taken and then, the source program is handed over to the compiler. Pre-processor directives follow special syntax rule that are different from normal C syntax. They all begin with '#' and don't require a semi-colon at the end.

Example:

```
#include<stdio.h>
#define PI 3.1416
#define TRUE 1
#define FALSE 0
```

Other pre-processor directives are: #if, #elif, #else, #endif, etc.

Q.2 What are function call by value and call by reference? Explain with suitable examples. Fall 2019

➔ When values of actual arguments are passed to the functions as arguments, it is known as function call by value. In this case, changes made to the formal argument inside the function have no effect on the actual argument.

Example:

```
#include <stdio.h>
  void swap(int x, int y);
```

```
int main()
{
    int a = 10, b = 20;
    swap(a, b);       // Pass by Values
   printf("a=%d b=%d\n", a, b);
   return 0;
}

void swap(int x, int y)      // Swap functions that swaps
{
    int t;
    t = x;
    x = y;
    y = t;
    printf("x=%d y=%d\n", x, y);
}
```

Output:
x=20 y=10
a=10 b=20

A function call by reference is such a method where the address of variable or argument is passed to the function using pointers. Pointers is a variable that stores a memory address of a variable. In this case, changes made in formal argument inside the function affect the actual argument.

Example:

```
#include <stdio.h>
void swap(int*, int*);
int main()
{
    int a = 10, b = 20;
    swap(&a, &b);   // Pass reference
    printf("a=%d b=%d\n", a, b);
    return 0;
}

void swap(int* x, int* y)
{
    int t;

    t = *x;
    *x = *y;
    *y = t;

    printf("x=%d y=%d\n", *x, *y);
}
```

Output:
x=20 y=10
a=20 b=10

Q.3 List the major advantages of recursive function. Write a recursive program to generate the 10 terms Fibonacci sequence starting from 2. Spring 2018 Fall 2016

➔ When a function calls itself from its body is called Recursion.

Advantages:

Reduce unnecessary calling of function.

Through Recursion one can Solve problems in easy way while its iterative solution is very big and complex.

```
#include <stdio.h>
//function to print fibonacii series
void fibo(int a,int b, int n)
{
    int sum;
    if(n>0)
    {
        sum=a+b;
        printf("%d\t",sum);
        a=b;
        b=sum;
        fibo(a,b,n-1);
    }
}
int main()
{
    int a,b,sum,n;
    int i;
    a=2;        //first term
    b=3;        //second term
    printf("Enter total number of terms: ");
    scanf("%d",&n);
    printf("Fibonacii series is :\n");
    //print a and b as first and second terms of series
    printf("%d\t%d\t",a,b);
    //call function with (n-2) terms
    fibo(a,b,n-2);
    printf("\n");
    return 0;
}
```

Q.4 What do you mean by function? Differentiate between function call by value and call by reference with suitable program. Fall 2018 ,Fall 2013

→ A function is a self-contained block of statements that carries out some specific, well-defined tasks. This is a logical unit composed of a number of statements grouped in a single unit.

Call by value	Call by reference
1.The function is called by directly passing value of variable as argument.	1.The function is called by directly passing address of variable as argument.
2.We need to declare a general variable as function argument.	2.We need to declare a pointer variable as argument.
3. Calling function by value does not changes actual values of variables.	3. Calling function by reference changes actual values of variables.
4.It is a slow way of calling function as we are calling it by passing value.	4.It is a fast way of calling function as we are calling it by passing address of a variable.
5. Example:	5. Example:

```c
#include <stdio.h>
  void swap(int x, int y);

int main()
{
    int a = 10, b = 20;
    swap(a, b);       // Pass by
Values
  printf("a=%d b=%d\n", a, b);
  return 0;
}
void swap(int x, int y)      //
Swap functions that swaps
{
    int t;
    t = x;
    x = y;
    y = t;
    printf("x=%d y=%d\n", x, y);
}
```

```c
#include <stdio.h>

void swap(int*, int*);

int main()
{
    int a = 10, b = 20;
    swap(&a, &b);   // Pass
reference
    printf("a=%d b=%d\n", a, b);
    return 0;
}
void swap(int* x, int* y)
{
    int t;
    t = *x;
    *x = *y;
    *y = t;
    printf("x=%d y=%d\n", *x, *y);
}
```

Q.5 What is a function prototype? Find the sum of first n natural number using recursive function. Fall 2018

→ A function prototype also known as function declaration tells the compiler about the function name, return type and how to call the function.

```c
#include <stdio.h>
int addNumbers(int n);
int main() {
    int num;
    printf("Enter a positive integer: ");
    scanf("%d", &num);
    printf("Sum = %d", addNumbers(num));
    return 0;
}

int addNumbers(int n) {
    if (n != 0)
        return n + addNumbers(n - 1);
    else
        return n;
}
```

Output

```
Enter a positive integer: 20
Sum = 210
```

Q.6 Does a function return multiple values? When and how a function will return single or multiple values, illustrate with suitable examples. Fall 2017 Fall 2013

→ We cannot return multiple values from a function directly but we can return more than one values from a function by using the method called “call by reference”. In the invoker function, we will

use two variables to store the results, and the function will take pointer type data. So we have to pass the address of the data.

For example;

```
#include<stdio.h>
void div(int a, int b, int *quotient, int *remainder)
{
   *quotient = a / b;
   *remainder = a % b;
}
main()
 {
   int a = 76, b = 10;
   int q, r;
   div(a, b, &q, &r);
   printf("Quotient is: %d\nRemainder is: %d\n", q, r);
}
```

Output:

```
Quotient is: 7
Remainder is: 6
```

Q.7 What are pre-processor directives? Differentiate between macro and function with describing necessary example. Fall 2018

→ Pre-processor directives are lines included in a program that begin with the character #, which make them different from a typical source code text.

Macro	Function
1. Macro is Preprocessed.	1. Function is Compiled.
2. No Type Checking is done in Macro.	2. Type Checking is Done in Function.
3. Using Macro increases the code length.	3. Using Function keeps the code length unaffected.
4. Use of macro can lead to side effect at later stages.	4. Functions do not lead to any side effect in any case.
5. Speed of Execution using Macro is Faster.	5. Speed of Execution using Function is Slower.
6. Before Compilation, macro name is replaced by macro value.	6. During function call, transfer of control takes place.

Example:	Example:

`#include<stdio.h>`
`#define NUMBER 10 // Macro`
`int main()`
`{`
`printf("%d", NUMBER);`
`return 0;`
	`#include<stdio.h>`
	`int number()`
	`{`
	`return 10;`
	`}`
	`int main()`

}
 	
	{
	    printf("%d", number());
	    return 0;
	}

Q.8 How arguments are passed by using call by value and call by reference? Explain with examples. Fall 2017, Spring 2017

In the function call by value, we need to declare a general variable as function argument to be passed

➔ Example:

```
#include <stdio.h>
void swap(int x, int y);
int main()
{
    int a = 10, b = 20;
    swap(a, b);       // Pass by Values
  printf("a=%d b=%d\n", a, b);
  return 0;
}

void swap(int x, int y)     // Swap functions that swaps
{
    int t;

    t = x;
    x = y;
    y = t;

    printf("x=%d y=%d\n", x, y);
}
```

But in call by reference, We need to declare a pointer variable as argument to be passed.

Example:

```
#include <stdio.h>
  void swap(int*, int*);
  int main()
{
    int a = 10, b = 20;
     swap(&a, &b);   // Pass reference
    printf("a=%d b=%d\n", a, b);
    return 0;
}
void swap(int* x, int* y)
{
    int t;
    t = *x;
    *x = *y;
    *y = t;
    printf("x=%d y=%d\n", *x, *y);
}
```

Q.9 Define recursion. Write a program to find sum of n natural number using recursion. Spring 2017

→ Function is called recursive if it is called within itself repeatedly until some specified condition is satisfied.

And rest are same as Q.5

Q.10 Why are functions used? Explain function call by value and call by reference with examples. Spring 2016, Spring 2015

→ Every C program has at least one function, which is main() and they are used for divide a large code into module, due to this we can easily debug and maintain the code.

And rest are same as Q.8

Q.11 Define function, function definition, function calling, function declaration with code examples. Spring 2016

→ A function is a self-contained block of statements that carries out some specific, well-defined tasks. This is a logical unit composed of a number of statements grouped in a single unit.

A function definition provides the actual body of the function.

A function can be accessed by specifying its name following a list of arguments separated by commas, enclosed in a pair of parentheses.

A function declaration tells the compiler about the function name, return type and how to call the function.

Example:

```
#include<stdio.h>
#include<conio.h>
int sum(int,int);  //function declaration
main( )
{
int a,b,s;
printf("Enter two numbers:");
scanf("%d%d",&a,&b);                          calling function
s=sum(a,b);
printf("sum of given number=%d",s);
getch();
}
int sum(int x,int y)   //function declarator
{
     int z;                                    called function
     z=x+y;          function body
     return z;
}
```

Q.12 Define function prototype? Write a program to read an integer number and find the sum of its digit using recursive function. Fall 2015

→ A function prototype also known as function declaration tells the compiler about the function name, return type and how to call the function.

And the rest are same as Q.5

OR

What are pre-processor directives? Differentiate between macro and function with describing necessary example.

Ans: Same as Q.7

Q.13 Use recursive function to evaluate:

f(x) = $x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots$ spring 2015

```
#include<stdio.h>
#include<conio.h>
int main()
{
            float series(float,int),x;
            int n;
            printf("\nEnter X:");
            scanf("%f",&x);
            printf("\nEnter n:");
            scanf("%d",&n);
            printf("\nAns %f",series(x,n));
            getch();
            return 0;
}
float series(float x,int n)
{
            long factorial(int);
            float power(float,int);
            float sum=0;
            int i,s=1;
            for(i=1;i<=n;i+=2)
            {
                        sum+=(power(x,i)/factorial(i))*s;
                        s*=-1;
            }
            return sum;
}
float power(float x, int y)
{
            float p=1;
            int i;
            for(i=1;i<=y;i++)p*=x;
            return p;
}
long factorial(int p)
{
            long f=1;
            int i;
            for(i=1;i<=p;i++)f*=i;
            return f;
}
```

Q.14 Define prototype. Which method of function call you should prefer to swap two integer values illustrate with the help of program. Fall 2014

A function prototype also known as function declaration tells the compiler about the function name, return type and how to call the function.

Choose a method of function from Q.2

Q.15 What is recursive function? Write a recursive program to generate the first 15 numbers of Fibonacci sequence. Spring 2014

→ Function is called recursive if it is called within itself repeatedly until some specified condition is satisfied.

```
#include<stdio.h>
int fibonacci(int);

int main(void)
{
      for(int n = 0; n < 15; n++)
    {
        printf("%d ", fibonacci(n));
    }

    return 0;
}


int fibonacci(int num)
{
    if(num == 0 || num == 1)       //base condition

    {
        return num;
    }
else
    {
        return fibonacci(num-1) + fibonacci(num-2);  // recursive call
    }

}

Output:
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

Q.16 What are the advantages of function call-by reference over call by value? How would you pass a variable by reference to a function? Give an example. Spring 2014

→ The advantages of function call by reference over call by value is that the function can change the value of the argument, which is quite useful. It does not create duplicate data for holding only one value which helps you to save memory space. In this method, there is no copy of the argument made.

And the rest are same as Q.2 for function call by reference.

Q.17 What is a recursive function? Write a recursive program to calculate factorial of a given number using recursive function. Spring 2013 Fall 2013

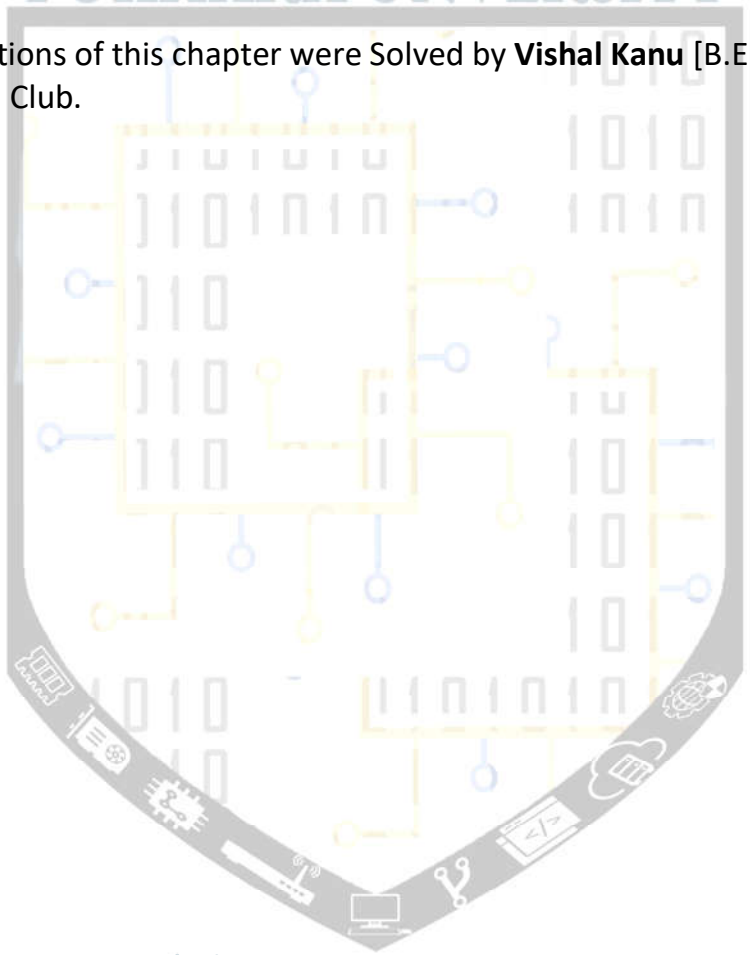
→ Function is called recursive if it is called within itself repeatedly until some specified condition is satisfied.

```
#include<stdio.h>
long int multiplyNumbers(int n);
int main() {
    int n;
```

```
    printf("Enter a positive integer: ");
    scanf("%d",&n);
    printf("Factorial of %d = %ld", n, multiplyNumbers(n));
    return 0;
}

long int multiplyNumbers(int n) {
    if (n>=1)
        return n*multiplyNumbers(n-1);
    else
        return 1;
}
```


Note: The questions of this chapter were Solved by **Vishal Kanu** [B.E. Computer], Treasurer, Tech Club.



CHAPTER 6: POINTERS

Q. What is a pointer variable?

Ans: A pointer is a special type of variable which contains memory address as its value.

Pointer Declaration:

*data_type *ptr_name;*

Here, the asterisk (*) tells that the variable ptr_name is a pointer variable and it points to a variable of type data_type.

Once a pointer has been declared we can use assignment operator to initialize the variable. For example:

```
int quantity;
int *ptr; //declaration
ptr = &quantity; //initialization
```

Or we can combine declaration and initialization as:

```
int *ptr = &quantity;
```

Q. What is dynamic memory allocation? Explain DMA process with example.

→ The process of allocating and deallocating memory in a program during run time is known as dynamic memory allocation (DMA). To manage memory at runtime(DMA process), library functions: malloc(), calloc() ,realloc() and free() are used, which are defined in 'stdlib.h' header file.

1. malloc() function : It allocates requested size of bytes of memory and returns a pointer to the first byte of the allocated memory to the program.

   Syntax: *ptr = (cast-type*) malloc(byte-size);*

   Where, ptr is a pointer of cast-type

   Example: `int *ptr = (int *) malloc(5 * sizeof(int));`

   Here, the sizeof(int) gives 4 bytes. Thus, a total memory of 20 bytes is reserved to ptr and the address of first byte of allocated memory is assigned to ptr.

   However, if space is insufficient then allocation fails and returns a NULL pointer.

2. calloc() function : It allocates requested size of bytes of memory, sets the bytes to zero and returns a pointer to the first byte of the allocated memory of the program. Unlike malloc(), it takes two arguments and it initializes all the bytes in allocated memory block to zero. Memory allocated by calloc() is contiguous and use widely for storing derived data types such as arrays and structure.

   Syntax : *ptr = (cast-type*) calloc(no_of_elements, size_of_each_element);*

   Where, no_of_elements specifies the number of items for which memory is to be allocated and size_of_each_element specifies the memory size for each item.

   Example: `int *ptr = (int *) calloc(5,sizeof(int));`

   The above statement allocates contigious memory space for 5 blocks each of size 4 bytes, initializes them to zero and returns the pointer to the first byte to ptr.

   However, if space is insufficient then allocation fails and returns a NULL pointer.

3. free() function : Dynamically allocated memory is not released on its own. Thus, free() function is used to free/deallocate the memory space allocated by malloc(), calloc() or realloc() function. It helps to reduce memory wastage by freeing it.

   Syntax: *free(ptr);*

Here, ptr is a pointer to a memory block which has already been allocated by malloc(), calloc() or realloc() function.

Note: If the allocated memory is not freed even after its use then, it results in memory leak.

4. realloac() function : It is used to modify the size of previously allocated space. Sometimes, previously allocated memory is insufficient or more than necessary, in such cases realloc() function is used.

Syntax: *realloc(ptr, new_sze);*

Here, ptr is a pointer to previously allocated memory and new_size defines the size of new memory space.

In failure to do so, the function returns NULL.

Q. What is a pointer variable? How can memory of a variable be initialized dynamically? Explain with example. [2019-Fall]

→ A variable can be initialized dynamically using dynamic memory allocation functions malloc(), calloc(), realloc() and free() , which are define in header file 'stdlib.h'.

For their details, please look at fore page.

Q. Void pointer [Short Note-2018 Spring,2016-Fall,2014-Fall]

→ A void pointer is a special type of pointer. It is not associated to any data type and it can be type casted to any data type. Using void pointer, the pointed data cannot be referenced directly (i.e. operator * cannot be used on them directly). The type casting must be used to access to data through void pointer.

An example to illustrate use of void pointer:

```
#include<stdio.h>
int main()
{
    void *vptr; //declaring a void pointer
    int p=5;
    float q=6.6;
    vptr = &p; //casting to int type
    printf("%d\n",* (int *) vptr); //not simply *ptr
    vptr = &q; //casting to float type
    printf("%f\n",* (float *) vptr); //not simply *ptr
    return 0;
}
```


OUTPUT:

```
5
6.600000
```

Q. What is dynamic memory allocation? Explain different functions used in dynamic memory allocation.

→ Already answered.

Q. What is DMA? Write a program to find sum of 5 numbers supplied by users using DMA. [2017-Spring]

→Solution:

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

```
#include<stdlib.h>
int main()
{
    int *array;
    array = (int *) calloc(5,sizeof(int)); //allocating memory dynamically
    int i,sum=0;
    printf("Enter 5 numbers :\n");
    for(i=0;i<5;i++)
    {
        scanf("%d",(array + i));
    }
    for(i=0;i<5;i++)
    {
        sum = sum + *(array + i);
    }
    printf("Sum of input numbers = %d\n",sum);
    free(array); //freeing the memory space occupied by array
    return 0;
}
```

Q. Explain the relationship between arrays and pointers. How can a pointer variable be used to access and modify single-dimensional and multi-dimensional arrays? [2017-Fall]

→

Relationship between Array and Pointer

An array name by itself is a pointer to the first element in the array (i.e it represents the base address of the array). Therefore, if **x** is a onedimensional array, then the address of the first array element can be expressed as either **&x[0]** or simply as **x** (i.e **x** is same as **&x[0]**). Thus, the address of second element of the array can be represented as either **&x[1]** or **(x + 1)**. In general, the address of array element (i + 1) can be expressed as either **&x[i]** or as **(x + i).**

Similarly, for (i+1)th element**, x[i]**and ***(x+i)** both represent the contents of that address, i.e., the value of the (i+1)th element of x.

For 2-D array, in general, **&x[i][j]** is same as ***(x+i)+j** and **x[i][j]** is same as ***(*(x+i) + j).**

Example: A program to initialize 1-D array and display its elements with their memory address using a pointer variable.

```
#include<stdio.h>
int main()
{
    int x[5]={10,20,30,40,50};
    int *ptr= &x,i;
    printf("Array-Element \t Value \t Memory-Address\n");
    for(i=0;i<5;i++)
    {
        printf("x[%d] \t\t %d \t %ld\n",i,*(ptr + i),(ptr+ i));
    }
    return 0;
}
```

OUTPUT:

```
Array-Element    Value   Memory-Address
x[0]             10      6422280
x[1]             20      6422284
x[2]             30      6422288
x[3]             40      6422292
x[4]             50      6422296
```

Example: A program to initialize 2-D array and display its elements with their memory address using a pointer variable.

```
#include<stdio.h>
int main()
{
    int x[2][3]={{1,2,3},{4,5,6}};
    int (*ptr)[3] = &x; //a pointer to 2-d array
    int i,k;
    printf("Array-Element \t Value \t Memory-Address\n");
    for(i=0;i<2;i++)
    {
       for(k=0;k<3;k++)
       {
         printf("x[%d][%d] \t %d \t %ld\n",i,k,*(*(ptr + i)+k),
     *(ptr+i)+k);
       }
    }
    return 0;
}
```

OUTPUT:

```
Array-Element    Value   Memory-Address
x[0][0]          1       6422272
x[0][1]          2       6422276
x[0][2]          3       6422280
x[1][0]          4       6422284
x[1][1]          5       6422288
x[1][2]          6       6422292
```

Q. Write a program to sort the array using dynamic memory allocation. [2016-Fall]

→ Solution:

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int n,i,j,temp;
    printf("How many integer numbers to input ?\n");
    scanf("%d",&n);
    int *array = (int *)calloc(n,sizeof(int));
    printf("Enter array elements:\n");
    for(i=0;i<n;i++)
    {
        scanf("%d",(array + i)); //(array + i) is same as &array[i]
    }
     for(i=0;i<(n-1);i++)
    {
        for(j=i+1;j<n;j++)
        {
           if(*(array + i)>*(array + j)) // *(array+i) is same as array[i]

           {
               temp = *(array + i);
               *(array + i)= *(array + j);
               *(array + j)= temp;
```

```
            }
        }
    }
    printf("After sorting in ascending order, the elements are :\n");
    for(i=0;i<n;i++)
    {
        printf("%d\t",*(array + i));
    }
    free(array); //freeing up the allocated memory
    return 0;
}
```

OUTPUT:

```
How many integer numbers to input ?
5
Enter array elements:
99
2
55
3
111
After sorting in ascending order, the elements are :
2       3       55      99      111
```

Q. How dynamic memory allocation can be achieved? Explain with a suitable example. What are the advantages of dynamic memory allocation? [2016-Spring]

→ For dynamic memory allocation process, look at fore page.

Example to illustrate usage of dynamic memory allocation :

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int n1,n2,i;
    printf("How many integer numbers to input ?\n");
    scanf("%d",&n1);
    int *array = (int *)calloc(n1,sizeof(int)); //malloc() can also beused
    printf("Enter array elements:\n");
    for(i=0;i<n1;i++)
    {
        scanf("%d",(array + i)); //(array + i) is same as &array[i]
    }
    printf("The array elements are :\n");
    for(i=0;i<n1;i++)
    {
        printf("%d\t",*(array + i));
    }
    printf("\nHow many elements to be added to array ?\n");
    scanf("%d",&n2);
    realloc(array, (n1+n2)*sizeof(int)); //using realloc() to increase
//memory
    printf("Enter additional array elements:\n");
    for(i=n1;i<(n1+n2);i++)
    {
        scanf("%d",(array + i)); //(array + i) is same as &array[i]
    }
    printf("The final array elements are :\n");
```

```
    for(i=0;i<(n1+n2);i++)
    {
        printf("%d\t",*(array + i));
    }
    free(array); //freeing up the allocated memory
    return 0;
}
```

OUTPUT:

```
How many integer numbers to input ?
2
Enter array elements:
1
2
The array elements are :
1       2
How many elements to be added to array ?
2
Enter additional array elements:
3
4
The final array elements are :
1       2       3       4
```

Advantages of dynamic memory allocation:

- Dynamic Allocation is done at run time.
- Data structures can grow and shrink to fit changing data requirements.
- We can allocate (create) additional storage whenever we need them.
- We can de-allocate (free/delete) dynamic space whenever we are done with them.

Q. The Pointer arithmetic. [2016-Spring]

→ A lmited set of arithmetic operations can be performed on pointers. A pointer may be: incremented (++), decremented (--) , an integer may be added to a pointer (+ or +=), an integer may be subtracted from a pointer (– or -=). The main application of pointer arithmetic in C is in **arrays**

However, it should be known that the arithmetic done has nothing do with the value contended by the address assigned to the pointer. Consider a pointer variable 'ptr' which is a integer type pointer variable.

- An integer can be added to or subtracted from a pointer. For example:
  `ptr + 2` : It specifies an address in memory which is two memory blocks for integer (i.e 8 memory bytes) beyond the address pointed by **ptr**.
  `ptr - 2` : It specifies an address in memory which is two memory blocks for integer (i.e 8 memory bytes) ahead of the address pointed by **ptr**.
- A pointer variable can be incremented or decremented.
  `ptr++` : It specifies an address in memory which is one memory block for integer (i.e 4 memory bytes) beyond the address pointed by **ptr**.
  `ptr--` : It specifies an address in memory which is one memory block for integer (i.e 4 memory bytes) ahead of the address pointed by **ptr**.

An example to illustrate this :

```
#include<stdio.h>
int main()
{
    int val[3]={5,10,15};
    int *ptr =val; //Assigning address of val[0] to ptr
    printf("Adress of val[0] : %ld\n",ptr);
```

```
    ptr++; //ptr++ = ptr +1 , now ptr points to val[1]
    printf("Adress of val[1] : %ld\n",ptr);
    ptr = ptr +1; // now ptr points to val[2]
    printf("Adress of val[2] : %ld\n",ptr);
    ptr--; //ptr-- = ptr -1 , now ptr points to val[1]
    printf("Adress of val[1] : %ld\n",ptr);
    ptr = ptr -1;; // now ptr points to val[0]
    printf("Adress of val[0] : %ld\n",ptr);
    return 0;
}
```

OUTPUT:

```
Adress of val[0] : 6422288
Adress of val[1] : 6422292
Adress of val[2] : 6422296
Adress of val[1] : 6422292
Adress of val[0] : 6422288
```

Q. What do you mean by dynamic memory allocation? Explain about memory leak. [2015-Spring]

→ Memory leak is related dynamic memory allocation Memory leak occurs when programmers create a memory in heap and forget to delete it. Memory leaks are particularly serious issues as the allocated memory is not freed and other applications may not work and eventually system may slow down due to shortage of memory.

When you allocate memory dynamically it's your responsibility to *free* the allocated memory, i.e. give it back to the OS, so it can be reused, if you fail to free memory and it turns out that it is enough memory, your system could run out of memory, causing all running programs to fail and new programs would not be able to start.

Memory leak can be avoided by deleting the dynamically allocated memory space after its use.

Example to illustrate memory leak:

```
void my_func() {

   int *data = new int;

   *data = 50;

}
```

Here the problem is *data pointer is never deleted, so memory is wasted.

Q. What is a pointer? Explain memory allocation in C programming. Why dynamic memory allocation is better? [2015-Fall]

→ Already answered.

Q. Write a program using pointers to read in an array of integers. Next add the elements in the array and display the sum on the screen. [2015-Fall]

→ Solution:

```
#include<stdio.h>
int main()
{
    int n;
    printf("Enter number of elements :\n");
    scanf("%d",&n);
    int array[n];
    int *ptr = array; //assigning address of array[0] to ptr
```

```
    int i,sum=0;
    printf("Enter the array elements :\n");
    for(i=0;i<n;i++)
    {
        scanf("%d",(ptr + i));
    }
    for(i=0;i<n;i++)
    {
        sum = sum + *(ptr + i);
    }
    printf("Sum of all elements of array is : %d\n",sum);
    return 0;
}
```

Q. Find the output. [2015-Fall]

```
void fun(int *p);

void main()
{
    int x=4;
    printf("%d\n",x);
    fun(&x);
    printf("%d\n",x);
}

 fun(int *p)
{
    *p = *p/2 + 13;
}
```

→

OUTPUT:

4

15

Q. What is pointer? How does a pointer differ from an array? Explain dynamic memory allocation. [2014-Spring]

→

Pointer	Array
Assigning any other values rather than memory address to a pointer variable causes problem in execution.	Assigning any address to an array variable is not allowed.
sizeof(pointer) only returns the amount of memory used by the pointer variable itself	sizeof(array) returns the amount of memory used by all elements in array
Arithmetic on pointer variable is allowed.	Arithmetic on array is not allowed.
&pointer returns the address of pointer	&array is an alias for &array[0] and returns the address of the first element in array

Q. What is pointer variable? How memory of a variable can be initialized dynamically? Explain with example. [2014-Fall]

→ Same question has been answered already. Look at fore page.

Q. Write a program to sort(ascending order) n integer values in an array using pointer. [2014-Fall]

→ Solution:

```
#include<stdio.h>
int main()
{
    int n,i,j,temp;
    printf("Enter number of array elements:\n");
    scanf("%d",&n);
    int array[n];
    int *ptr = &array; //&array is same as &array[0]
    printf("Enter the array elements:\n");
    for(i=0;i<n;i++)
    {
        scanf("%d",(ptr + i));
    }
    for(i=0;i<(n-1);i++)
    {
         for(j=i+1;j<n;j++)
        {
            if(*(ptr + i)>*(ptr + j))
            {
            temp = *(ptr +i);
            *(ptr + i) = *(ptr +j);
            *(ptr +j) = temp;
            }
        }
    }
     printf("The array elements after sorting in ascending order:\n");
    for(i=0;i<n;i++)
    {
        printf("%d\t",*(ptr + i));
    }
    return 0;
}
```

OUTPUT:

```
Enter number of array elements:
3
Enter the array elements:
9
2
7
The array elements after sorting in ascending order:
2       7       9
```

Note: Same program has been done with dynamic memory allocation approach at fore page.

Q. Pointer to Array. [Short Note:2014-Fall]

→ An array address can be assigned to pointer. If x be a single dimensional array such as:

```
int x[5];
int *ptr = &x; //&x is same as &x[0]
```

Here ptr points to base memory address of x.

If x be a two-dimensional matrix such as then:

```
int x[2]3];
int (*ptr)[3] = &x;
```

Here, ptr points to a two-dimensional array having 3 as subscript.

Also, an array name by itself is a pointer to the first element in the array (i.e it represents the base address of the array). Therefore, if **x** is a one-dimensional array, then the address of the first array

element can be expressed as either **&x[0]** or simply as **x** (i.e **x** is same as **&x[0]**). Thus, for (i+1)th element**, x[i]**and ***(x+i)** both represent the contents of that address, i.e., the value of the (i+1)th element of x. For 2-D array, in general, **&x[i][j]** is same as ***(x+i)+j** and **x[i][j]** is same as ***(*(x+i) + j).**

Example: A program to initialize 1-D array and 2-d array and display their elements with their memory address using a pointer variable.

```
#include<stdio.h>
int main()
{
    int x[2]={10,20};
    int *ptrX= &x,i,j; //&x is same as &x[0]
    int y[2][2]={{0,1},{2,3}};
    int (*ptrY)[2] = &y; //&y is same as &y[0][0]
    printf("Array-Element \t Value \t Memory-Address\n");
    for(i=0;i<2;i++) //for single-dimensional array
    {
        printf("x[%d] \t\t %d \t %ld\n",i,*(ptrX + i),(ptrX+ i));
    }
    printf("---------------------------------\n");
    for(i=0;i<2;i++) //for two-dimensional array
    {
     for(j=0;j<2;j++)
     {
       printf("y[%d][%d] \t %d \t %ld\n",i,j,*(*(ptrY + i)+j),
*(ptrY+ i)+j);
     }
    }
    return 0;
}
```

OUTPUT:

```
Array-Element    Value   Memory-Address
x[0]             10      6422280
x[1]             20      6422284
---------------------------------
y[0][0]          0       6422264
y[0][1]          1       6422268
y[1][0]          2       6422272
y[1][1]          3       6422276
```

Q. Write a program using pointers to read in an array of integers and print its element in reverse order. [2013-spring].

→ Solution:

```
#include<stdio.h>
int main()
{
    int array[5],i;
    int *ptr = &array; //&array is same as &array[0]
    printf("Enter 5 elements of array:\n");
    for(i=0;i<5;i++)
    {
        scanf("%d",(ptr + i));
    }
    printf("The array elements in reverse order are:\n");
    for(i=4;i>=0;i--)
```

```
    {

        printf("%d\t",*(ptr + i));
    }
    return 0;
}
```

Q. Below are two different definitions of the function search(). [2013-spring].

i.
```
void search(int *m[], int x)
{
}
```
ii.
```
void search(int *m, int x)
{
}
```

Are they equivalent? Explain.

→

No, the definitions are not equivalent. In the first definition, there is 'int *m[]' which means that m is an array of integer pointers. However, in the second definition, there is 'int *m' which means that m is a pointer to integer type.

The parameters in the two definitions are different. Due to this fact, the search() function in the first definition accepts an array of integer pointers and a integer value. Whereas, the search() function accepts a pointer to integer-type and an integer value. So, the definitions are not equivalent.

Q. Write the output: [2013-spring].

```
void main()
{
int m[2];
int *p=m;
m[0]=100;
m[1]=200;
printf("%d%d",++*p,*p);
}
```

➔ **OUTPUT:**

101100

Q. What is memory leak? Write a program to print reverse elements of an array using Dynamic Memory Allocation. [2013-Fall].

→ For memory leak, look at fore page.

Solution:

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int n,i,j,temp;
    printf("How many integer numbers to input ?\n");
    scanf("%d",&n);
    int *array = (int *)calloc(n,sizeof(int));
    printf("Enter array elements:\n");
    for(i=0;i<n;i++)
    {
        scanf("%d",(array + i)); //(array + i) is same as &array[i]
```

```
    }
     for(i=0;i<(n-1);i++)
    {
        for(j=i+1;j<n;j++)
        {
           if(*(array + i)>*(array + j)) // *(array+i) is same as array[i]

           {
               temp = *(array + i);
               *(array + i)= *(array + j);
               *(array + j)= temp;
           }
        }
    }
    printf("After sorting in ascending order, the elements are :\n");
    for(i=0;i<n;i++)
    {
        printf("%d\t",*(array + i));
    }
    free(array); //freeing up the allocated memory
    return 0;
}
```

OUTPUT:

```
How many integer numbers to input ?
3
Enter array elements:
1
2
3
The elements in reverse order are :
3       2       1
```

Note: The questions of this chapter were solved by **Roshan Lamichhane** [B.E Computer], President, Tech Club.

CHAPTER 7 : STRUCTURES AND UNIONS

Q. 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 only the information of those employee whose address is "Kathmandu". [2019-Fall]

→ Solution:

```
#include<stdio.h>
#include<string.h>
struct Employee
{
    char name[256],address[256],gender;
    int eid;
};
int main()
{
    struct Employee e[50],temp;
    int i;
    FILE *fptr;
    printf("Enter details of 50 employees :\n");
    for(i=0;i<50;i++)
    {
        printf("Employee %d :\n",i+1);
        printf("Name : ");
        gets(e[i].name);
        printf("eid : ");
        scanf("%d",&e[i].eid);
        printf("Address : ");
        gets(e[i].address);
        printf("Gender :");
        scanf("%c",&e[i].gender);
    }
    fptr=fopen("info.dat","w"); //opening in write mode
    for(i=0;i<50;i++)
    {
      fprintf(fptr,"\t%s %s %c %d\n"
,e[i].name,e[i].address,e[i].gender,e[i].eid);
    }
    fclose(fptr);
    fptr=fopen("info.dat","r"); //opening in read mode
    printf("Employee details who are from Kathmandu :\n");
   char c;
    while(c=(fgetc(fptr))!=EOF) //read until end of file
    {
      fscanf(fptr,"\t%s %s %c
%d",&temp.name,&temp.address,&temp.gender,&temp.eid);
      if(strcmp("Kathmandu",temp.address)==0)
      {
           printf("Name : %s \t Eid : %d \t Gender: %c \n",
           temp.name,temp.eid,temp.gender);
      }
    }
    fclose(fptr); //closing the file
```

```
    return 0;
}
```

Q. What is the significance of file pointer in file handling. Consider a 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. Also display these records of students who are not from Kathmandu.[2018-Spring]

➔ For file pointer, Mahesh sir.

Solution:

```
#include<stdio.h>
#include<string.h>
struct DateofBirth
{
    int mm,dd,yy;
};
struct Student
{
    char name[256],address[256],faculty[256];
    int roll;
    struct DateofBirth dob;
};
int main()
{
    struct Student std[100],temp;
    int i;
    FILE *fptr;
    printf("Enter details of 100 students :\n");
    for(i=0;i<100;i++)
    {
        printf("Student %d :\n",i+1);
        printf("Name : ");
        gets(std[i].name);
        printf("Roll no : ");
        scanf("%d",&std[i].roll);
        printf("Address : ");
        gets(std[i].address);
        printf("Faculty :");
        gets(std[i].faculty);
        printf("Date of birth (mm/dd/yy) :");
        scanf("%d%d%d",&std[i].dob.mm,&std[i].dob.dd,&std[i].dob.yy);
    }
    fptr=fopen("student.txt","w"); //opening in write mode
    for(i=0;i<100;i++)
    {
      fprintf(fptr,"\t%s %d %s %s %d-%d-%d\n"  ,std[i].name,std[i].roll,
      std[i].address,std[i].faculty,std[i].dob.mm,std[i].dob.dd,std[i].dob.
      yy);
    }
    fclose(fptr);
    fptr=fopen("student.txt","r"); //opening in read mode
```

```
    printf("Student records who are not from Kathmandu :\n");
    char c;
    while(c=(fgetc(fptr))!=EOF) //read until end of file
    {
       fscanf(fptr,"\t%s %d %s %s %d-%d-%d\n" ,&temp.name,&temp.roll,
      &temp.address,&temp.faculty,&temp.dob.mm,&temp.dob.dd,&temp.dob.yy);
       if(strcmp("Kathmandu",temp.address)!=0)
       {
         printf("Name : %s \t Roll : %d \t Address: %s \t Faculty: %s \t
      Dateofbirth: %d-%d-%d
      \n",temp.name,temp.roll,temp.address,temp.faculty,
      temp.dob.mm,temp.dob.dd,temp.dob.yy);
       }
    }
    fclose(fptr); //closing the file
    return 0;
  }
```

Q. Write a program to create structure for the following data for cricket game. (Country name, Player name, Playing type(e.g. bating, balling or both), Number of matches played by player and salary). Save the information in a file named "cricket.txt" and display the information of those players who had played more than 10 matches. [2018-Fall]

→ Solution:

```
#include<stdio.h>
struct Cricket
{
    char cName[256],pName[256],pType[100];
    int matches;
    float salary;
};
int main()
{
    int n,i;
    printf("How many player records to enter ?\n");
    scanf("%d",&n);
    struct Cricket player[n],temp;
    FILE *fptr;
    printf("Enter details of %d students :\n",n);
    for(i=0;i<n;i++)
    {
        printf("Player %d :\n",i+1);
        printf("Player name : ");
        gets(player[i].pName);
        printf("Country name : ");
        gets(player[i].cName);
        printf("Playing type : ");
        gets(player[i].pType);
        printf("No.of matches :");
        scanf("%d",&player[i].matches);
        printf("Salary :");
        scanf("%f",&player[i].salary);
    }
    fptr=fopen("cricket.txt","w"); //opening in write mode
    for(i=0;i<n;i++)
```

```
    {
      fprintf(fptr,"\t%s %s %s %d %f\n",player[i].pName,player[i].cName,
      player[i].pType,player[i].matches,player[i].salary);
    }
    fclose(fptr);
    fptr=fopen("cricket.txt","r"); //opening in read mode
    printf("Players who have played more than 10 matches :\n");
    char c;
    while(c=(fgetc(fptr))!=EOF) //read until end of file
    {
       fscanf(fptr,"\t%s %s %s %d %f\n",&temp.pName,&temp.cName,
      &temp.pType,&temp.matches,&temp.salary);
       if(temp.matches>10)
       {
         printf("Name : %s \t Country : %s \t Playing-type: %s \t Matches:
      %d \t Salary: %.2f \n",temp.pName,temp.cName,temp.pType,
      temp.matches,temp.salary);
       }
    }
    fclose(fptr); //closing the file
    return 0;
}
```

Q. Differentiate structure and union. How the members of structure are accessed? [2017-Spring]

→

Structure	Union
The keyword **struct** is used to define a structure.	The keyword **union** is used to define a union.
Each member of a structure is assigned its own unique storage. It takes more memory spaces than that of union	All members of a union share the memory space. It takes less memory spaces than that of structure.
The amount of memory required to store a structure is the sum of the memory spaces required by all its members.	The amount of memory required to store a union is same as memory size occupied by a member which requires the largest memory space among the members.
All the members of structure can be accessed at any point of the time.	Only one member of union can be accessed at any given time.

Structure is declared as
struct book
{
char name[100],author[100];
int bookid;
float price;
};
	Union is declared as
	union book
	{
	char name[100],author[100];
	int bookid;
	float price;
	};

A structure member can be accessed using period or dot (i.e. ".") operator. The synatax for accessing of a structure variable is as follows:

structure_name.member

where **structure_name** refers to the name of the structure-type variable and **member** refers the name of a member within the structure. Notice the period (.) that separates the structure variable name from the member name. For nested structure, the member is accessed as:

structure_name.member_structure.member

An example to illustrate the processing of a structure:

```
#include<stdio.h>
struct book
 {
      char name[256];
      char author[256];
      int bookId;
};
int main()
{
      struct book b = {"C:Questions And Answers","PU Tech Club",101};
      //structure can be initialized in same way as array
      printf("Book Details:\n");
      printf("Name: %s\n",b.name); //accessing a structure member
      printf("Author: %s\n",b.author);
      printf("Book Id: %d\n",b.bookId);
      return  0;
}
```

Q. How do you declare and initialize array of structure variables? How is structure different from Union. Give example codes [2017-Fall]

→

A structure can be declared using keyword **struct.** A structure is declared as:

```
struct structure_name
{
      //body
}
```

A structure variable can be initialized in a similar manner as that of an array. The members to be initialized must appear in order as in the definition of structure within braces and separated by commas. However, C does not allow the initialization of individual structure members within its definition.

Example:

```
struct student
{
      char name[256];
      char address[256];
      int roll;
};

struct student std = {"Ram","Hemja",1}; //intialization of structure
```

It is same as :

```
struct student std;
strcpy(std.name,"Ram");
strcpy(std.address,"Hemja");
std.roll=1;
```

Q. Self-referential structure [Short note:2017-Fall,2015-Spring]

→ A structure which has at least one member of type pointer to the same structure is known as self-referential structure. In other words, structures pointing to the same type of structures are self-referential in nature.

Example:

```
struct node {
      int data1;
      char data2;
      struct node* link;
};

int main()
{
      struct node ob;
      return 0;
}
```


Here, 'link' is a pointer to a structure of type 'node'. Hence, the structure 'node' is a self-referential structure with 'link' as the referencing pointer.

Self-referential structures are very useful in creation of other complex data structures like linked lists, stacks, queues, trees, etc.

Q. 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". [2016-Fall]

→ Similar to 2018-Fall question

Q. Differentiate structure and union. How the members of Nested structure are accessed? Show it with example. [2016-Fall]

→ For difference, please look at fore page.

A structure which is defined as a member to another structure is known as nested structure. A nested structure can be accessed in same manner as any other member of structure.

Q. What do you mean by nested structure? Write a program to explain nested structure. [2015-Fall]

→ One structure may be nested within another structure's definition. In other words, a structure may be defined as a member of another structure. The outer structure is known as nesting structure and the inner structure(member structure) is nested structure. The nested structure can be accessed using dot (.) operator just like any other member of structure.

A nested may be created as follows:

```
struct nesting_structure {
data-type variable;
..
struct nested_structure name;
};
```


Here, **nested_strucutre** is an already existing structure.

A nested structure may be accessed as follows:

nesting_strucutre.nested_structure.member

An example to explain nested structure;

```
#include<stdio.h>
struct date
{
    int mm,dd,yy;
};
```

```
struct student //nesting structure
{
    char name[256];
    int roll;
    struct date dob; //nested structure
};
int main()
{
    struct student st ={
    "Laxman",
    15,
    {11,11,1999}
    }; //initializing structure
    printf("Student details :\n");
    printf("Name : %s\n",st.name);
    printf("Roll : %d\n",st.roll);
    printf("Date of birth : %d-%d-%d\n",st.dob.mm,st.dob.dd,st.dob.yy);
    return 0;
}
```

Q. 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 50 players and print a country-wise list containing names of players with their number of goals scored. [2014-Fall]

➔ Solution:

```
#include<stdio.h>
#include<string.h>
struct football
{
    char pName[256],cName[256];
    int goals;
};
int main()
{
    struct football player[50],temp;
    int arr[50],i,j;
    printf("Enter details of 50 students :\n");
    for(i=0;i<50;i++)
    {
        printf("Player %d :\n",i+1);
        printf("Player name : ");
        gets(player[i].pName);
        printf("Country name : ");
        gets(player[i].cName);
        printf("No.of goals scored :");
        scanf("%d",&player[i].goals);
    }
    printf("List of player according to country:\n");
    for(i=0;i<50;i++)
    {
        if(arr[i]!=0)
```

```
        {
        printf("Country : %s\n",player[i].cName);
       printf("Player name: %s \t Goals scored : %d \n",player[i].pName,
     player[i].goals);
        arr[i]=0; //0 for printed
            for(j=0;j<50;j++)
            {
            if(arr[j]!=0 && strcmp(player[i].cName,player[j].cName)==0)
                {
                printf("Player name: %s \t Goals scored : %d
\n",player[j].pName,  player[j].goals);
                arr[j]=0;
                }
            }
        }
    }
    return 0;
}
```

Extra questions:

Q. How to return multiple values from a function in C ?

➔ By default, a non-void function returns a single value. But programmers often need to return multiple values from a function. Luckily, there are several workarounds in C to return multiple values.

1. Using pointers : We can use pointers in C to return more than one value from the function by passing pointers as function parameters and use them to set multiple values, which will then have visibility in the caller function.

Example:

```
#include<stdio.h>
void setValue(int *x, int *y, char *z)
{
    *x=1, *y=1, *z='A';
}
int main()
{
    int a,b;
    char c;
    setValue(&a,&b,&c);
    printf("a = %d, b = %d, c = %c\n",a,b,c);
    return 0;
}
```

OUTPUT:

a = 1, b = 1, c = A

2. Using structure : We can also use structures in C to return more than one value from the function. We know that a structure is user defined datatype in C that can hold several data types of the same or different kind. The idea is to create a struct containing all required data types as its members and return that struct from our function. Then we can retrieve the desired values from the struct inside our caller function.

Example:

```
#include<stdio.h>
struct items {
int x,y;
char z;
};
struct items setValue() //function to return multiple values
{
    struct items temp = {1,1,'A'};
    return temp; //returning a struture from function
}
int main()
{
    int a,b;
    char c;
    struct items test = setValue();
    a = test.x;
    b = test.y;
    c = test.z;
    printf("a = %d, b = %d, c = %c\n",a,b,c);
    return 0;
}
```

OUTPUT:

a = 1, b = 1, c = A

3. Using array: We have seen how to return values of different data types from the function using pointers and structure. Now if all values are of same datatype, we can encapsulate the values in an array and return that array as shown below:

Example:

```
#include<stdio.h>
void setValue(int arr[])
{
    arr[0]=0;
    arr[1]=1;
    arr[2]=2;
}
int main()
{
    int a,b,c;
    int temp[3];
    setValue(temp);
    a= temp[0];
    b= temp[1];
    c=temp[2];
    printf("a = %d, b = %d, c = %d\n",a,b,c);
    return 0;
}
```

OUTPUT:

a = 0, b = 1, c = 2

Note: The questions of this chapter were solved by **Roshan Lamichhane** [B.E Computer], President, Tech Club.

CHAPTER 8 : FILE HANDLING

Q.1) What are the file opening modes? Write a program to open a new file, read name, address and telephone number of 10 employees from a user and write to a file.[2013Fall]

Ans: The file opening modes specifies the way in which a file should be opened (i.e. for reading, writing or both, appending at the end of the file, overwriting the file, etc. I other words, it specifies the purpose of opening of file. There are mainly six modes.

Mode	Meaning of mode	During Inexistence of file
r	Open for reading.	If the file does not exist, fopen() returns NULL.
w	Open for writing	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
a	Open for append. Data is added to the end of the file.	If the file does not exist, it will be created.
r+	Open for both reading and writing.	New data is written at the beginning overwriting existing data. If the file does not exist, fopen() returns NULL.
w+	Open for both reading and writing.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
a+	Open for both reading and appending.	New data is appended at the end of file. If the file does not exist, it will be created.

```
#include <stdio.h>
int main()
{
   char name[50],address[20];
   int num;
   long  tel;
   printf("Enter number of students: ");
   scanf("%d", &num);
   FILE *fptr;
   fptr = (fopen("D:\\student.txt", "w"));
   if(fptr == NULL)
   {
       printf("Error!");
       exit(1);
   }
   for(int i=0; i <num;i++ )
   {
      printf("For Employee%d  \n",i+1);
      printf("Enter name:");
      scanf("%s",&name);
```

```
      printf("Enter address:");
      scanf("%s",&address);
      printf("Enter telephone:");
      scanf("%l",&tel);
      fprintf(fptr,"\n Name: %s \nAddress:%s\n Telephone:
%l",name,address,tel);
   }
   fclose(fptr);
   return 0;
}
```

Q.2) Why do we need file handling? Describe the different handling modes.[2013Spring]

Ans: There is a time when the output generated by compiling and running the program does not serve the purpose. If we want to check the output of the program several times, it becomes a tedious task to compile and run the same program multiple times. This is where file handling comes into play. Here are some of the following reasons behind the popularity of file handling:

Reusability: It helps in preserving the data or information generated after running the program.

Large storage capacity: Using files, you need not worry about the problem of storing data in bulk.

Saves time: There are certain programs that require a lot of input from the user. You can easily access any part of the code with the help of certain commands.

Portability: You can easily transfer the contents of a file from one computer system to another without having to worry about the loss of data.

Second part[2013Fall] Qno.1

Q.3) What is the significance of file pointer in file handling? 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. Also display these records of students who are not from Pokhara.[2014Fall]

→

File Pointer:

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.

Opening a file:

The fopen() function opens a stream for use and links a file with that stream. A file pointer associated with that file is then returned by the fopen() function. Most often the file is a disk file.

```
FILE *fp;
fp=fopen("try.c","r");
```

fopen() opens the file "try.c" in read mode. When a file is opened in read mode, three important tasks are performed by fopen().

1. A search is carried out on the disk for the file to be opened.

2. If the file is found, it is loaded into memory from the disk. In case the file is too large, then the file is loaded part wise. In case the file is not found, a NULL() is returned by fopen(). “stdio.h” contains a macro defined as NULL, which indicates that the attempt to open the file failed.

3. fopen() then open sets up a character pointer. The character pointer is a part of FILE structure and points to the first character in memory where the file is loaded.

Closing a File: As said earlier, there is usually a limit on the number of files that can be opened at one time, and so it is important to close the file once it has been used. This ensures that various system resources will be free and reduces the risk of overshooting the set limit. The fclose() function closes a stream that was opened by a call to fopen(). It writes any data still remaining in the disk buffer to the file. The prototype for fclose() is

*Fclose (FILE *fp);*

Where fp is the file pointer. The function fclose() returns an integer value 0 for successful closure, any other value indicates an error. The fclose() generally fails when a disk has been prematurely removed from the drive or there is no more space on the disk.

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
      FILE *fptr;
      char Name[20], Address[20],Faculty[20];
      int Month, Day,Year,Roll;
      fptr=fopen("D:\\student.txt","w");
      if(fptr==NULL)
      {
        printf("\n File cannot be created");
        exit(1);
      }
fprintf(fptr,  " Roll number\tName\tAddress\tFaculty\tDate of birth- Month
Day  Year");
      for(int i=0;i<100;i++)
        {
            printf("For Student%d",i);
         printf("\nRoll\n");
        scanf("%d",&Roll);
        printf("Name\n");
        scanf("%s",&Name);
        printf("Address\n");
        scanf("%s",&Address);
        printf("Faculty\n");
        scanf("%s",&Faculty);
        printf("Date of Birth in Month-Day-Year\n");
        scanf("%d%d%d",&Month,&Day,&Year);

        fprintf(fptr,"\n %d\t%s\t%s\t%s\t%d-%d-%d" ,
Roll,Name,Address,Faculty,Month,Day,Year);
         }
      fclose(fptr);
      return(0);
}
```

Q.4) What is the significance of file pointer? Describe the different file opening modes in C.[2014Spring]

Ans: page [2014Fall] Qno.3

Second part[2013Fall] Qno.1

Q.5) Why header files in C is included in program? Give reasons. Also list-out different header files that you know. Illustrate the program showing the use of header file.[2015Fall]

Ans: A header file is a file containing C declarations and macro definitions to be shared between several source files. We request the use of a header file in the program by *i*ncluding it, with the C preprocessing directive '#include'.

Header files serve two purposes.

System header files declare the interfaces to parts of the operating system. We include them in your program to supply the definitions and declarations we need to invoke system calls and libraries.

We own header files contain declarations for interfaces between the source files of our program. Each time we have a group of related declarations and macro definitions all or most of which are needed in several different source files, it is a good idea to create a header file for them.

Including a header file produces the same results as copying the header file into each source file that needs it. Such copying would be time-consuming and error-prone. With a header file, the related declarations appear in only one place. If they need to be changed, they can be changed in one place, and programs that include the header file will automatically use the new version when next recompiled. The header file eliminates the labor of finding and changing all the copies as well as the risk that a failure to find one copy will result in inconsistencies within a program.

C Header File

1. #include<stdio.h> (Standard input-output header)

Used to perform input and output operations in C like scanf() and printf().

2. #include<string.h> (String header)

Perform string manipulation operations like strlen and strcpy.

3. #include<conio.h> (Console input-output header)

Perform console input and console output operations like clrscr() to clear the screen and getch() to get the character from the keyboard.

4. #include<stdlib.h> (Standard library header)

Perform standard utility functions like dynamic memory allocation, using functions such as malloc() and calloc().

5. #include<math.h> (Math header)

Perform mathematical operations like sqrt() and pow(). To obtain the square root and the power of a number respectively.

6. #include<ctype.h>(Character type header)

Perform character type functions like isaplha() and isdigit(). To find whether the given character is an alphabet or a digit respectively.

7. #include<time.h>(Time header)

Perform functions related to date and time like setdate() and getdate(). To modify the system date and get the CPU time respectively.

Example using your own header file

```
// C program to use the above created header file
#include <stdio.h>  //built-in header file
#include "myhead.h" //user defined header file
```

```
int main()
{
    add(4, 6);

    /*This calls add function written in myhead.h
      and therefore no compilation error.*/
    multiply(5, 5);

    // Same for the multiply function in myhead.h
    printf("BYE!See you Soon");
    return 0;
}
```

Output:

Added value:10

Multiplied value:25

BYE!See you Soon

NOTE : The above code compiles successfully and prints the above output only if you have created the header file and saved it in the same folder the above c file is saved.

Q.6) Write a program to read the name, author, and the price of 500 books in a library from the file 'library.dat'. Now print the book name and price of those books whose price is above Rs.300.

Ans:

```
#include<stdlib.h>
#include <stdio.h>
int main()
{
      char BookName[20], Author[20];
      float Price;
      FILE *fptr;
      fptr=fopen("D:\\library.dat","r");
      if(fptr==NULL)
         {
            printf("Error!");
            exit(1);
         }
      for(int i=0;i<2;++i)
         {
        sprintf("Enter Book Name");
         scanf("%s",&BookName);
         printf("Enter Author Name");
         scanf("%s",&Author);
         printf("Enter Book Price");
         scanf("%f",&Price);
         if(Price>300)
         {
            fprintf(fptr,"\n %s \t %f ", BookName,Price);
         }
         }
      fclose(fptr);
      return 0;
}
```

Q.7) What are the different file opening modes in C? 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 on a 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.

Ans: First part solution Qno.1

```
#include <stdio.h>
void main()
{
    FILE *fptr;
    char name[20], address[20],faculty[20];
    int reg , acd_year, n;
    printf("Enter number of students:");
    scanf("%d",&n);

    fptr=fopen("D://STUDENT.DAT","a");

    if (fptr == NULL)
    {
        printf("File does not exists \n");
        return;
    }
    for(int i=0;i<n;i++)
{
        printf("Pokhara University  \n");
        printf("Enter the name \n");
        scanf("%s",&name);
        printf("Enter Address\n");
        scanf("%s",&address);
        printf("Enter the registration number\n");
        scanf("%i", &reg);
        printf("Faculty\n");
        scanf("%s",&faculty);
        printf("Academic Year \n");
        scanf("%i",&acd_year);

        if(acd_year=2016)
            {
                fprintf(fptr,"\nStudents Admitted in 2016 \n
Name\t\tAddress\t\tReg Number\t\tFaculty\n %s\t\t%s\t\t%i\t\t%s\t
",name,address,reg,faculty);

            }
}
    fclose(fptr);
}
```

Q.8) Write short notes on file opening in C.[2015Fall]

Ans: Qno.1

Q.9) What are the different modes of opening a file? Write a program to create a file "hello.txt" , write data into the file and finally read the data from the user.[2017Fall]

Ans: First part …[2013Fall] Qno.1

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

```
int main()
{
int DATA_SIZE=1000;
    char data[DATA_SIZE];
    FILE * fPtr;
    fPtr = fopen("D://hello.txt","w");
    if(fPtr == NULL)
    {
        printf("Unable to create file.\n");
        exit(1);
    }
    printf("Enter contents to store in file : \n");
    fgets(data, DATA_SIZE, stdin);
    fputs(data, fPtr);
    fclose(fPtr);
    printf("File created and saved successfully. \n");
    return 0;
}
```

Q.10) Why is file handling necessary in C programming? Write a program to input a 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 students whose faculty is 'Engineering" and GPA 3.5.[2016Spring]

Ans:[2013Spring] Qno.2

```
#include <stdio.h>
 #include<stdlib.h>
int main()
{
   char name[50],address[20],faculty[20],program[20];
   int num,i;
   float GPA;
   printf("Enter number of students: ");
   scanf("%d", &num);
   FILE *fptr;
   fptr = (fopen("D:\\Result.DAT", "w"));
   if(fptr == NULL)
   {
       printf("Error!");
       exit(1);
   }
   fprintf(fptr,"\nName\t\tAddress\t\tFaculty\t\tProgram\t\tGPA\n");
   for( i=0; i <num;i++ )
   {
      printf("For student %d \n",i+1);
      printf("Enter name:");
      scanf("%s",&name);
      printf("Enter address:");
      scanf("%s",&address);
      printf("Enter faculty:");
      scanf("%s",&faculty);
      printf("Enter program");
      scanf("%s", &program);
      printf("Enter GPA");
```

```
      scanf("%f",&GPA);
      if(faculty=='engineering'||'Engineering' &&GPA==3.5)
      {
          fprintf(fptr,"%s\t\t%s\t\t%s\t\t%s\t\t%f\n",
name,address,faculty,program,GPA);
      }

   }
   fclose(fptr);
   return 0;
}
```


Note: The questions of this chapter were solved by **Maheshwor Acharya** [B.E. Computer], Vice-President, Tech Club.