

SRI GCSR COLLEGE

Department of Computer Science

Problem Solving in 'C'



Name Of the Student : _____

Class : _____ Group : _____

Register/Roll Number : _____

Problem Solving in C

UNIT I General Fundamentals:

Introduction to computers: Block diagram of a computer, characteristics and limitations of computers, applications of computers, types of computers, computer generations.

Introduction to Algorithms and Programming Languages: Algorithm – Key features of Algorithms, Flow Charts, Programming Languages – Generations of Programming Languages – Structured Programming Language- Design and Implementation of Correct, Efficient and Maintainable Programs.

UNIT II Introduction to C:

Introduction – Structure of C Program – Writing the first C Program – File used in C Program – Compiling and Executing C Programs – Using Comments – Keywords – Identifiers – Basic Data Types in C – Variables – Constants – I/O Statements in C- Operators in C- Programming Examples. Decision Control and Looping Statements: Introduction to Decision Control Statements– Conditional Branching Statements – Iterative Statements – Nested Loops – Break and Continue Statement – Goto Statement

UNIT III Arrays:

Introduction – Declaration of Arrays – Accessing elements of the Array – Storing Values in Array– Operations on Arrays – one dimensional, two dimensional and multi dimensional arrays, character handling and strings.

UNIT IV Functions:

Introduction – using functions – Function declaration/ prototype – Function definition – function call – return statement – Passing parameters – Scope of variables – Storage Classes – Recursive functions. Structure, Union, and Enumerated Data Types: Introduction – Nested Structures – Arrays of Structures – Structures and Functions– Union – Arrays of Unions Variables – Unions inside Structures – Enumerated Data Types.

UNIT V Pointers:

Understanding Computer Memory – Introduction to Pointers – declaring Pointer Variables – Pointer Expressions and Pointer Arithmetic – Null Pointers - Passing Arguments to Functions using Pointer – Pointer and Arrays – Memory Allocation in C Programs – Memory Usage – Dynamic Memory Allocation – Drawbacks of Pointers Files: Introduction to Files – Using Files in C – Reading Data from Files – Writing Data to Files – Detecting the End-of-file – Error Handling during File Operations – Accepting Command Line Arguments.

UNIT I

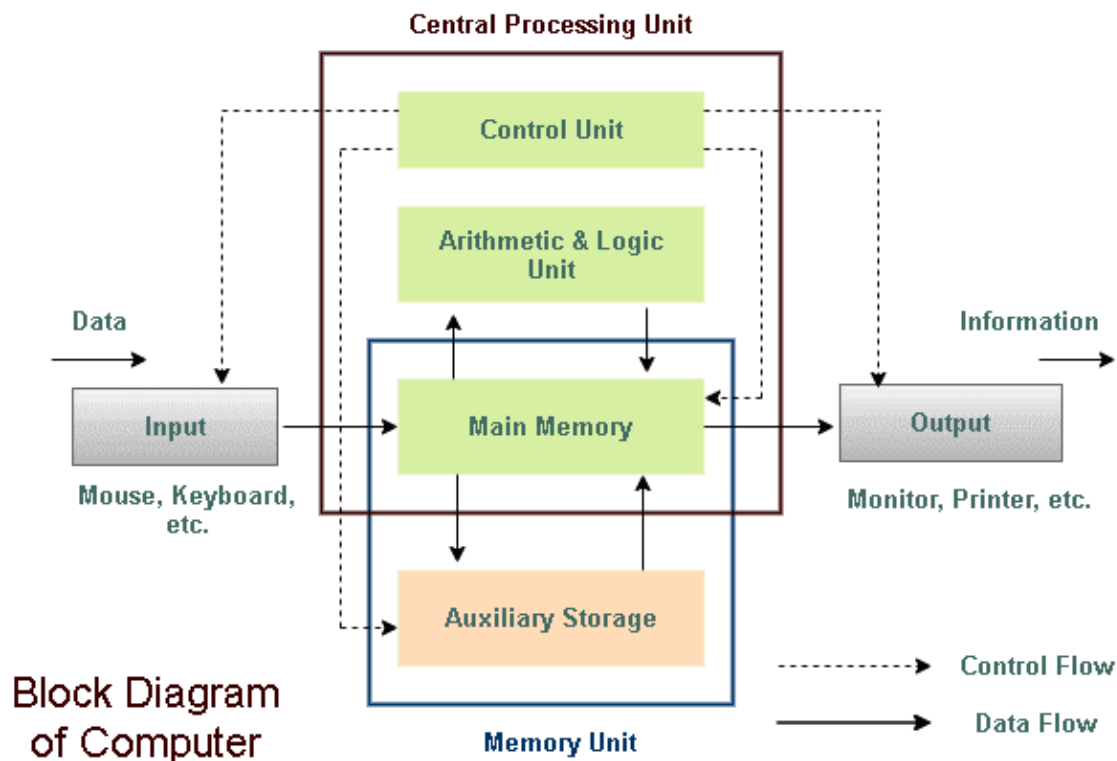
Q: Define computer?

The term computer is derived from the Latin term 'computare', this means to calculate or programmable machine.

Charles Babbage is the "Father" of the computer and invented the first mechanical computer in the early 19th century called Analytical Engine. It uses read-only memory and input data to be provided to the machine via punched cards.

A programmable machine that performs high-speed processing of numbers, as well as of text, graphics, symbols, and sound. All computers contain a central processing unit that interprets and executes instructions; input devices, such as a keyboard and a mouse, through which data and commands enter the computer; memory that enables the computer to store programs and data; and output devices, such as printers and display screens, that show the results after the computer has processed data.

Q: Explain Block diagram of computer?



The basic components of a modern digital computer are: Input Device, Output Device, Central Processor Unit (CPU), mass storage device and memory.

Input:

All the data received by the computer goes through the input unit. The input unit comprises different devices. Like a mouse, keyboard, scanner etc.

In other words, each of these devices acts as a mediator between the users and the computer.

The 3 major functions of the input unit are-

- ✓ Take the data to be processed by the user.
- ✓ Convert the given data into machine-readable form.
- ✓ And then, transmit the converted data into the main memory of the computer.

The sole purpose is to connect the user and the computer. In addition, this creates easy communication between them.

CPU – Central Processing Unit:

Central Processing Unit or the CPU, is the brain of the computer. It works the same way a human brain works. As the brain controls all human activities, the CPU too controls all tasks.

Moreover, the CPU conducts all the arithmetical and logical operations in the computer. Now the CPU comprises of two units, namely – ALU (Arithmetic Logic Unit) and CU (Control Unit).

ALU – Arithmetic Logic Unit:

The Arithmetic Logic Unit is made of two terms, arithmetic and logic. There are two major functions that this unit performs.

1. Data inserted through the input unit into the primary memory. Performs the basic arithmetical operation on it. Like addition, subtraction, multiplication, and division. It performs all sorts of calculations required on the data. Then sends back data to the storage.
2. The unit is also responsible for performing logical operations like, AND, OR, Equal to, Less than, etc. In addition to this it conducts merging, sorting, and selection of the given data.

CU – Control Unit:

The control unit as the name suggests is the controller of all the activities/tasks and operations. All this is performed inside the computer.

The memory unit sends a set of instructions to the control unit. Then the control unit in turn converts those instructions. After that these instructions are converted to control signals.

These control signals help in prioritizing and scheduling the activities. Thus, the control unit coordinates the tasks inside the computer in sync with the input and output units.

Memory Unit:

All the data that has to be processed or has been processed is stored in the memory unit. The memory unit acts as a hub of all the data.

There are two types of computer memory-

1. **Primary memory** – This type of memory used to store recent data. The data stored in this is temporary. It can get erased once the power is switched off. Therefore, is also called temporary memory or the main memory.

RAM stands for Random Access Memory. It is an example of primary memory. This memory is directly accessible by the CPU. It is used for reading and writing purposes. For data to be processed, it has to be first transferred to the RAM and then to the CPU.

2. **Secondary memory** – The primary memory stores temporary data. Thus it cannot be accessed in the future. For permanent storage purposes, secondary memory is used. It is also called the permanent memory or the auxiliary memory. The hard disk is an example of secondary memory. Even in a power failure data does not get erased easily.

Output:

All the information sent to the computer once processed is received by the user through the output unit. Devices like printers, monitors, projector, etc. all come under the output unit.

The output unit displays the data either in the form of a soft copy or hard copy. The printer is for the hard copy. The monitor is for softcopy.

The 2 major functions of the input unit are-

- ✓ The output unit accepts the data in binary form from the computer.
- ✓ It then converts it into a readable form for the user.

Functionalities of a Computer:

Digital computer carries out the following five functions:

Step 1 – Takes data as input.

Step 2 – Stores the data/instructions in its memory and uses them as required.

Step 3 – Processes the data and converts it into useful information.

Step 4 – Generates the output.

Step 5 – Controls all the above four steps.

Q: What is mean by Data and Information?

In general, **data** is any set of characters that is gathered and translated for some purpose, usually analysis.

There are multiple types of data. Some of the more common types of data include the following:

- Single character
- Boolean (true or false)
- Text (string)
- Number (integer or floating-point)
- Picture
- Sound
- Video
-

Information is data that has been processed in such a way as to be meaningful to the person who receives it. It is anything that is communicated.

The following characteristics:

- **Timely** – Information should be available when required.
- **Accuracy** – Information should be accurate.
- **Completeness** – Information should be complete.

Q: what are the different characteristics and limitations of computer?

Basic characteristics (Advantages) about computer are:

A. Speed of Computer

Computers are much faster to perform mathematical calculations than human. It is capable of performing millions of tasks per second. The computer is so fast that it can do work in the blink of an eye.

B. Accuracy of Computer

A computer is very accurate. A computer never gives us wrong results under any circumstances. You can absolutely trust the output of the computer. Sometimes we get some error but these are because of the mistake performed by us.

C. Diligence of Computer

Diligence means that the computer is capable of doing any work for a long time without getting tired and getting stops. Even after the computer has worked for such a long time, there is no decrease in its ability to work and the accuracy of the result.

D. Reliability of Computer

Today every major industry and companies in the world have full confidence in their computers, and their entire business is running from computers. These companies store all their data in the computer.

E. Versatility of Computer

Versatility means that the computer is capable of doing any kind of work. Today computers are being used almost everywhere like schools, colleges, hospitals, offices, railway stations, hotels etc.

F. Storage Capacity of Computer

Computer storage is also called permanent storage because once in this store any data means - file, folder, text data, video, image is stored, then that data is stored for life, and when As long as you do not delete any data

G. Automatic

A computer is an automatic machine because once started on a job they carry on until the job is finished without any human assistance.

Let us understand this with an example, suppose you have to copy 100 or 200 files of your computer in Pen drive. The computer starts copying all your 100 or 200 files to the Pen drive, then you do not need to give instructions to copy every file again and again.

This whole process of a computer is called automation.

H. Multitasking

Multitasking is also a very special feature of computers. A user can do different types of tasks on the computer at the same time.

Like we are using MS Word in computer as well as listening to songs and also getting printouts.

I. Power of Remembering

You can store many types of information and data on your computer in very large quantities. Whenever you need this data in future, you can get that data in a few seconds matter why.

Limitations of computer:

A. No Feeling

In computers, like humans, there is no feeling and emotion, nor does the computer have any knowledge and experience, because a computer is a machine which works continuously on the instruction of humans without any selfishness and without tiredness.

B. No IQ

A computer is a dumb machine, without a user, a computer is a useless machine and device.

Until a user does not give any instruction, it cannot do any work and only after completing the instruction, he completes that work very fast.

Q: what are the different Types of Computer?

We can categorize computer in two ways: on the basis of data handling capabilities and size.

On the basis of data handling capabilities, the computer is of *three* types:

1) Analogue Computer:

Analogue computers are designed to *process analogue data*. Analogue data is continuous data that changes continuously and cannot have discrete values. Such as speed, temperature, pressure and current.

Speedometer and mercury thermometer are examples of analogue computers.

Advantages of using analogue computers:

- It allows real-time operations and computation at the same time and continuous representation of all data within the range of the analogue machine.

2) Digital Computer:

Digital computer is designed to perform calculations and logical operations at high speed. It accepts the raw data as input in the form of digits or binary numbers (0 and 1) and processes it with programs stored in its memory to produce the output. All modern computers like laptops, desktops including smart phones that we use at home or office are digital computers.

3) Hybrid Computer:

Hybrid computer has features of both analogue and digital computer. It is *fast like an analogue* computer and has memory and *accuracy like digital computers*. It can process both continuous and discrete data. It accepts analogue signals and convert them into digital form before processing.

For example, a processor is used in petrol pumps that converts the measurements of fuel flow into quantity and price. Similarly, they are used in airplanes, hospitals, and scientific applications.

On the basis of size, the computer can be of *five* types:

1) Supercomputer:

Supercomputers are the **biggest and fastest computers**. They are designed to process huge amount of data. A supercomputer can **process trillions of instructions in a second**. It has thousands of interconnected processors.

Supercomputers are particularly used *in scientific and engineering applications* such as weather forecasting, scientific simulations and nuclear energy research. The first supercomputer was developed by **Roger Cray in 1976**.



Characteristics or applications of supercomputers:

- It has the ability to decrypt your password to enhance protection for security reasons.
- It produces excellent results in animations.
- It helps in designing the flight simulators for pilots at the beginner level for their training.
- It helps in the diagnosis of various critical diseases and in producing accurate results in brain injuries, strokes, etc.
- It helps in scientific research areas by accurately analyzing data obtained from exploring the solar system, satellites, and movement of Earth.
- It also used in a smog control system where it predicts the level of fog and other pollutants in the atmosphere.

2) Mainframe computer:

Mainframe computers are designed to **support hundreds or thousands of users simultaneously**. They *can support multiple programs* at the same time. It means they can execute different processes simultaneously. These features of mainframe computers make them ideal for big organizations like banking and telecom sectors, which need to manage and process a high volume of data that requires integer operations such as indexing, comparisons, etc.



Characteristics of Mainframe Computers:

- It can process huge amount of data, e.g. millions of transactions in a second in the banking sector.
- It has a very long life. It can run smoothly for up to 50 years after proper installation.
- There are fewer chances of error or bugs during processing in mainframe computers. If any error occurs it can fix it quickly without affecting the performance.

Applications of mainframe computers:

- In **health care**, it enabled hospitals to maintain a record of their millions of patients in order to contact them for treatment or related to their appointment, medicine updates or disease updates.
- In the **field of defence**, it allows the defence departments to share a large amount of sensitive information with other branches of defence.

- In the ***field of education***, it helps big universities to store, manage and retrieve data related to their courses, admissions, students, teachers, employees and affiliated schools and colleges.

3) Minicomputer:

A minicomputer **lies between the mainframe and** microcomputer as it is smaller than mainframe but larger than a microcomputer. It is a midsize multiprocessing computer. It consists of two or more processors and can support 4 to 200 users at one time. Mainframe computers are used in institutes and departments for tasks such as billing, accounting and inventory management.



Characteristics of minicomputer:

- It is light weight that makes it easy to carry and fit anywhere.
- It is less expensive than mainframe computers.

4) Workstation:

Workstation is a **single user computer** that is designed for **technical or scientific applications**. It has a faster microprocessor, a large amount of RAM and high speed graphic adapters. It generally **performs a specific job with great expertise**; accordingly, they are of different types such as graphics workstation, music workstation and engineering design workstation.



Characteristics of workstation computer:

- It is a high-performance computer system designed for a single user for business or professional use.
- It has larger storage capacity, better graphics, and more powerful CPU than a personal computer.
- It can handle animation, data analysis, CAD, audio and video creation and editing.

5) Microcomputer:

Microcomputer is also known as a personal computer. It is a general-purpose computer that is designed for individual use. It has a microprocessor as a central processing unit, memory, storage area, input unit and output unit. Laptops and desktop computers are examples of microcomputers. They are suitable for personal work that may be making an assignment, watching a movie, or at office for office work.



Characteristics of a microcomputer:

- It is the smallest in size among all types of computers.
- A limited number of software can be used.
- It is designed for personal work and applications. Only one user can work at a time.
- It is less expensive and easy to use.

- It is capable of multitasking such as printing, scanning, browsing, watching videos, etc.

Q: What are Applications of computer?

Computers play a role in every field of life. They are used in homes, business, educational institutions, research organizations, medical field, government offices, entertainment, etc.

a. Role of Computer in Home:

Computers are used at homes for several purposes like online bill payment, watching movies at home, social media access, playing games, internet access, etc. They provide communication through electronic mail. Computers help the student community to avail online educational support.

b. Role of Computer in Medical Field:

Computers are used in hospitals to maintain a database of patients' history, diagnosis, X-rays, live monitoring of patients, etc. Surgeons nowadays use robotic surgical devices to perform delicate operations, and conduct surgeries remotely.



c. Role of Computer in Entertainment:

Computers help to watch movies online, play games online, listening to music, etc



d. Role of Computer in Industry:

Computers are used to perform several tasks in industries like managing inventory, designing purpose, creating virtual sample products, interior designing, video conferencing, etc.

e. Role of Computer in Education

Computers are used in education sector through online classes, online examinations, referring e-books, online tutoring, etc. They help in increased use of audio-visual aids in the education field.



f. Role of Computer in Government:

In government sectors, computers are used in data processing, maintaining a database of citizens and supporting a paperless environment.

g. Role of Computer in Banking:

In the banking sector, computers are used to store details of customers and conduct transactions, such as withdrawal and deposit of money through ATMs.



h. Role of Computer in Arts:

Computers are extensively used in dance, photography, arts and culture. The fluid movement of dance can be shown live via animation. Photos can be digitized using computers.

i. Role of Computer in Business:

Nowadays, computers are totally integrated into business. The main objective of business is transaction processing, which involves transactions with suppliers, employees or customers.



j. Role of Computer in Training:

Many organizations use computer-based training to train their employees, to save money and improve performance. Video conferencing through computers allows saving of time and travelling costs by being able to connect people in various locations.

k. Role of Computer in Science and Engineering:

Computers with high performance are used to stimulate dynamic process in Science and Engineering. Supercomputers have numerous applications in area of Research and Development (R&D). Topographic images can be created through computers. Scientists use computers to plot and analyze data to have a better understanding of earthquakes



Q: Explain about generations of computer?

The advances in technology that have led to the development of the many computing devices that we use today. Our journey of the five generations of computers starts in 1940 with vacuum tube circuitry and goes to the present day and beyond with artificial intelligence (AI) systems and devices.

Five generations of computers are:

1. First Generation: Vacuum Tubes
2. Second Generation: Transistors
3. Third Generation: Integrated Circuits
4. Fourth Generation: Microprocessors
5. Fifth Generation: Artificial Intelligence

First generation: vacuum tubes (1940-1956):

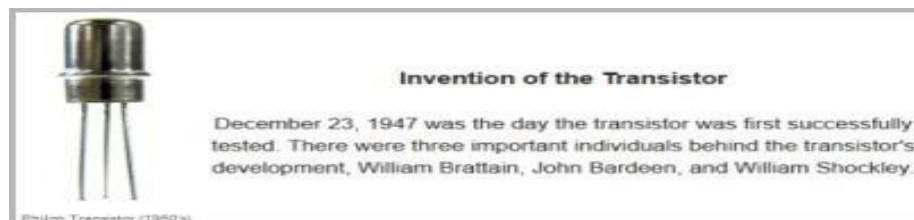
- The first computer systems used vacuum tubes for circuitry and magnetic drums for memory, and were taking up entire rooms.
- These computers were very expensive to operate and in addition to using a great deal of electricity, the first computers generated a lot of heat, which was often the cause of malfunctions.
- First generation computers relied on machine language, the lowest-level programming language understood by computers, to perform operations, and they could only solve one problem at a time.
- It would take operators days or even weeks to set-up a new problem. Input was based on punched cards and paper tape, and output was displayed on printouts.
- The univac and eniac computers are examples of first-generation computing devices. The univac was the first commercial computer delivered to a business client, the u.s. census bureau in 1951.



A univac computer at the census bureau

Second generation: transistors (1956-1963)

- The transistors replace vacuum tubes in the second generation of computers.
- The transistor was far superior to the vacuum tube, allowing computers to become smaller, faster, cheaper, more energy-efficient, and more reliable than their first-generation predecessors.
- Though the transistor still generated a great deal of heat that subjected the computer to damage, it was a vast improvement over the vacuum tube.
- Second-generation computers still relied on punched cards for input and printouts for output.



An early philco transistor (1950s).

Second-generation computers moved from cryptic binary machine language to symbolic, or assembly language which allowed programmers to specify instructions in

words. These were also the first computers that stored their instructions in their memory, which moved from a magnetic drum to magnetic core technology.

Third generation: integrated circuits (1964-1971)

- The development of the integrated circuit was the hallmark of the third generation of computers.
- Transistors were miniaturized and replaced with ic which drastically increased the speed and efficiency of computers.
- Instead of punched cards and printouts, users interacted with third generation computers through keyboards and monitors and interfaced with an operating system, which allowed the device to run many different applications at one time with a central program that monitored the memory.
- Computers for the first time became accessible to a mass audience because they were smaller and cheaper than their predecessors.

Fourth generation: microprocessors (1971-present)

- The microprocessor developed in the fourth generation of computers, as thousands of integrated circuits were built onto a single silicon chip.
- What in the first generation filled an entire room could now fit in the palm of the hand. The intel 4004 chip, developed in 1971, located all the components of the computer from the central processing unit and memory to input/output controls on a single chip.
- in 1981, ibm introduced its first computer for the home user, and in 1984 apple introduced the macintosh.
- As these small computers became more powerful, they could be linked together to form networks, which eventually led to the development of the internet.
- Fourth generation computers also saw the development of guis, the mouse and handheld devices.



Fifth generation: artificial intelligence (present and beyond)

Fifth generation computing devices, based on artificial intelligence, are still in development, though there are some applications, such as voice recognition, that are being used today. The use of parallel processing and superconductors is helping to make artificial intelligence a reality.



Syllabus: Introduction to Algorithms and Programming Languages: Algorithm – Key features of Algorithms, Flow Charts, Programming Languages – Generations of Programming Languages – Structured Programming Language- Design and Implementation of Correct, Efficient and Maintainable Programs.

Q: what is Algorithm?

Algorithm is a step by step procedure, which defines a set of instructions to be executed in certain order to get the desired output.

or

Algorithm is a sequential solution of any program that written in human language called algorithm.

or

Algorithm is not the computer code. Algorithms are just the instruction which gives clear idea to write the computer code.

Characteristics of an Algorithm:

- A. **Unambiguous** – Algorithm should be clear and unambiguous. Each of its steps and their input/outputs should be clear and must lead to only one meaning.
- B. **Input** – an algorithm should have 0 or more well defined inputs.
- C. **Output** – an algorithm should have 1 or more well defined outputs, and should match the desired output.
- D. **Finiteness** – Algorithms must terminate after a finite number of steps.
- E. **Feasibility** – should be feasible with the available resources.
- F. **Independent** – an algorithm should have step-by-step directions which should be independent of any programming code.
- G. **Must be effective**- this means that an algorithm must provide the correct answers at all times.

Ex: Write a algorithm to find out number is odd or even?

```
step 1 : start
step 2 : input number
step 3 : rem=number mod 2
step 4 : if rem=0 then
            print "number even"
            else
            print "number odd"
            endif
step 5 : stop
```

Ex: Write an algorithm to add two numbers entered by user.

```
Step 1:    Start
Step 2:    Declare variables num1, num2 and sum.
Step 3:    Read values num1 and num2.
```

Step 4: Add num1 and num2 and assign the result to sum.
 $Sum \leftarrow num1 + num2$
 Step 5: Display sum
 Step 6: Stop

Ex: Write an algorithm to find the largest among three different numbers entered by user.

Step 1: Start
 Step 2: Declare variables a,b and c.
 Step 3: Read variables a,b and c.
 Step 4: If $a > b$
 If $a > c$
 Display a is the largest number.
 Else
 Display c is the largest number.
 Else
 If $b > c$
 Display b is the largest number.
 Else
 Display c is the greatest number.
 Step 5: Stop

Ex: Write an algorithm to find all roots of a quadratic equation $ax^2+bx+c=0$.

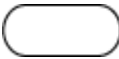

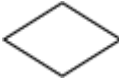




Step 1: Start
 Step 2: Declare variables a, b, c, D, x1, x2, rp and ip;
 Step 3: Calculate discriminant
 $D \leftarrow b^2 - 4ac$
 Step 4: If $D \geq 0$
 $r1 \leftarrow (-b + \sqrt{D}) / 2a$
 $r2 \leftarrow (-b - \sqrt{D}) / 2a$
 Display r1 and r2 as roots.
 Else
 Calculate real part and imaginary part
 $rp \leftarrow -b / 2a$
 $ip \leftarrow \sqrt{-D} / 2a$
 Display $rp + j(ip)$ and $rp - j(ip)$ as roots
 Step 5: Stop

Q: Define Flowchart:

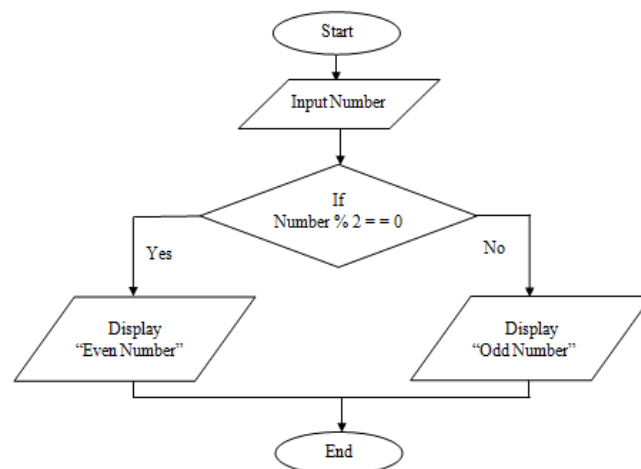
1. Graphical representation of any program is called flowchart.
2. There are some standard graphics that are used in flowchart as following:

Common Flowchart Symbols

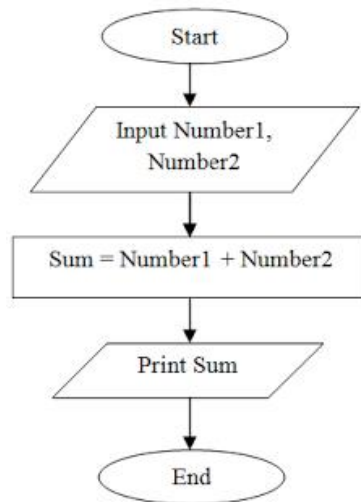
Different flow chart symbols have different meanings. The most common flow chart symbols are:

- **Terminator:** An oval flow chart shape indicating the start or end of the process. 
- **Process:** A rectangular flow chart shape indicating a normal process flow step. 
- **Decision:** A diamond flow chart shape indicating a branch in the process flow. 
- **Connector:** A small, labeled, circular symbol. This symbol is typically small and is used as a Connector to show a jump from one point in the process flow to another. Connectors are usually labeled with capital letters (A, B, AA) to show matching jump points. 
- **Data:** A parallelogram that indicates data input or output (I/O) for a process. 
- **Document:** Used to indicate a document or report 
- **Flow line:**Used to indicate the flow of logic by connecting symbols. 

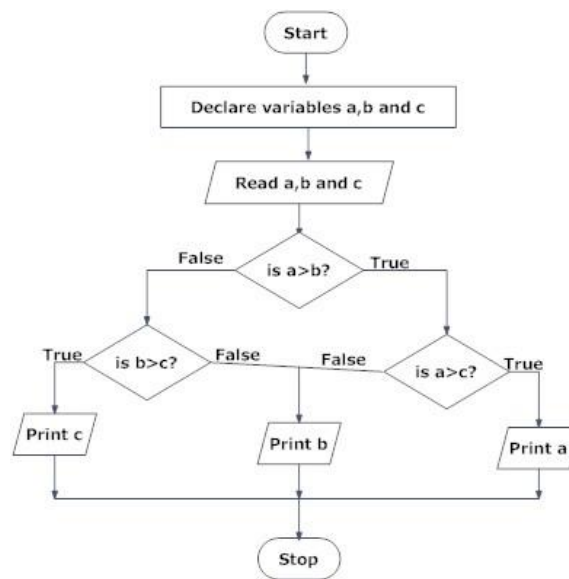
Draw Flowchart to check Odd or Even Number.



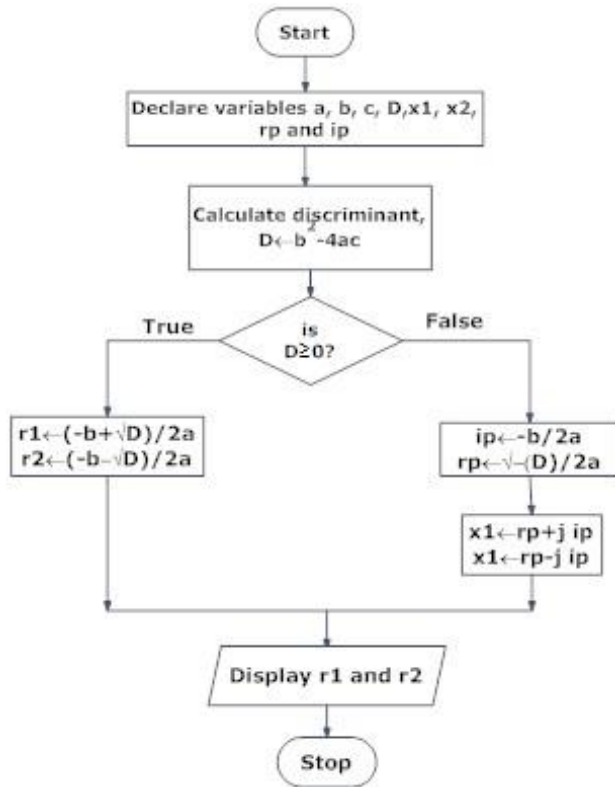
Draw a flowchart to add two numbers entered by user.



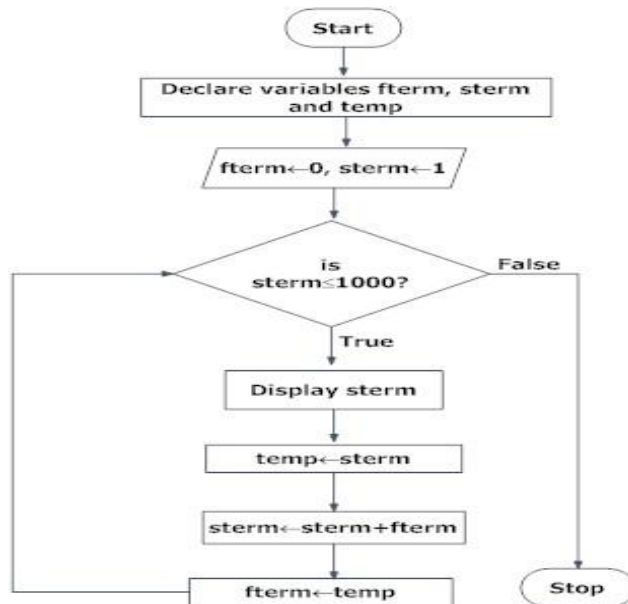
Draw flowchart to find the largest among three different numbers entered by user.



Draw a flowchart to find all the roots of a quadratic equation $ax^2+bx+c=0$



Draw a flowchart to find the Fibonacci series till term ≤ 1000 .



Q: Definition of 'Pseudo code'?

Pseudo code is not an actual programming language. So it cannot be compiled into an executable program. It uses short terms or simple English language syntaxes to write code for programs before it is actually converted into a specific programming language. This is done to identify top level flow errors, and understand the programming data flows that the final program is going to use. This helps save time during actual programming.

It cannot be compiled into an executable program.

Example:	C code :	<code>if (i < 10) { i++; }</code>
	pseudocode :	if i is less than 10, increment i by 1.

Ex: Algorithm, pseudo code, c program print array.

Algorithm: The step-by-step procedure of this program –

START

Step 1 → Take an array A and define its values

Step 2 → Loop for each value of A

Step 3 → Display A[n] where n is the value of current iteration

STOP

Pseudo code: The pseudo code of this algorithm –

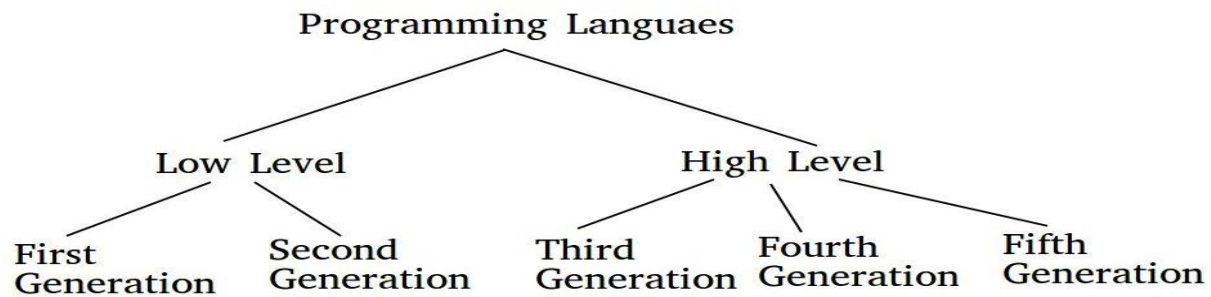
```
procedure print_array(A)
  FOR EACH value in A DO
    DISPLAY A[n]
  END FOR
end procedure
```

Implementation: The implementation of the above derived pseudocode is as follows –

```
#include <stdio.h>
int main() {
int array[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 0};
int loop;
for(loop = 0; loop < 10; loop++)
printf("%d ", array[loop]);
return 0;
}
```

Q: Generation of Programming Languages:

Programming languages have been classified into several programming language generations. Historically, this classification was used to indicate increasing power of programming styles.



1. First Generation Language :

The first generation languages are also called machine languages/ 1G language. This language is machine-dependent. The machine language statements are written in binary code (0/1 form) because the computer can understand only binary language.

Advantages :

1. Fast & efficient as statements are directly written in binary language.
2. No translator is required.

Disadvantages :

1. Difficult to learn binary codes.
2. Difficult to understand – both programs & where the error occurred.

2. Second Generation Language :

The second-generation languages are also called assembler languages/ 2G languages. Assembly language contains human-readable notations that can be further converted to machine language using an assembler.

Assembler – converts assembly level instructions to machine level instructions.

Programmers can write the code using symbolic instruction codes that are meaningful abbreviations of mnemonics. It is also known as low-level language.

Advantages :

1. It is easier to understand if compared to machine language.
2. Modifications are easy.
3. Correction & location of errors are easy.

Disadvantages :

1. Assembler is required.
2. This language is architecture /machine-dependent, with a different instruction set for different machines.

3. Third Generation Language :

The third generation is also called procedural language /3 GL. It consists of the use of a series of English-like words that humans can understand easily, to write instructions. Its also called High-Level Programming Language.

For execution, a program in this language needs to be translated into machine language using Compiler/ Interpreter. Examples of this type of language are C, PASCAL, FORTRAN, COBOL, etc.

Advantages :

1. Use of English-like words makes it a human-understandable language.
2. Lesser number of lines of code as compared to above 2 languages.
3. Same code can be copied to another machine & executed on that machine by using compiler-specific to that machine.

Disadvantages :

1. Compiler/ interpreter is needed.
2. Different compilers are needed for different machines.

4. Fourth Generation Language :

The fourth-generation language is also called a non – procedural language/ 4GL. It enables users to access the database. Examples: SQL, Foxpro, Focus, etc.

These languages are also human-friendly to understand.

Advantages :

1. Easy to understand & learn.
2. Less time required for application creation.
3. It is less prone to errors.

Disadvantages :

1. Memory consumption is high.
2. Has poor control over Hardware.
3. Less flexible.

5. Fifth Generation Language :

The fifth-generation languages are also called 5GL. It is based on the concept of artificial intelligence. we make computers learn to solve any problem. Parallel Processing & superconductors are used for this type of language to make real artificial intelligence.

Example: PROLOG, LISP, etc.

Advantages :

1. Machines can make decisions.
2. Programmer effort reduces to solve a problem.
3. Easier than 3GL or 4GL to learn and use.

Disadvantages :

1. Complex and long code.
2. More resources are required & they are expensive too

Q: Explain about Structured Programming Approach?

Structured Programming Approach, as the word suggests, can be defined as a programming approach in which the program is made as a single structure. It means that the code will execute the instruction by instruction one after the other. It doesn't support the possibility of jumping from one instruction to some other with the help of any statement like GOTO, etc. Therefore, the instructions in this approach will be executed in a serial and structured manner. The languages that support Structured programming approach are: C, C++

The structured program mainly consists of three types of elements:

- ✓ Selection Statements
- ✓ Sequence Statements
- ✓ Iteration Statements

The structured program consists of well structured and separated modules. But the entry and exit in a Structured program is a single-time event. It means that the program uses single-entry and single-exit elements. Therefore a structured program is well maintained, neat and clean program. This is the reason why the Structured Programming Approach is well accepted in the programming world.

Advantages of Structured Programming Approach:

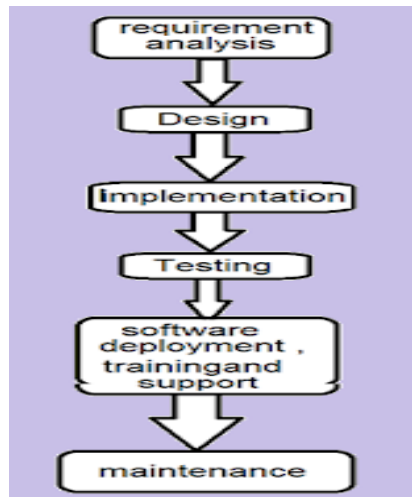
1. Easier to read and understand
2. User Friendly
3. Easier to Maintain
4. Mainly problem based instead of being machine based
5. Development is easier as it requires less effort and time
6. Easier to Debug
7. Machine-Independent, mostly.

Disadvantages of Structured Programming Approach:

1. Since it is Machine-Independent, So it takes time to convert into machine code.
2. The converted machine code is not the same as for assembly language.
3. The program depends upon changeable factors like data-types. Therefore it needs to be updated with the need on the go.
4. Usually the development in this approach takes longer time as it is language-dependent. Whereas in the case of assembly language, the development takes lesser time as it is fixed for the machine.

Q: Explain about Design and Implementation of Correct Efficient and Maintainable Programs?

The entire program or software (collection of programs) development process is divided in to a number of phases, where each phase performs a well- defined task



Phases in Software development Life cycle.

Requirements analysis:

in this phase , the user's expectations are gathered to understand why the program or software has to be developed. Then , all the gathered requirements are analyzed The functionality, capability, performance, and availability of hardware and software components are all analysed in this phase.

Design:

The requirement documented in the previous phase act as the input to the design phase. In this phase, a plan of actions is made before the actual development process starts. This plan will be followed throughout the development process.

Implementation:

In this phase, the designed algorithms are converted in to program code using any of the high level languages. The particular choice of language will depend on the type of application program. C is preferred for writing system programs

Testing:

In this phase , all the modules are tested together to ensure that the overall system works well as a whole product. In this phase, the software is tested using a large number of varied inputs.

Software deployment, training, and support:

After the code is tested and the software or the program is approved by the user's it is then installed or deployed in the production environment.

Maintenance:

Maintenance and enhancements are ongoing activities that are done to cope with newly discovered problems or new requirements. Such activities may take a long time to complete

UNIT II :Introduction to C

Introduction – Structure of C Program – Writing the first C Program – File used in C Program – Compiling and Executing C Programs – Using Comments – Keywords – Identifiers – Basic Data Types in C – Variables – Constants – I/O Statements in C- Operators in C- Programming Examples. Decision Control and Looping Statements: Introduction to Decision Control Statements– Conditional Branching Statements – Iterative Statements – Nested Loops – Break and Continue Statement – Goto Statement

C-PROGRAMMING

Q: Explain about C – Language History?

- The C language is a structure oriented programming language, developed at AT&T Bell Laboratories in USA 1972 by Dennis M Ritchie
- C language features were derived from an earlier language called “B” (Basic Combined Programming Language – BCPL).
- C language was invented for implementing UNIX operating system
- In 1978, Dennis Ritchie and Brian Kernighan published the first edition “The C Programming Language” and commonly known as K&R C
- In 1983, the **American National Standards** Institute (ANSI) established a committee to provide a modern, comprehensive definition of C. The resulting definition, the ANSI standard, or “ANSI C”, was completed late 1988.
- It is also called as procedure oriented programming language. The C language is reliable, simple and easy to use. C has been coded in assembly language.
- C was originally first implemented on the DEC PDP-11 computer in 1972.
- C is middle level language because the features of assembly language and high-level language are combined by the C compiler.

Q: Explain about Structure of a C program?

Structure of C program is defined by set of rules called protocol, to be followed by programmer while writing C program. All C programs are having the following sections.

1. Documentation section
2. Link Section
3. Definition Section
4. Global declaration section
5. Function prototype declaration section
6. Main function
7. User defined function definition section

1. **Documentation section:** We can give comments about the program, creation or modified date, author name etc in this section. The characters or words or anything which are given between “/*” and “*/”, won’t be considered by C compiler for compilation process. These will be ignored by C compiler during compilation.
Ex: /* comment line1 comment line2 comment 3 */
2. **Link Section:** Header files that are required to execute a C program are included in this section
3. **Definition Section:** In this section, variables are defined and values are set to these variables.
4. **Global declaration section:** Global variables are defined in this section. When a variable is to be used throughout the program, can be defined in this section.
5. **Function prototype declaration section:** Function prototype gives many information about a function like return type, parameter names used inside the function.
6. **Main function:** Every C program is started from main function and this function contains two major sections called declaration section and executable section.
7. **User defined function section:** User can define their own functions in this section which perform particular task as per the user requirement.

Example C program to compare all the sections:

```

/* C basic structure program Documentation section
Author: fresh2refresh.com
Date : 01/01/2012
*/
#include <stdio.h>          /* Link section */
int total = 0;             /* Global declaration and definition section */
int sum (int, int);       /* Function declaration section */
int main ()               /* Main function */
{
printf (“This is a C basic program \n”);
total = sum (1, 1);
printf (“Sum of two numbers : %d \n”, total);
return 0;
}
int sum (int a, int b)    /* User defined function */
{                         /* definition section */
return a + b;
}

```

The compilation and execution process of C can be divided in to multiple steps:

- **Preprocessing** - Using a Preprocessor program to convert C source code in expanded source code. "#includes" and "#defines" statements will be processed and replaced actually source codes in this step.
- **Compilation** - Using a Compiler program to convert C expanded source to assembly source code.
- **Assembly** - Using a Assembler program to convert assembly source code to object code.
- **Linking** - Using a Linker program to convert object code to executable code. Multiple units of object codes are linked to together in this step.
- **Loading** - Using a Loader program to load the executable code into CPU for execution.

Q: What are tokens in C-Language?

- C tokens are the basic buildings blocks in C language which are constructed together to write a C program.
- Each and every smallest individual unit in a C program is known as C tokens.
- C tokens are of six types. They are,
 1. Keywords
 2. Identifiers
 3. Constants
 4. Strings
 5. Special symbols
 6. Operators

1. Keywords in C language:

- Keywords are pre-defined words in a C compiler.
- Each keyword is meant to perform a specific function in a C program.
- Since keywords are referred names for compiler, they can't be used as variable name.
- C language supports 32 keywords.

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

2. Identifiers in C language:

- Each program elements in a C program are given a name called identifiers.
- Names given to identify Variables, functions and arrays are examples for identifiers.

eg. x is a name given to integer variable in above program.

3. Constant:

- C Constants are also like normal variables. But, only difference is, their values cannot be modified by the program once they are defined.
- Constants refer to fixed values. They are also called as literals

Syntax: const data_typevariable_name; (or)
 const data_type *variable_name;

Types of C constant:

S.no	Constant type	data type	Example
1	Integer constants	int unsigned int long int long long int	53, 762, -478 etc 5000u, 1000U etc 483,647 2,147,483,680
2	Real or Floating point constants	float double	10.456789 600.123456789
3	Octal constant	int	013 /* starts with 0 */
4	Hexadecimal constant	int	0x90 /* starts with 0x */
5	character constants	char	'A' , 'B' , 'C'
6	string constants	char	"ABCD" , "Hai"

Escape sequence characters in C:

- ✓ There are some characters which have special meaning in C language.
- ✓ They should be preceded by backslash symbol to make use of special function of them.

Backslash character	Meaning
\\b	Backspace
\\f	Form feed
\\n	New line
\\r	Carriage return
\\t	Horizontal tab
\\"	Double quote
\\'	Single quote
\\\\	Backslash
\\v	Vertical tab
\\a	Alert or bell
\\?	Question mark
\\N	Octal constant (N is an octal constant)

\XN	Hexadecimal constant (N – hex.dcm1cnst)
-----	---

Strings: They are always represented as an array of characters having null character '\0' at the end of the string. This null character denotes the end of the string. Strings in C are enclosed within double quotes, while characters are enclosed within single characters. The size of a string is a number of characters that the string contains.

Special symbols in C:

Some special characters are used in C, and they have a special meaning which cannot be used for another purpose. Those are

- Square brackets []
- Simple brackets () Curly braces { }
- Comma (,): Hash/pre-processor (#)
- Asterisk (*)
- Tilde (~): It is used as a destructor to free memory.
- Period (.)

Q: Explain C – Data Types?

- ✓ C **data types** are defined as the data storage format that a variable can store a data to perform a specific operation.
- ✓ Data types are used to define a variable before to use in a program.
- ✓ Size of variable, constant and array are determined by data types.

C – data types are,

S.no	Types	Data Types
1	Basic data types	int, char, float, double
2	Enumeration data type	enum
3	Derived data type	pointer, array, structure, union
4	Void data type	void

Basic data types in C:

1. Integer data type:

- ✓ Integer data type allows a variable to store numeric values.
- ✓ “int” keyword is used to refer integer data type.
- ✓ The storage size of int data type is 2 or 4 or 8 byte.
- ✓ It varies depend upon the processor in the CPU that we use.

If we are using 16 bit processor 2 byte of memory will be allocated for int data type.

32 bit processor 4 byte of memory will be allocated for int data type

64 bit processor 8 byte of memory will be allocated for int data type.

- ✓ int (2 byte) can store values from -32,768 to +32,767
- ✓ int (4 byte) can store values from -2,147,483,648 to +2,147,483,647.
- ✓ If you want to use the integer value that crosses the above limit, you can go for “long int” and “long long int” for which the limits are very high.

2. Character data type:

- ✓ Character data type allows a variable to store only one character.
- ✓ Storage size of character data type is 1 byte.
- ✓ “char” keyword is used to refer character data type.
- ✓ Character datatype values are enclosed in single quotations marks.

3. Floating point data type:

Floating point data type consists of 2 types. They are **float, double**

3.1 float:

- ✓ Float data type allows a variable to store decimal values.
- ✓ Storage size of float data type is 4. This also varies depend upon the processor in the
- ✓ We can use up-to 6 digits after decimal using float data type.

Ex: 10.456789 can be stored in a variable using float data type.

3.2 double:

- ✓ Double data type is also same as float data type which allows up-to 10 digits after decimal.
- ✓ The range for double data type is from $1E-37$ to $1E+37$.

Modifiers in C:

- ✓ The amount of memory space to be allocated for a variable is derived by modifiers.
- ✓ Modifiers are prefixed with basic data types to modify (either increase or decrease) the amount of storage space allocated to a variable.

Ex: storage space for int data type is 4 byte for 32 bit processor.

We can increase the range by using long int which is 8 byte.

We can decrease the range by using short int which is 2 byte.

There are 5 modifiers available in C language. They are,

1. short
2. long

3. signed
4. unsigned
5. long long

S.No	Data types	Storage Size	Range
1	char	1	-127 to 127
2	int	2	-32,767 to 32,767
3	float	4	IE-37 to IE +37 with six digits of precision
4	double	8	IE -37 to IE +37 with ten digits of precision
5	long double	10	IE -37 to IE +37 with ten digits of precision
6	long int	4	-2,147,483,647 to 2,147,483,647
7	short int	2	-32,767 to 32,767
8	unsigned short int	2	0 to 65,535
9	signed short int	2	-32,767 to 32,767
10	long long int	8	$-(2^{\text{power}(63)} - 1)$ to $2^{\text{power}(63)} - 1$
11	signed long int	4	-2,147,483,647 to 2,147,483,647
12	unsigned long int	4	0 to 4,294,967,295
13	unsigned long long int	8	$2^{\text{power}(64)} - 1$

Derived data type in C:

Array, pointer, structure and union are called derived data type in C language.

Void data type in C:

- ✓ Void is an empty data type that has no value.
- ✓ This can be used in **functions** and pointers.

Q: Explain about variable in c?

A variable is nothing but a name given to a storage area. This location is used to hold the value of the variable.

- ✓ The value of the C variable may get change in the program.
- ✓ C variable might be belonging to any of the data type like int, float, char etc.

Rules for naming C variable:

- ✓ Variable name must begin with letter or underscore.
- ✓ Variables are case sensitive.
- ✓ They can be constructed with digits, letters.

- ✓ No special symbols are allowed other than underscore.

Declaring & initializing C variable:

- ✓ Variables should be declared in the C program before to use.
- ✓ Memory space is not allocated for a variable while declaration. It happens only on variable definition.
- ✓ Variable initialization means assigning a value to the variable.

Variable declaration: syn: data_typevariable_name;

Ex : int x, y, z;
char flat, ch;

Variable initialization: syn: data_typevariable_name = value;

Ex: int x = 50, y = 30;
char flag = 'x', ch='l';

There are three types of variables in C program they are.

- Local variable.
- Global variable.
- Environment variable.

A. Local variable in C:

- ✓ These variables are declared within the function and can't be accessed outside the function
- ✓ The scope of local variables will be within the function only.

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

```
void test();
```

```
int main()
```

```
{
```

```
    int m = 22, n = 44;           // m, n are local variables of main function
```

```
    printf("\nvalues : m = %d and n = %d", m, n);
```

```
    test();
```

```
}
```

```
void test()
```

```
{
```

```
    int a = 50, b = 80;           // a, b are local variables of test function
```

```
    printf("\nvalues : a = %d and b = %d", a, b);
```

```
}
```

B. Global variable in C:

- ✓ The scope of global variables will be throughout the program. These variables can be accessed from anywhere in the program.

- ✓ This variable is defined outside the main function. So that, this variable is visible to main function and all other sub functions.

```
#include<stdio.h>

void test();

int m = 22, n = 44;

int a = 50, b = 80;

int main()
{
    printf("All variables are accessed from main function");
    printf("\nvalues : m= %d : n= %d : a= %d : b= %d",m,n,a,b);
    test();
}

void test()
{
    printf("\n\nAll variables are accessed from" \ " test function");

    printf("\nvalues : m= %d : n= %d : a= %d : b= %d",m,n,a,b);
}
```

C. Environment variables in C:

- ✓ Environment variable is a variable that will be available for all C applications and C programs.
- ✓ We can access these variables from anywhere in a C program without declaring and initializing in an application or C program.
- ✓ The inbuilt functions which are used to access, modify and set these environment variables are called environment functions.
- ✓ There are 3 functions which are used to access, modify and assign an environment variable in C. They are,

- 1.setenv()
- 2.getenv()
- 3.putenv()

getenv(): This function gets the current value of the environment variable.

Example: /usr/bin/test/

setenv(): This function sets the value for environment variable.

Example: File = /usr/bin/example.c

putenv(): This function modifies the value for environment variable.

example: Directory name before modifying = /usr/bin/example/
Directory name after modifying = /usr/home/

Difference between variable declaration & definition in C:

S.no	Variable declaration	Variable definition
1	Declaration tells the compiler about data type and size of the variable.	Definition allocates memory for the variable.
2	Variable can be declared many times in a program.	It can happen only one time for a variable in a program.
3	The assignment of properties and identification to a variable.	Assignments of storage space to a variable.

Q: Explain about Constant in C language?

- ✓ C Constants are also like normal variables. But, only difference is, their values cannot be modified by the program once they are defined.
- ✓ Constants refer to fixed values. They are also called as literals

Syntax: const data_type variable_name; (or)
const data_type *variable_name;

Types of C constant:

S.no	Constant type	data type	Example
1	Integer constants	int unsigned int long int long long int	53, 762, -478 etc 5000u, 1000U etc 483,647 2,147,483,680
2	Real or Floating point constants	float double	10.456789 600.123456789
3	Octal constant	int	013 /* starts with 0 */
4	Hexadecimal constant	int	0x90 /* starts with 0x */
5	character constants	char	'A' , 'B', 'C'
6	string constants	char	"ABCD" , "Hai"

Rules for constructing C constant:

- Integer Constants in C:**
 - An integer constant must have at least one digit.

- It must not have a **decimal point**.
- It can either be positive or negative.
- No commas or blanks are allowed within an integer constant.
- If no sign precedes an integer constant, it is assumed to be positive.
- The allowable range for integer constants is -32768 to 32767.

b. Real constants in C:

- A real constant must have at least one digit.
- It must have a decimal point.
- It could be either positive or negative.
- If no sign precedes an integer constant, it is assumed to be positive.
- No commas or blanks are allowed within a real constant.

c. Character and string constants in C:

- A character constant is a single alphabet, a single digit or a single special symbol enclosed within single quotes.
- The maximum length of a character constant is 1 character.
- String constants are enclosed within double quotes.

d. Backslash Character Constants in C(escape sequence characters):

- There are some characters which have special meaning in C language.
- They should be preceded by backslash symbol to make use of special function of them.(Refer above for list of backslash characters)

Q: How to use constants in a C program?

- We can define constants in a C program in the following ways.
 - By “const” keyword
 - By “#define” preprocessor directive
- Please note that when you try to change constant values after defining in C program, it will through **error**.

1. Example program using const keyword in C:

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

```
void main()
```

```
{
```

```
    const int height = 100;           /*int constant*/
```

```
    const float number = 3.14;       /*Real constant*/
```

```
    const char letter = 'A';         /*char constant*/
```

```
    const char letter_sequence[10] = "ABC"; /*string constant*/
```

```
    const char backslash_char = '\\'; /*special char cnst*/
```

```

printf("value of height  : %d \n", height );
printf("value of number : %f \n", number );
printf("value of letter : %c \n", letter );
printf("value of letter_sequence : %s \n", letter_sequence);
printf("value of backslash_char : %c \n", backslash_char);
}

```

2. Example program using #define preprocessor directive in C:

```

#include <stdio.h>

#define height 100
#define number 3.14
#define letter 'A'
#define letter_sequence "ABC"
#define backslash_char '\?'

void main ()
{
    printf("value of height  : %d \n", height );
    printf("value of number : %f \n", number );
    printf("value of letter : %c \n", letter );
    printf("value of letter_sequence : %s \n", letter_sequence);
    printf("value of backslash_char : %c \n", backslash_char);
}

```

Q: Explain about I/O statements in C?

There are some library functions which are available for transferring the information between the computer and the standard input and output devices. These functions are related to the symbolic constants and are available in the header file.

Some of the input and output functions are as follows:

i) **C printf() function:**

- printf() function is used to print the character, string, float, integer value onto the output screen.
- To generate a newline, we use “\n” in C printf() statement.

Syntax: printf("format specifier",arg1,arg2,.....);

Example: printf("enter a value:");

ii) **C scanf() function:**

➤ scanf() function is used to read character, string, numeric data from keyboard.

➤ We use format specifiers with scanf() function. Those are

%d is used to display integer variable,

%c is used to display character,

%f for float variable,

%s for string variable,

%lf for double variable,

%x for hexadecimal variable.

Syntax: scanf("format string",&arg1,&arg2,.....);

Arguments can be variables. Each variable must be preceded by an ampersand (&).

Example: int avg;

float per;

char grade;

scanf("%d %f %c",&avg, &per, &grade);

iii) **getch():** This function is used to input a single character. The character is read instantly and it does not require an enter key to be pressed. The character type is returned but it does not echo on the screen.

Syntax: int getch(void);

ch=getch();

where, **ch** - assigned the character that is returned by getch.

iv) **putch():** this function is a counterpart of getch(). Which means that it will display a single character on the screen. The character that is displayed is returned.

Syntax: int putch(int);

putch(ch); where, **ch** - the character that is to be printed.

v) **getche():** This function is used to input a single character. The main difference between getch() and getche() is that getche() displays the (echoes) the character that we type on the screen.

Syntax: int getch(void);

ch=getche();

vi) **getchar():** This function is used to input a single character. The enter key is pressed which is followed by the character that is typed. The character that is entered is echoed.

Syntax: `ch=getchar();`

vii) putchar: This function is the other side of `getchar`. A single character is displayed on the screen.

Syntax: `putchar(ch);`

viii) gets() and puts(): They help in transferring the strings between the computer and the standard input-output devices. Only single arguments are accepted. The arguments must be such that it represents a string. It may include white space characters. If `gets` is used enter key has to be pressed for ending the string.

The `gets` and `puts` function are used to offer simple alternatives of `scanf` and `printf` for reading and displaying.

```
Ex: #include <stdio.h>
void main()
{
    char line[30];
    gets (line);
    puts (line);
}
```

Q: Explain about C – Operators and Expressions:

- ✓ The symbols which are used to perform logical and mathematical operations in a C program are called C operators.
- ✓ These C operators join individual constants and variables to form expressions.
- ✓ Operators, functions, constants and variables are combined together to form expressions.

Ex: Consider the **expression** $A + B * 5$.

Where, $+$, $*$ are operators, A , B are variables, 5 is constant and $A + B * 5$ is an expression.

Types of C operators:

1. Arithmetic operators
2. Relational operators
3. Logical operators
4. Bit wise operators
5. Conditional operators (ternary operators)
6. Increment/decrement operators
7. Assignment operators
8. Special operators

Arithmetic Operators in C:

These are used to perform mathematical calculations like addition, subtraction, multiplication, division and modulus.

S.no	Arithmetic Operators	Operation	Example
------	----------------------	-----------	---------

1	+	Addition	A+B
2	-	Subtraction	A-B
3	*	multiplication	A*B
4	/	Division	A/B
5	%	Modulus	A%B

Example program for C arithmetic operators:

```
#include <stdio.h>

int main()
{
    int a=40,b=20, add,sub,mul,div,mod;
    add = a+b;
    sub = a-b;
    mul = a*b;
    div = a/b;
    mod = a%b;
    printf("Addition of a, b is : %d\n", add);
    printf("Subtraction of a, b is : %d\n", sub);
    printf("Multiplication of a, b is : %d\n", mul);
    printf("Division of a, b is : %d\n", div);
    printf("Modulus of a, b is : %d\n", mod);
}
```

1. Relational operators in C:

These are used to find the relation between two variables. i.e. to compare the values of two variables in a C program.

S.no	Operators	Example	Description
1	>	x > y	x is greater than y
2	<	x < y	x is less than y
3	>=	x >= y	x is greater than or equal to y
4	<=	x <= y	x is less than or equal to y
5	==	x == y	x is equal to y
6	!=	x != y	x is not equal to y

Ex:

```
#include <stdio.h>
int main()
{
    int m=40,n=20;
    if (m == n)
    {
```

```

printf("m and n are equal");
}
else
{
printf("m and n are not equal");
}
}

```

2. Logical operators in C:

- ✓ These **operators** are used to perform **logical** operations on the given expressions.
- ✓ There are 3 logical operators in C language. They are, logical AND (&&), logical OR (||) and logical NOT (!).

S.no	Operators	Name	Example	Description
1	&&	logical AND	(x>5)&&(y<5)	It returns true when both conditions are true
2		logical OR	(x>=10) (y>=10)	It returns true when at-least one of the condition is true
3	!	logical NOT	!((x>5)&&(y<5))	It reverses the state of the operand “((x>5) && (y<5))” If “((x>5) && (y<5))” is true, logical NOT operator makes it false

Example program for logical operators in C:

```

#include <stdio.h>

int main()
{
int m=40,n=20,p=30;
if (m>n && m>p)
{
printf("m is maximum value");
}
else if (n>p)
{
printf("n is maximum value");
}
else

```

```

    {
        printf("p is maximum value");
    }
}

```

3. Bit wise operators in C:

- ✓ These **operators** are used to perform bit operations. Decimal values are converted into **binary** values which are the sequence of bits and bit wise operators work on these bits.
- ✓ Bit wise operators in C language are & (bitwise AND), | (bitwise OR), ~ (bitwise NOT), ^ (XOR), << (left shift) and >> (right shift).

x	y	x y	x & y	x ^ y
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	0

Operatorsymbol	Operatortname
&	Bitwise AND
	Bitwise OR
~	Bitwise NOT
^	XOR
<<	Left Shift
>>	Right Shift

Ex: Consider x=40 and y=80.

Binary form of these values are x = 00101000 y= 01010000

All bit wise operations for x and y are.

x & y = 00000000 (binary) = 0 (decimal)

x|y = 01111000 (binary) = 120 (decimal)

Example program for bit wise operators in C:

Bit wise operations are performed as shown above and output is displayed in decimal format.

```

#include <stdio.h>
int main()
{
    int m=40,n=80,AND_opr,OR_opr,XOR_opr,NOT_opr ;
    AND_opr = (m&n);
    OR_opr = (m|n);
    NOT_opr = (~m);
}

```



```

XOR_opr = (m^n);
printf("AND_opr value = %d\n",AND_opr );
printf("OR_opr value = %d\n",OR_opr );
printf("NOT_opr value = %d\n",NOT_opr );
printf("XOR_opr value = %d\n",XOR_opr );
printf("left_shift value = %d\n", m << 1);
printf("right_shift value = %d\n", m >> 1);

}

```

Output:

```

AND_opr value = 0
OR_opr value = 120
NOT_opr value = -41
XOR_opr value = 120
left_shift value = 80
right_shift value = 20

```

4. Conditional or ternary operators in C:

- ✓ **Conditional operators** return one value if condition is true and returns another value if condition is false.
- ✓ This operator is also called as ternary operator.

Syntax : (Condition? true_value: false_value);

Ex : (A > 100 ? 0 : 1);

if A is greater than 100, 0 is returned else 1 is returned. This is equal to if else conditional statements.

Example program for conditional/ternary operators in C:

```

#include <stdio.h>
int main()
{
    int x=1, y;
    y = ( x ==1 ? 2 : 0 );
    printf("x value is %d\n", x);
    printf("y value is %d", y);
}

```

5. C – Increment/decrement Operators:

Increment operators are used to increase the value of the variable by one and **decrement** operators are used to decrease the value of the variable by one in C programs.

Syntax: Increment operator : ++var_name; (or) var_name++;

Decrement operator : -- var_name; (or) var_name --;

Example: Increment operator : ++ i; i++;

Decrement operator: -- i; i--;

Example for increment **operators**

```

#include <stdio.h>

int main()
{

```

```

int i=1;
while(i<10)
{
printf("%d ",i);
i++;
}
}

```

Difference between pre/post increment & decrement operators in C:

S.no	Operator type	Operator	Description
1	Pre increment	++i	Value of i is incremented before assigning it to variable i.
2	Post increment	i++	Value of i is incremented after assigning it to variable i.
3	Pre decrement	--i	Value of i is decremented before assigning it to variable i.
4	Post decrement	i--	Value of i is decremented after assigning it to variable i.

6. Assignment operators in C:

These are used to assign the values for the variables in C programs.

Operators		Example	Explanation
Simple assignment operator	=	sum=10	10 is assigned to variable sum
Compound assignment operators	+=	sum+=10	This is same as sum = sum + 10
	-=	sum-=10	This is same as sum = sum - 10
	=	sum=10	This is same as sum = sum * 10
	/=	sum/=10	This is same as sum = sum / 10
	%=	sum%=10	This is same as sum = sum % 10

Example program for C assignment operators:

```

#include <stdio.h>
int main()
{
int Total=0,i;
for(i=0;i<10;i++)

```

```

    {
    Total+=i; // This is same as Total = Total+i
    }
    printf("Total = %d", Total);
}

```

7. Special Operators in C:

S.no	Operators	Description
1	&	This is used to get the address of the variable. Ex :&a will give address of a.
2	*	This is used as pointer to a variable. Ex : * a where, * is pointer to the variable a.
3	Sizeof ()	Used to find the memory space allocated for each C data types Ex : size of (char) will give us 1.

Example program for & and * operators in C:

```

#include <stdio.h>
int main ()
{
    int *ptr, q;
    q = 50;                /* address of q is assigned to ptr */
    ptr = &q;              /* display q's value using ptr variable */
    printf ("%d", *ptr);
    return 0;
}

```

Example program for sizeof() operator in C:

```

#include <stdio.h>
#include <limits.h>
int main()
{
    int a;
    char b;
    float c;
    double d;
    printf("Storage size for int data type:%d \n",sizeof(a));
    printf("Storage size for char data type:%d \n",sizeof(b));
}

```

```

printf("Storage size for float data type:%d \n",sizeof(c));
printf("Storage size for double data type:%d\n",sizeof(d));
return 0;
}

```

Q:What is Type Casting in C? Explain its types?

Typecasting concept in C language is used to modify a variable from one data type to another data type. New data type should be mentioned before the variable name or value in brackets which to be typecast.

Ex: (datatype)expression;

- It is best practice to convert lower data type to higher data type to avoid data loss.
- Data will be truncated when higher data type is converted to lower.

Ex: if float is converted to int, data which is present after decimal point will be lost.

Implicit type casting: Implicit conversions can be performed by the compiler automatically.

Example:

- ✓ In the below C program, 7/5 alone will produce integer value as 1.
- ✓ So, type cast is done before division to retain float value (1.4).

```

#include <stdio.h>
int main ()
{
    float x;
    x = (float) 7/5;
    printf("%f",x);
}

```

Explicit type casting:

Ex: Converting the value of 'c' to ASCII before performing the actual addition operation.

```

#include<stdio.h>
main()
{
    int i=15,sum;
    char c='c';
    sum=i+c;
    printf("%d",sum);
}

```

Q: EXPLAIN ABOUT DECISION CONTROL STATEMENTS?

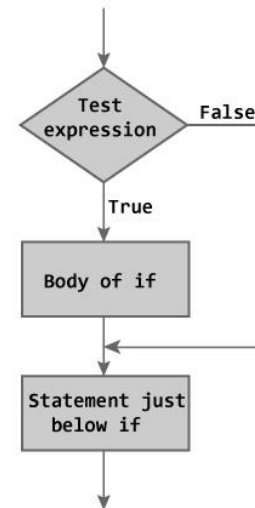
- ✓ In decision **control** statements group of statements are executed when condition is true. If condition is false, then else part statements are executed.
- ✓ There are 4 types of **decision making** control statements in C language. They are,
 1. if statements
 2. if else statements
 3. nested if statements
 4. else if else ladder.

a. Simple if statement:

If the Boolean condition evaluates to **true**, then the block of code inside the 'if' statement will be executed. If the Boolean expression evaluates to **false**, then the first set of code after the end of the 'if' statement (after the closing curly brace) will be executed.

Syn: if(condition)
{
 Statements;
}

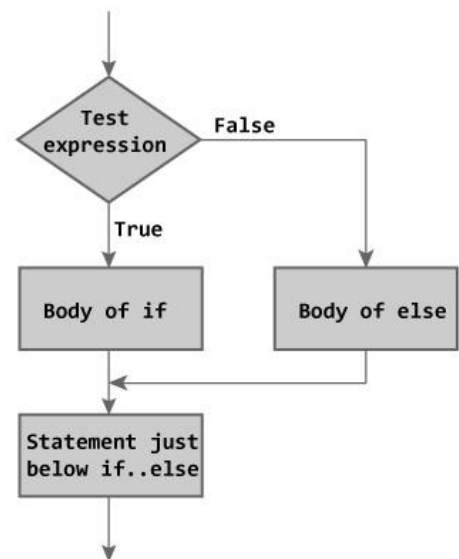
Ex:
int main()
{
 int m=40,n=40;
 if (m == n)
 {
 printf("m and n are equal");
 }
}



b. If..else statement:

In C if else control statement, group of statements are executed when condition is true. If condition is false, then else part statements are executed.

Syn: if (condition)
{
 Statement1;
 Statement2;
}
else
{
 Statement3;
 Statement4;



```

    }
Ex:  #include <stdio.h>
int main()
{
    int m=40,n=20;
    if (m == n)
    {
        printf("m and n are equal");
    }
    else
    {
        printf("m and n are not equal");
    }
}

```

c. Nested if-else:

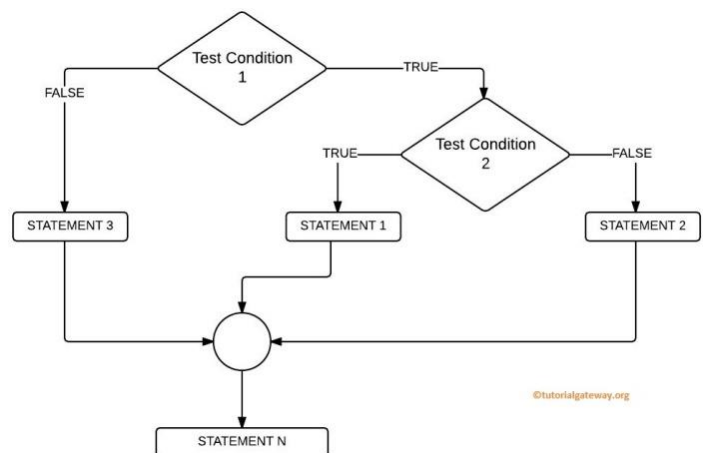
- In “nested if” control statement, if condition 1 is false, then condition 2 is checked and statements are executed if it is true.
- If condition 2 also gets failure, then else part is executed.

Syn: if (condition1)

```

{
    Statement1;
}
elseif (condition2)
{
    Statement2;
}
else
    Statement 3;

```



Ex:

```

#include <stdio.h>
int main()

```

```

{
    int m=40,n=20;
    if (m>n) {
        printf("m is greater than n");
    }
    else if(m<n) {
        printf("m is less than n");
    }
    else
    {
        printf("m is equal to n");
    }
}

```

If else Ladder Statement in C Programming

The if else ladder statement in C programming language is used to test set of conditions in sequence. If any of the conditional expression evaluates to true, then it will execute the corresponding code block and exits whole if-else ladder.

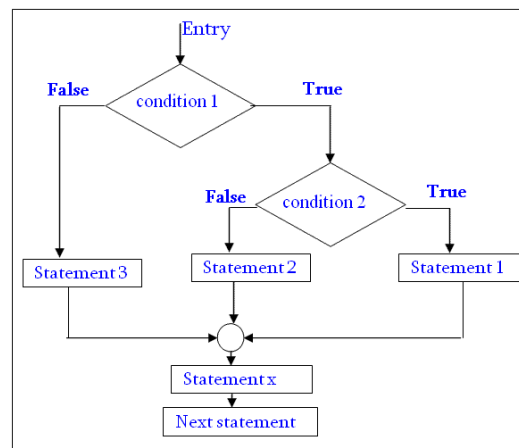
An if condition is tested only when all previous if conditions in if-else ladder is false.

Syntax:

```

if(condition1)
{
    statement1;
}
elseif (condition2)
{
    statement2;
}
else if (condition3)
{
    statement3;
}
else
{
    statement4;
}

```



- First of all condition1 is tested and if it is true then statement1 will be executed and control comes out of whole if else ladder.

- If condition1 is false then only condition2 is tested. Control will keep on flowing downward, If none of the conditional expression is true.
- The last else is the default block of code which will gets executed if none of the conditional expression is true.

C Program to print grade of a student using If Else Ladder Statement

```
#include<stdio.h>
#include<conio.h>
void main()
    {
int marks;
printf("Enter your marks between 0-100\n");
scanf("%d", &marks);
if(marks >= 90)
{
printf("YOUR GRADE : A\n");
}
else if (marks >= 70 && marks < 90)
{
printf("YOUR GRADE : B\n");
}
else if (marks >= 50 && marks < 70)
{
printf("YOUR GRADE : C\n");
}
else
{
printf("YOUR GRADE : Failed\n");
}
getch();
}
```


Q: C – Looping control statements?

A loop statement allows us to execute a statement or group of statements multiple times until the given condition is true. Control comes out of the loop statements once condition becomes false.

Types of loop control statements in C:

There are 3 types of loop control statements in C language. They are,

1. for
2. while
3. do-while

- Syntax for each C loop control statements are given in below table with description.

1. For loop:

- A **for** loop is a repetition control structure that allows you to efficiently write a loop that needs to be executed a specific number of times.
- A **for** loop is useful when you know how many times a task is to be repeated.

Syntax: for(initialization; Boolean_expression; update)

```
{  
  // Statements  
}
```

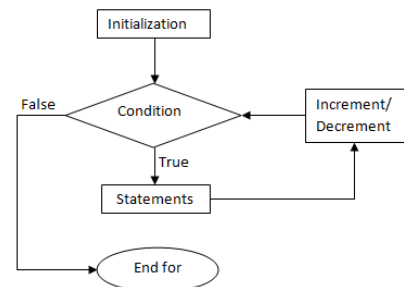


fig: Flowchart for for loop

- The **initialization** step is executed first, and only once. This step allows you to declare and initialize any loop control variables.
- Next, the **Boolean expression** is evaluated. If it is true, the body of the loop is executed. If it is false, the body of the loop will not be executed and control jumps to the next statement after the for loop.
- After the **body** of the for loop gets executed, the control jumps back up to the update(inc/dec) statement. This statement allows you to update any loop control variables.
- The Boolean expression is now evaluated again. If it is true, the loop executes and the process repeats. After the Boolean expression is false, the for loop terminates.

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

```

int i;
for(i=0;i<10;i++)
{
printf("%d ",i);
}
}

```

2. While loop:

The **while** statement continually executes a block of statements while a particular condition is **true**.

✓ It is called as entry level loop control structure.

Syntax: while (condition)

```

{
    statement(s);
}

```

The while statement evaluates condition, which must return a boolean value. If the condition evaluates to true, the while statement executes the statement(s) in the while block. The while statement continues testing the condition and executing its block until the condition evaluates to false.

Ex:

```

#include <stdio.h>
int main()
{
    int i=3;
    while(i<10)
    {
        printf("%d\n",i);
        i++;
    }
}

```

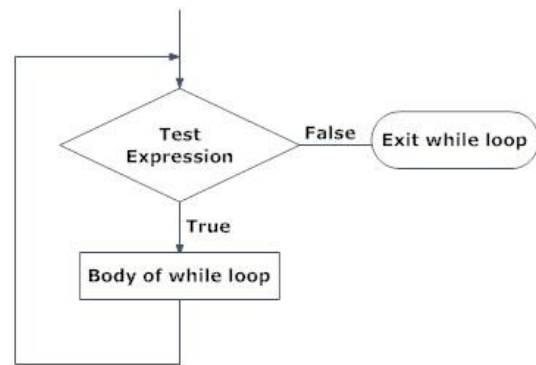


Figure: Flowchart of while loop

2. Do while loop:

- ✓ In do..while loop control statement, while loop is executed irrespective of the condition for first time. Then 2nd time onwards, loop is executed until condition becomes false.
- ✓ It called as exit level loop control structure.

Syn: do {
statements;
}
while (condition);

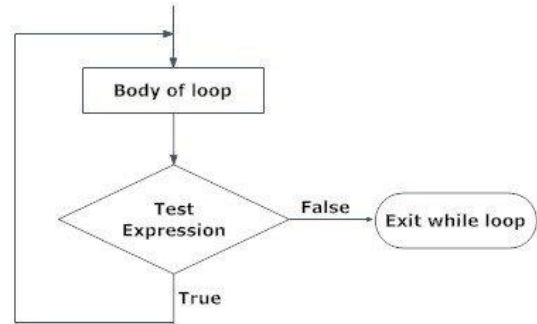


Figure: Flowchart of do...while loop

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

int main()
{
int i=1;
do
{
printf("Value of i is %d\n",i);
i++;
}while(i<=4 && i>=2);
}
```

Difference between while & do while loops in C:

S.no	while	do while
1	Loop is executed only when condition is true.	Loop is executed for first time irrespective of the condition. After executing while loop for first time, then condition is checked.
2	Entry level loop control structure	Exit level loop control structure.

C – Case control statements:

The **statements** which are used to execute only specific block of statements in a series of blocks are called **case** control statements. That is Switch case;

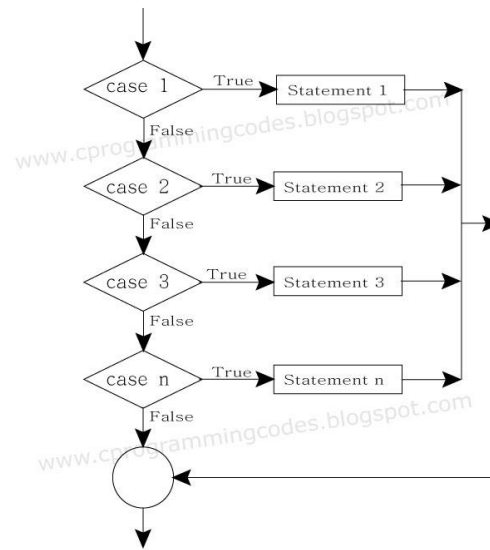
Switch case statement in C:

- ✓ This case statement is a multi-way decision statement which is a simpler version of the if-else block which evaluates only one variable.

- ✓ Switch case statements are used to execute only specific case statements based on the switch expression.
- ✓ A **switch** statement allows a variable to be tested for equality against a list of values. Each value is called a case.
- ✓ When the variable being switched on is equal to a case, the statements following that case will execute until a **break** statement is reached.
- ✓ When a **break** statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.
- ✓ Not every case needs to contain a **break**. If no **break** appears, the flow of control will fall through to subsequent cases until a break is reached.
- ✓ A **switch** statement can have an optional **default** case, which must appear at the end of the switch. The default case can be used for performing a task when none of the cases is true. No **break** is needed in the default case.

Syntax:

```
switch (expression)
{
    case label1:  statements;
                break;
    case label2:  statements;
                break;
    default:     statements;
                break;
}
```



Example program for switch..case statement in C:

```
#include <stdio.h>
void main ()
{
    int value = 3;
    switch(value)
    {
        case 1: printf("Value is 1 \n" );
                break;
```

```

        case 3:printf("Value is 3 \n" );
                break;
        default:printf("Value is other than 1,2,3,4 \n" );
    }
}

```

Jumping statements:

1. break
2. continue
3. goto

Break statement in C:

The **break** statement in C programming has the following two usages –

- ✓ When a **break** statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop.
- ✓ It can be used to terminate a case in the **switch** statement (covered in the next chapter).

Syntax: break;

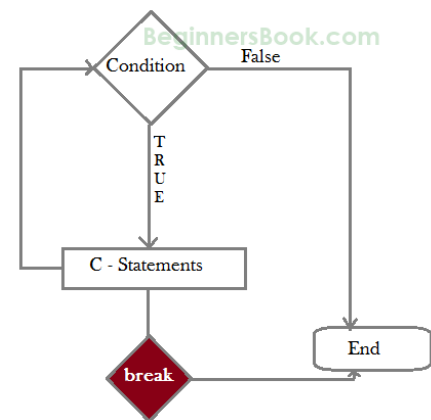
Example program for break statement in C:

```

#include <stdio.h>

int main()
{
    int i;
    for(i=0;i<10;i++)
    {
        if(i==5)
        {
            printf("\nComing out of for loop when i = 5");
            break;
        }
        printf("%d ",i);
    }
}

```



Continue statement in C:

- ✓ Continue statement is used to continue the next iteration. So, the remaining statements are skipped within the loop for that particular iteration.

Syntax: continue;

Example program for continue statement in C:

```
#include <stdio.h>
int main()
{
    int i;
    for(i=0;i<10;i++)
        {
            if(i==5)
            {
                printf("\nSkipping %d from display ",i);
                continue;
            }
            printf("%d ",i);
        }
}
```

4. goto statement in C:

- ✓ goto statements is used to transfer the normal flow of a program to the specified label in the program.
- ✓ Below is the syntax for goto statement in C.

```
{
    .....
    go to label;
    .....
    LABEL:
    statements;
}
```

Example program for goto statement in C:

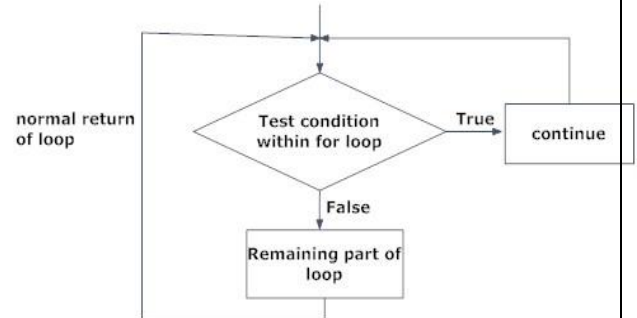
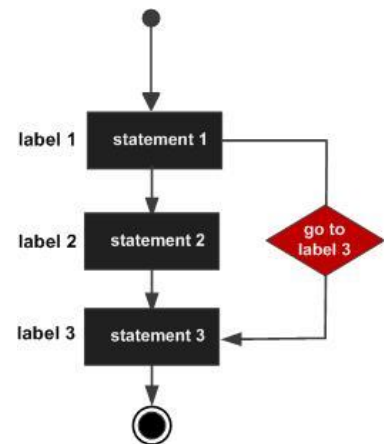


Fig: Flowchart of continue statement



```
#include <stdio.h>
int main()
{
int i;
for(i=0;i<10;i++)
{
    if(i==5)
    {
        printf("\nWe are using goto statement when i = 5");
        goto HAI;
    }
printf("%d ",i);
}
HAI :printf("\nNow, we are inside label name \"hai\" \n");
}
```

UNIT III Arrays:

Introduction – Declaration of Arrays – Accessing elements of the Array – Storing Values in Array– Operations on Arrays – one dimensional, two dimensional and multi dimensional arrays, character handling and strings.

Q: explain about C – Arrays?

- ✓ Array is a collection of variables belonging to the same data type. You can store group of data of same data type in an array.
- ✓ Array might be belonging to any of the **data types**.
- ✓ Array size must be a constant value.
- ✓ Always, Contiguous (adjacent) memory locations are used to store array elements in memory.

The Various types of Array those are provided by c as Follows:-

1. Single Dimensional Array
2. Two Dimensional Array
3. Multi Dimensional array

1. Single dimensional array:

- ✓ An array variable with one dimension is called single dimensional array.
- ✓ So there is only the single bracket then it called the Single Dimensional Array.
- ✓ The [] is used for dimensional or the sub-script of the array. A dimensional is used representing the elements of the array.
- ✓ Array indexes are start at zero and stop one less than the array size

Ex: int a[5]

Array variables are a[0],a[1],a[2],a[3],a[4]

- ✓ For Accessing the Element from the array we can use the array variable with index value of the Array.

a[2]=23;

2) Two Dimensional Array or the Matrix

- ✓ The Two Dimensional array is used for representing the elements of the array in the form of the rows and columns and these are used for representing the Matrix .
- ✓ Two Dimensional Array uses the two subscripts for declaring the elements of the Array

Ex: int a[3][2]

In this first 3 represents the total number of Rows and the Second 2 Represents the Total number of Columns.

The Total Number of elements are Multiplying the Numbers of Rows * Number of Columns in The Array in the above array the Total Number of elements are 6

3) Multidimensional Array:

- ✓ Multi-dimensional arrays are declared by providing more than one set of square [] brackets after the variable name in the declaration statement.
- ✓ Multi-dimensional arrays do not require **the first** dimension to be given if the array is to be completely initialized. All dimensions after the first must be given in any case.
- ✓ An array with more than two dimensions is called as multidimensional array.

Ex: `int a[3][2][2]`

Declaring Arrays

Array variables are declared identically to variables of their data type, except that the variable name is followed by one pair of square [] brackets for each dimension of the array.

single dimensionalarray:

syn: `data-type arr_name[array_size];`

Ex: `int age [5];`

Double dimensionalarray:

Syn: `data-type arr_name[no of rows][no of columns];`

Ex: `int age [5][3];`

Array initialization:

- ✓ Arrays may be initialized when they are declared, just as any other variables.
- ✓ Place the initialization data in curly {} braces following the equals sign.
- ✓ If an array is to be completely initialized, the dimension of the array is not required. The compiler will automatically size the array to fit the initialized data.

single dimensionalarray:

syn: `data_typearr_name [arr_size]={value1, value2, value3,...};`

ex: `int age[5]={0, 1, 2, 3, 4, 5};`

Double dimensionalarray:

Syn: data-type arr_name[no of rows][no of cols]={ value1, value2, value3,...};

Ex: int age[2][3]={0, 1, 2, 3, 4, 5};

(or)

int age[2][3]={{2,3,4},{5,6,7}}

Accessing array:

Elements of an array are accessed by specifying the index of the desired element within square [] brackets after the array name.

Syn: arr_name[index];

age[0]; /*first element is accessed*/

age[1]; /*second element is accessed*/

Ex: Single dimensional array

Program for accessing each variable

```
#include<stdio.h>
void main()
{
int i;
int arr[5] = {10,20,30,40,50};
for (i=0;i<5;i++)
    {
        printf("value of arr[%d] is %d \n", i, arr[i]);
    }
}
```

Ex: Double dimensional array

This c program for addition of two matrices.

```
#include <stdio.h>
void main()
{
int r, c, a[10][10], b[10][10], sum[10][10], i, j;
printf("\nEnter elements of 1st matrix:\n");
for(i=0; i<2; ++i)
    for(j=0; j<5; ++j)
```

```

        {
            printf("Enter element a%d%d: ",i+1,j+1);
            scanf("%d",&a[i][j]);
        }
printf("Enter elements of 2nd matrix:\n");
for(i=0; i<2; ++i)
    for(j=0; j<5; ++j)
        {
            printf("Enter element a%d%d: ",i+1, j+1);
            scanf("    %d", &b[i][j]);
        }
// Adding Two matrices
for(i=0;i<2;++i)
    for(j=0;j<5;++j)
        {
            sum[i][j]=a[i][j]+b[i][j];
        }
// Displaying the result
printf("\nSum of two matrix is: \n\n");
for(i=0;i<2;++i)
    for(j=0;j<5;++j)
        {
            printf("%d  ",sum[i][j]);
        }
printf("\n");
}
}
}

```

Q: explain about String's in C?

- ✓ C Strings are nothing but array of characters ended with null character ('\0').

- ✓ This null character indicates the end of the string.
- ✓ Strings are always enclosed by double quotes. Whereas, character is enclosed by single quotes **Example for C string:**

```
char string[20] = { 'f', 'r', 'e', 's', 'h', '2', 'r', 'e', 'f', 'r', 'e', 's', 'h', '\0' };
                (or)
```

```
char string[20] = "fresh2refresh";           (or)
```

```
char string [] = "fresh2refresh";
```

- ✓ Difference between above declarations are, when we declare char as "string[20]", 20 bytes of memory space is allocated for holding the string value.
- ✓ When we declare char as "string[]", memory space will be allocated as per the requirement during execution of the program.

Example program for C string:

```
#include <stdio.h>

int main ()
{
    char string[20] = "fresh2refresh.com";
    printf("The string is : %s \n", string );
    return 0;
}
```

C String functions:

String.h header file supports all the **string functions** in C language. All the string functions are given below.

strcat():

Strcat() function in C language concatenates two given strings. It concatenates source string at the end of destination string.

Syntax: strcat(str2, str1);

str2 is concatenated at the end of str1.

Example:

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

```

char source[ ] = " GCSR" ;
char target[ ]= " College" ;
printf( "\nSource string = %s", source ) ;
printf( "\nTarget string = %s", target ) ;
strcat( target, source ) ;
printf( "\nTarget string after strcat() = %s", target ) ;
}

```

strcpy():

strcpy() function copies contents of one string into another string.

Syntax: strcpy(str1, str2);

It copies contents of str1 into str2.

If destination string length is less than source string, entire source string value won't be copied into destination string will be truncated.

Ex: strcpy (target, source) ;

strlen():

strlen() function counts the number of characters in a given string and returns the integer value. It stops counting the character when null character is found. Because, null character indicates the end of the string in C.

Ex: len = strlen(array) ;

strcmp():

This function in C compares two given strings and returns zero if they are same.

If length of string1 < string2, it returns < 0 value.

If length of string1 > string2, it returns > 0 value.

Syntax for strcmp) function is given below.

strcmp (str1, str2) ;

strcmp() function is case sensitiv. i.e, "A" and "a" are treated as different characters.

strcmpi():

This function in C is same as strcmp() function. But, strcmpi() function is not case sensitive. i.e, "A" and "a" are treated as same characters. Where as, strcmp() function treats "A" and "a" as different characters.

```
strcmpi (str1, str2 );
```

strlwr():

strlwr() function converts a given string into lowercase.

```
strlwr(string);
```

strupr():

strupr() function converts a given string into uppercase.

```
strupr(string);
```

strrev():

strrev() function reverses a given string in C language.

```
strrev(string);
```

Ex: Program for handling strings by using string handling functions in c:

```
#include <stdio.h>
#include <string.h>
int main()
{
char source[ ] = " GCSR" ;
char target[ ]= " College" ;
int c,l;
char s1[10],s2[10];
printf( "\nSource string = %s", source ) ;
printf( "\nTarget string = %s", target ) ;
strcat( target, source ) ;
printf( "\nTarget string after strcat() = %s", target ) ;
c=strcmpi (target, source);
printf( "\n the comparisions strings are %d",c);
l= strcat( target);
printf( "\n the length of target string are %d",l);
```

```
s1=strlow(source);
printf( "\n The lower case string is",s);
s2=strupr(target);
printf( "\n The upper case string is",s2);
strrev(soure);
printf( "\n The reverse Source string = %s", source ) ;
getch()
}
}
```

UNIT IV Functions:

Introduction – using functions – Function declaration/ prototype – Function definition – function call – return statement – Passing parameters – Scope of variables – Storage Classes – Recursive functions. Structure, Union, and Enumerated Data Types: Introduction – Nested Structures – Arrays of Structures – Structures and Functions– Union – Arrays of Unions Variables – Unions inside Structures – Enumerated Data Types.

Q: What is C function?

- A function is a group of statements that together perform a task. Every C program has at least one function, which is **main()**.
- The core concept of C functions are, re-**usability**, dividing a big task into small pieces to achieve the functionality and to improve understandability of very large C programs.
- A large C program is divided into basic building blocks called C function. C function contains set of instructions enclosed by “{ }” which performs specific operation in a C program.
- Actually, Collection of these functions creates a C program.
- Functions are used to avoid rewriting same logic/code again and again in a program.
- There is no limit in calling C functions to make use of same functionality wherever required.
- We can call functions any number of times in a program and from any place in a program.

C function declaration, function call and function definition:

There are 3 aspects in each C function. They are,

- A. Function declaration or prototype - This informs compiler about the function name, function parameters and return value's data type.

Syn:return_typefunction_name(arguments list)

```
{  
    Body of function;  
}
```

- B. Function call – This calls the actual function

Syn:function_name(arguments list);

c. Function definition – This contains all the statements to be executed.

Syn:return_typefunction_name(argument list);

Ex:

```
#include<stdio.h>

float square ( float x);           // function prototype, also called function
declaration.

int main( )                       // main function, program starts from here
{
float m, n ;
printf( "\nEnter some number for finding square \n");
scanf( "%f", &m );                // function call
n = square ( m );
printf( "\nSquare of the given number %f is %f",m,n );
}

float square ( float x )          // function definition
{
float p ;
p = x * x ;
return ( p );
}
```

Q: How to call C functions in a program?

There are two ways that a C function can be called from a program. They are,

1. Call by value
2. Call by reference

1. Call by value:

- ✓ In call by value method, the value of the variable is passed to the function as parameter.
- ✓ The value of the actual parameter cannot be modified by formal parameter.
- ✓ Different Memory is allocated for both actual and formal parameters. Because, value of actual parameter is copied to formal parameter.

Note:

- Actual parameter – This is the argument which is used in function call.

- Formal parameter – This is the argument which is used in function definition

Example:

```
#include<stdio.h>
void swap(int a, int b);
int main()
{
int m = 22, n = 44;
printf(" values before swap m = %d \nand n = %d", m, n);
swap(m, n);
}
void swap(int a, int b)
{
int tmp;
tmp = a;
a = b;
b = tmp;
printf(" \nvalues after swap m = %d\n and n = %d", a, b);
}
```

2. Call by reference:

- ✓ In call by reference method, the address of the variable is passed to the function as parameter.
- ✓ The value of the actual parameter can be modified by formal parameter.
- ✓ Same memory is used for both actual and formal parameters since only address is used by both parameters.

Ex:

```
#include<stdio.h>
void swap(int *a, int *b);
int main()
{
int m = 22, n = 44;
printf("values before swap m = %d \n and n = %d",m,n);
```

```

swap(&m, &n);
}
void swap(int *a, int *b)
{
int tmp;
tmp = *a;
*a = *b;
*b = tmp;
printf("\n values after swap a = %d \nand b = %d", *a, *b);
}

```

Q: Explain about Passing parameters to functions?

All C functions can be called either with arguments or without arguments in a C program. These functions may or may not return values to the calling function.

- i. C function with arguments (parameters) and with return value
- ii. C function with arguments (parameters) and without return value
- iii. C function without arguments (parameters) and without return value
- iv. C function without arguments (parameters) and with return value

1. Function with arguments & with return value:

Syn:

```

int function ( int );    // function declaration
function ( a );         // function call
int function( int a )   // function definition
{
statements;
return a;
}

```

Ex:

```

#include<stdio.h>
int sum();
void main()
{
int addition;

```

```

addition = sum(12,15);
printf("\nSum of two given values = %d", addition);
}
int sum(int x,int y )
{
int a = x, b = y, sum;
sum = a + b;
return sum;
}

```

2. Function with arguments & without return value:

Syn:

```

void function ( int ); // function declaration
function( a ); // function call
void function( int a ) // function definition
{
statements;
}

```

3. Function without arguments & without return value:

Syn:

```

void function ( ); // function declaration
function( ); // function call
void function( ) // function definition
{
statements;
}

```

4. Function without arguments & with return value:

Syn:

```

int function ( ); // function declaration
function ( ); // function call
int function() // function definition

```

```
{  
statements;  
return a;  
}
```

Return statement in c:

- ✓ Only one value can be returned from a function.
- ✓ If you try to return **more** than one values from a function, only one value will be returned that appears at the right most place of the return statement.
- ✓ For example, if you use “return a,b,c” in your function, value for c only will be returned and values a, b won’t be returned to the program.

Types of functions : There are 2 types of functions in C. They are, library functions and user defined functions.

1. C – Library functions

- Library functions in C language are inbuilt functions which are grouped together and placed in a common place called library.
- These library functions are created by the persons who designed and created C compilers.
- All C **standard library** functions are declared in many header files which are saved as file_name.h.
- We are including these header files in our C program using “#include<file_name.h>” command to make use of the functions those are declared in the header files.

2. User defined functions:

User defined functions are the functions which are written by us for our own requirement.

Adding user defined functions in C library:

- ✓ It is possible to add, delete, modify and access our own user defined function to C library.
- ✓ The advantage of adding user defined function in C library is, this function will be available for all C programs once added to the C library.

Q: Explain about C - variable Scope Rules:?

A scope in any programming is a region of the program where a defined variable can have its existence and beyond that variable it cannot be accessed. There are three places where variables can be declared in C programming language –

- Inside a function or a block which is called **local** variables.
- Outside of all functions which is called **global** variables.
- In the definition of function parameters which are called **formal** parameters.
- Let us understand what are **local** and **global** variables, and **formal** parameters.

Local Variables:

- ✓ Variables that are declared inside a function or block are called local variables.
- ✓ They can be used only by statements that are inside that function or block of code.
- ✓ Local variables are not known to functions outside their own.

```
#include <stdio.h>

int main ()
{
    int a, b;
    int c;
    a = 10;
    b = 20;
    c = a + b;
    printf ("value of a = %d, b = %d and c = %d\n", a, b, c);
    return 0;
}
```

Global Variables:

- ✓ Global variables are defined outside a function, usually on top of the program.
- ✓ Global variables hold their values throughout the lifetime of your program and they can be accessed inside any of the functions defined for the program.

```
#include <stdio.h>

int g;

int main ()
{
    int a, b;
```

```

a = 10;
b = 20;
g = a + b;
printf ("value of a = %d, b = %d and g = %d\n", a, b, g);
return 0;
}

```

Note: A program can have same name for local and global variables but the value of local variable inside a function will take preference.

```

#include <stdio.h>
int g = 20;
int main ()
{
int g = 10;
printf ("value of g = %d\n", g);
return 0;
}

```

Q: Explain about C – Storage Classes?

Storage class specifiers in C language tells the compiler where to store a variable, how to store the variable, what is the initial value of the variable and life time of the variable.

Syntax: storage_specifier data_type variable_name

There are 4 storage class specifiers available in C language. They are,

1. auto
2. extern
3. static
4. register

S.No.	Storage Specifier	Storage place	Initial / default value	Scope	Life
1	auto	CPU Memory	Garbage value	local	Within the function only.
2	extern	CPU memory	Zero	Global	Till the end of the main program. Variable definition might be anywhere in the C program

3	static	CPU memory	Zero	local	Retains the value of the variable between different function calls.
4	register	Register memory	Garbage value	local	Within the function

Note:

- ✓ For faster access of a variable, it is better to go for register specifiers rather than auto specifiers.
- ✓ Because, register variables are stored in register memory whereas auto variables are stored in main CPU memory.

Example:

```
#include<stdio.h>
void increment(void);
int main()
{
increment();
increment();
return 0;
}
void increment(void)
{
static int i = 0 ;
printf( "%d ", i );
i++;
}
```

Q: Explain about Recursion in C?

- ✓ Recursion is the process of repeating items in a self-similar way.
- ✓ In programming languages, if a program allows you to call a function inside the same function, then it is called a recursive call of the function.
- ✓ The C programming language supports recursion, i.e., a function to call itself. But while using recursion need to be careful to define an exit condition from the function, otherwise it will go into an infinite loop.

Example: calculates the factorial of a given number using a recursive function –


```

#include <stdio.h>
int factorial(unsigned int i)
{
if(i<= 1)
{
return 1;
}
return i * factorial(i - 1);
}
int main()
{
int i = 15;
printf("Factorial of %d is %d\n", i, factorial(i));
return 0;
}

```

Example generates the Fibonacci series for a given number using a recursive function –

```

#include <stdio.h>
int fibonacci(int i)
{
if(i == 0)
{
return 0;
}
if(i == 1)
{
return 1;
}
return fibonacci(i-1) + fibonacci(i-2);
}
int main()

```

```
{
int i;
for (i = 0; i < 10; i++) {
printf("%d\t\n", fibonacci(i));
}
return 0;
}
```

Difference between Recursion and Iteration:

1. In recursion, function **call itself** until the base condition is reached.
On other hand iteration means **repetition of process** until the condition fails.
2. Iterative approach involves four steps, initialization , condition, execution and updation.
In recursive function, only base condition (terminate condition) is specified.
3. Recursion is slower than iteration due to overhead of maintaining stack whereas iteration is faster.
4. Recursion takes more memory than iteration due to overhead of maintaining stack.

Q: Explain about C – Structures?

- A normal C variable can hold only one data of one data type at a time.
- An array can hold group of data of same data type.
- A structure can hold group of data of different data types which are grouped together and each element in a C structure is called member.
- If you want to access structure members in C, structure variable should be declared.
- Many structure variables can be declared for same structure and memory will be allocated for each separately.

Difference between C variable, C array and C structure:

- Data types can be int, char, float, double and **long double** etc.

Datatype	C variable		C array		C structure	
	Syntax	Example	Syntax	Example	Syntax	Example
int	int a	a = 20	int a[3]	a[0] = 10 a[1] = 20 a[2] = 30 a[3] = '\0'	struct student { int a; char b[10]; }	a = 10 b = "Hello"
char	char b	b='Z'	char b[10]	b="Hello"		

Declare a C structure:

Syntax:

```
struct tag_name
{
data type var_name1;
data type var_name2;
data type var_name3;
};
```

Ex:

```
struct student
{
int mark;
char name[10];
float average;
};
```

Declaring structure variable:

Ex: struct student report;

Initializing structure variable:

Ex: struct student report = {100, "Mani", 99.5};

Accessing structure members:

Ex: report.mark
report.name
report.average

initialize a C structure:

1. How to access the members of a C structure?

Example program for C structure:

This program is used to store and access “id, name and percentage” for one student.

```
#include <stdio.h>
#include <string.h>
struct student
{
int id;
char name[20];
float percentage;
};
int main()
{
struct student record = {0}; //Initializing to null
record.id=1;
strcpy(record.name, "Raju");
record.percentage = 86.5;
printf(" Id is: %d \n", record.id);
printf(" Name is: %s \n", record.name);
printf(" Percentage is: %f \n", record.percentage);
return 0;
}
```

C – Array of Structures

C Structure is collection of different data types which are grouped together. Whereas, array of structures is nothing but collection of structures. This is also called as structure array in C.

Example program for array of structures in C:

This program is used to **store** and access “id, name and percentage” for 3 students.

```
struct student
{
int id;
```

```
char name[30];
float percentage;
};
struct student record[2];
```

C – Nested Structure

Nested structure in C is nothing but structure within structure. One structure can be declared inside other structure as we declare structure members inside a structure. The structure variables can be a normal structure variable or a pointer variable to access the data.

Q: Explain about C – Union?

- C Union is also like **structure**, i.e. collection of different **data types** which are grouped together. Each element in a union is called member.
- Union and structure in C are same in concepts, except allocating memory for their members.
- Structure allocates storage space for all its members separately.
- Whereas, Union allocates one common storage space for all its members
- We can access only one member of union at a time. We can't access all member values at the same time in union. But, structure can access all member values at the same time. This is because Union allocates one common storage space for all its members. Whereas Structure allocates storage space for all its members separately.
- Many union variables can be created in a program and memory will be allocated for each union variable separately.

Declare a union:

Syntax:

```
union tag_name
{
    data type var_name1;
    data type var_name2;
    data type var_name3;
};
```

Example:

```
union student
{
```

```
int mark;  
char name[10];  
float average;  
};
```

Declaring union variable: union student report;

Initializing union variable: union student report = {100, "Mani", 99.5};

Accessing union members:

```
report.mark;  
report.name;  
report.average;
```

Example program for C union:

```
#include <stdio.h>  
#include <string.h>  
union student  
{  
char name[20];  
char subject[20];  
float percentage;  
};  
int main()  
{  
union student record1;  
strcpy(record1.name, "Raju");  
strcpy(record1.subject, "Maths");  
record1.percentage = 86.50;  
printf("Union record1 values example\n");  
printf(" Name    : %s \n", record1.name);  
printf(" Subject  : %s \n", record1.subject);  
printf(" Percentage : %f \n\n", record1.percentage);  
}
```

Difference between structure and union in C:

S.no	C Structure	C Union
1	Structure allocates storage space for all its members separately.	Union allocates one common storage space for all its members. Union finds that which of its member needs high storage space over other members and allocates that much space
2	Structure occupies higher memory space.	Union occupies lower memory space over structure.
3	We can access all members of structure at a time.	We can access only one member of union at a time.
4	Structure example: struct student { int mark; char name[6]; double average; };	Union example: union student { int mark; char name[6]; double average; };
5	For above structure, memory allocation will be like below. int mark - 2B char name[6] - 6B double average - 8B Total memory allocation = $2+6+8 = 16$ Bytes	For above union, only 8 bytes of memory will be allocated since double data type will occupy maximum space of memory over other data types. Total memory allocation = 8 Bytes

Q: Explain about Enumeration data type in C?

- ✓ Enumeration data type consists of named integer constants as a list.
- ✓ It start with zero by default and value is incremented by 1 for the sequential identifiers in the list.

Syntax : enum identifier
 {
 enumerator-list
 };

Example: enum month { Jan, Feb, Mar }; or
/* Jan, Feb and Mar variables will be assigned to 0, 1 and 2 respectively by default */
enum month { Jan = 1, Feb, Mar };
/* Feb and Mar variables will be assigned to 2 and 3 respectively by default */
enum month { Jan = 20, Feb, Mar };
/* Jan is assigned to 20. Feb and Mar variables will be assigned to 21 and 22
respectively by default */
#include <stdio.h>

int main()
{
enum MONTH { Jan = 0, Feb, Mar };
enum MONTH month = Mar;
if(month == 0)
printf("Value of Jan");
else if(month == 1)
printf("Month is Feb");
if(month == 2)
printf("Month is Mar");
}

UNIT V Pointers:

Understanding Computer Memory – Introduction to Pointers – declaring Pointer Variables – Pointer Expressions and Pointer Arithmetic – Null Pointers - Passing Arguments to Functions using Pointer – Pointer and Arrays – Memory Allocation in C Programs – Memory Usage – Dynamic Memory Allocation – Drawbacks of Pointers
Files: Introduction to Files – Using Files in C – Reading Data from Files – Writing Data to Files – Detecting the End-of-file – Error Handling during File Operations – Accepting Command Line Arguments.

Q: Explain about C – Pointers?

C Pointer is a variable that stores/points the address of another variable. C Pointer is used to allocate memory dynamically i.e. at run time. The pointer variable might be belonging to any of the data type such as int, float, char, double, short etc.

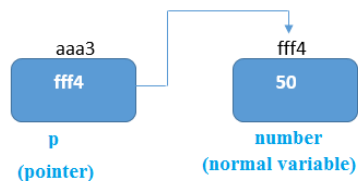
Syntax: data_type *var_name;

Example: int *p;
char *p;

Where, * is used to denote that “p” is pointer variable and not a normal variable.

Example: int n = 50;

`int* p = &n;` // Variable p of type pointer is pointing to the address of the variable n of type integer



Key points to remember about pointers in C:

- Normal variable stores the value whereas pointer variable stores the address of the variable.
- The content of the C pointer always be a whole number i.e. address.
- &symbol is used to get the address of the variable.
- Symbol is used to get the value of the variable that the pointer is pointing to.
- Two pointers can be subtracted to know how many elements are available between these two pointers.
- But, Pointer addition, multiplication, division are not allowed.
- The size of any pointer is 2 byte (for 16 bitcompiler).

Example program for pointer in C:

```
#include<stdio.h>
int main()
{
int number=50;
int *p;
p=&number;//stores the address of number variable
printf("Address of p variable is %x \n",p);
printf("Value of p variable is %d \n",*p);
return 0;
}
```

Output

```
Address of number variable is fff4
Address of p variable is fff4
Value of p variable is 50
```

Types of pointers: pointers are classified into

NULL Pointer

A pointer that is not assigned any value but NULL is known as the NULL pointer. If you don't have any address to be specified in the pointer at the time of declaration, you can assign NULL value.

```
int *p=NULL;
```

Void Pointer

It is a pointer that has no associated data type with it. A void pointer can hold addresses of any type and can be typecast to any type.

It is also called a **generic pointer** and does not have any standard data type.

It is created by using the keyword void.

Ex: include <stdio.h>

```
int main(){
    void *p = NULL; //void pointer
```

```
printf("The size of pointer is:%d\n",sizeof(p)); //size of p depends on compiler
return 0;
}
```

Wild Pointer

Wild pointers are also called uninitialized pointers. Because they point to some arbitrary memory location and may cause a program to crash or behave badly.

Because they may point to some unknown memory location which may cause problems in our program. This may lead to the crashing of the program.

It is advised to be cautious while working with wild pointers.

Example

Following is the C program for the wild pointer –

```
#include <stdio.h>
int main() {
    int *p; //wild pointer
    printf("\n%d",*p);return 0;
}
```

Dangling Pointers in C

Sometimes the programmer fails to initialize the pointer with a valid address, then this type of initialized pointer is known as a dangling pointer in C.

Dangling pointer occurs at the time of the object destruction when the object is deleted or de-allocated from memory without modifying the value of the pointer.

Q: Explain about Pointer Expressions and Pointer Arithmetic?

The following arithmetic operations are allowed to use on the pointer type operands:

- Incrementing a pointer
- Decrementing a pointer
- Adding an integer to pointer
- Subtracting an integer to pointer

1) Incrementing a pointer :

Each time a pointer is incremented its points to the memory location of the next element

Example: suppose ptr=1002, after the execution of the following statement

```
Ptr++;
```

The pointer value is incremented by 1, that is, it becomes 1004, rather than 1002 since size of int is 2 bytes, the next address base type is 1004.

2) **Decrementing a pointer :**

Each time a pointer is decremented its points to the memory location of the previous element .

Example: suppose ptr=1002, after the execution of the following statement

Ptr--;

The pointer value is decreased by 1, that is, it becomes 1000, rather than 1002 since size of int is 2 bytes, the next address base type is 1004.

3) **Adding an integer to pointer:**

Let ptr=1002.

Consider the statement

Ptr=ptr+2;

In the above expression when the integer 2 is added to the pointer variable ptr and the pointer value becomes 1006, rather than 1004. since the size of the int is 2 the pointer ptr value is incremented by $2 * \text{sizeof(int)} = 2 * 2 = 4$ that is the value of ptr is $= 1002 + 4 = 1006$

4) **subtracting an integer to a pointer :**

Let ptr=1004.

Consider the statement

Ptr=ptr-2;

In the above expression when the integer 2 is subtracted from the pointer variable ptr and the pointer value becomes 1000, rather than 1002. since the size of the int is 2 the pointer ptr value is decremented by $2 * \text{sizeof(int)} = 2 * 2 = 4$ that is the value of ptr is $= 1004 - 4 = 1000$

Advantages of Pointers in C

- Pointers are useful for accessing memory locations.
- Pointers provide an efficient way for accessing the elements of an array structure.
- Pointers are used for dynamic memory allocation as well as deallocation.
- Pointers are used to form complex data structures such as linked list, graph, tree, etc.

Disadvantages of Pointers in C

- Pointers are a little complex to understand.
- Pointers can lead to various errors such as segmentation faults or can access a memory location which is not required at all.
- Pointers are comparatively slower than that of the variables.
- Programmers find it very difficult to work with the pointers;

Q: Explain about Dynamic memory allocation in C?

The process of allocating memory during program execution is called dynamic memory allocation.

C language offers 4 dynamic memory allocation functions. They are,

1. malloc()
2. calloc()
3. realloc()
4. free()

S.no	Function	Syntax
1	malloc ()	malloc (number *sizeof(int));
2	calloc ()	calloc (number, sizeof(int));
3	realloc ()	realloc (pointer_name, number * sizeof(int));
4	free ()	free (pointer_name);

malloc() function in C:

- ✓ malloc () function is used to allocate space in memory during the execution of the program.
- ✓ malloc () does not initialize the memory allocated during execution. It carries garbage value.
- ✓ malloc () function returns null pointer if it couldn't able to allocate requested amount of memory.

Syntax:

```
ptr = (cast-type*) malloc(byte-size)
```

For Example:

```
ptr = (int*) malloc (100 * sizeof(int));
```

Since the size of int is 4 bytes, this statement will allocate 400 bytes of memory. And, the pointer ptr holds the address of the first byte in the allocated memory.

2. calloc() function in C:

- **calloc** or **“contiguous allocation”** method in C is used to dynamically allocate the specified number of blocks of memory of the specified type.
- it is very much similar to malloc() but has two different points and these are:
- It initializes each block with a default value '0'.
- It has two parameters or arguments as compare to malloc().

Syntax:

```
ptr = (cast-type*)calloc(n, element-size);
```

here, n is the no. of elements and element-size is the size of each element.

For Example:

```
ptr = (float*) calloc(25, sizeof(float));
```

This statement allocates contiguous space in memory for 25 elements each with the size of the float.

3. realloc() function in C:

“realloc” or “re-allocation” method in C is used to dynamically change the memory allocation of a previously allocated memory.

Syntax:

```
ptr = realloc(ptr, newSize);
```

where ptr is reallocated with new size 'newSize'

- realloc () function modifies the allocated memory size by malloc() and calloc() functions to new size.
- If enough space doesn't exist in memory of current block to extend, new block is allocated for the full size of reallocation, then copies the existing data to new block and then frees the old block.

4. free() function in C:

- ✓ free () function frees the allocated memory by malloc (), calloc (), realloc () functions and returns the memory to the system.

Syntax: free(mem_allocation);

Example program for realloc() and free() functions in C:

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

int main()
{
    char *mem_allocation;      /* memory is allocated dynamically */
    mem_allocation = malloc( 20 * sizeof(char) );
    if( mem_allocation == NULL )
    {
        printf("Couldn't able to allocate requested memory\n");
    }
    else
```

```

{
strcpy( mem_allocation,"srigcsrcollege.com");
}
printf("Dynamically allocated memory content : " \"%s\n", mem_allocation );
mem_allocation=realloc(mem_allocation,100*sizeof(char));
if( mem_allocation == NULL )
{
printf("Couldn't able to allocate requested memory\n");
}
else
{
strcpy( mem_allocation,"space is extended upto " \"100 characters");
}
printf("Resized memory : %s\n", mem_allocation );
free(mem_allocation);
}

```

Q: explain about Files I/O in c?

In C language the functions scanf() and printf() are used to read and write data. These are console oriented input and output functions which always use the terminal (keyboard and screen) as the target place. These functions works fine as long as the data is small. However in many real-life problem situations involves large - volume of data. In such situations, the console oriented Input Output operations raises two problems.

- 1) When a program is terminated, the entire data is lost.
- 2) If you have to enter large number of data,it will take a lot of time to enter them all.

Solution with files for the above problems is:

A File is place on your physical disk where a group of related data is stored.

- 1) Storing data in a file will preserve your data even if the program terminates.
- 2) If you have a file containing all the data, you can easily access the contents of the file using few commands in c.
- 3) You can easily move your data from one computer to another without any changes.

TYPES OF FILES:

When dealing with files, there are two types of files you should know about.

- 1) Text files
- 2) Binary files

1) Text Files: Text files are normal .txt files that you can easily create using notepad or any simple text editors. When you open those text files, you will see all the contents within the files as plain text. You can easily create using notepad or simple text editors.

They take minimum effort to maintain, are easily readable, and provide least security and takes bigger storage space.

2) Binary Files: Binary files are mostly the .bin files in your computer. Instead of storing data in plain text, they store it in the binary form (0's and 1's). They can hold higher amount of data, or not readable easily and provides a better security than text files.

USING THE FILES IN C LANGUAGE:

A file is a place on the disk where a group of related data is stored. When working with files, we need to declare a pointer of type file. This declaration is needed for communication between the file and program.\

In C, you can perform 6 major operations on the file either it may be text or binary files.

- 1) Creation of new file
- 2) Opening an existing file.
- 3) Reading from a file.
- 4) Writing to a file.
- 5) Moving to a specific location in a file.
- 6) Closing a file.

FILE HANDLING FUNCTIONS IN C:

- File is created for permanent storage of data. It is readymade structure.
- In C language, we use a structure pointer of the file type to declare a file.
`FILE *FP;`
- There are two distinct ways to perform file operations in C.
- The first one is known as the low-level I/O and uses UNIX system calls.
- The second method is referred to as the high-level I/O operation and uses functions in C's standard I/O library.

The important high level I/O file handling functions that are available in the stdio.h library are.

FUNCTION	DESCRIPTION
fopen()	Creates new file or open an existing file
fclose()	Closes a file which has been opened for use
getc ()	Reads a character from a file
putc()	Writes a character to a file
fscanf ()	Reads a set of data to a file
fprintf ()	Writes a set of data to a file
fgetc()	Read string of character from a file
fputc()	Write string of character from a file
ftell()	Gives current position in the file
feof()	Detects end of file marker in the file.

In C language processing of a file includes the following steps.

- Declare a file pointer variable by using FILE data structure.
- Open a file using fopen() function
- Process the file using suitable function.
- Close the file using fclose() function.

CREATING OR OPENING A FILE:-

Before you can Read/Write information from (to) a file on a disk we must open a file. To open a file we have called a function fopen().

General syntax for open a file

```
FILE *fp;
fp=fopen("filename","mode");
```

The first statement declares the variables fp as a "pointer to the data type file".

In the second statement "file name" is the name of the name of the file to be opened and "mode" specifies the purpose of opening the file.

modes in standard I/O can be of following types.

File mode	Description of mode	During inexistence of file
r	Open for reading	If the files doesnotexist,fopen() return NULL
w	Open for writing	If the file exists,its contents are overwritten. If the file doesnotexist,it will be create.
a	Open for append. i.e, 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	If the file doesnot exist fopen() returns NULL
w+	Open for both reading and writing	If the file exists,it contents are over written. If the file doesnot exist it will be created.
a+	Open for both reading and appending	If the file doesnot exist it will be created.

The above file modes are used for text files only. If you are going to handling binary files, then you will use following access mode rb, wb, ab, rb+, wb+, ab+

Example:

```
#include<stdio.h>
int main()
{
FILE *fp;
fp=open("sample.txt","w");
return 0;
}
```

CLOSING A FILE:

The file should be closed after reading or writing. This ensures that all out standing information associated with the file is flushed out from the buffers and all links to the file are broken. The fclose() function is used to close an already opened file.

Syntax: fclose(file-pointer);

This would close the file associated with the file pointer 'fptr'.

Example:

```
#include<stdio.h>
int main()
{
FILE *fp;
Fp=fopen("sample.txt", "w");
```

```
Fclose(fp);  
Return 0;  
}
```

Here `fclose()` function closes the file and returns 'zero' on success, or 'EOF' if there is an error in closing the file.

3. Writing to a file

To write the file, we must open the file in a mode that supports writing.

For example, if you open a file in "w" mode, you will be able to write the file

```
FILE *fpw;  
fpw = fopen("C:\\newfile.txt","w");
```

4. Reading a File

To read the file, we must open it first using any of the mode,

for example if you only want to read the file then open it in "r" mode.

```
/* Opening a file in r mode*/  
fp1= fopen ("C:\\myfiles\\newfile.txt", "r");
```

ACCEPTING A COMMAND LINE ARGUMENT:

Command line arguments are some values which are passed from the command line to your C programs when they are executed. These are important for your program especially when you want to control your program from outside instead of hard coding those values inside the code.

The command line arguments are handled using `main()` function arguments where 'argc()' refers to the number of arguments passed, and `argv[]` is a pointer array which points to each argument passed to the program.

Example program:

```
#include<stdio.h>  
#include<stdlib.h>  
int main(intargc,char *argv[ ])  
{  
if(argc!=5)  
{  
printf(" arguments passed through commandline not equal to 5");  
return 1;  
}  
}
```

```

printf("\n program name :%s\n",argv[0]);
printf("1starg : %s\n",argv[1]);
printf("2ndarg :%s\n",argv[2]);
printf("3rdarg :%s\n",argv[3]);
printf("4th arg :%s\n",argv[4]);
printf("5tharg :%s\n",argv[5]);
return 0;
}

```

DELETING FILES:

The C library function remove deletes the given filename so that it is no longer accessible.

Syntax: int remove (const char*filename);

Here file name is the c string containing the name of the file to be deleted. This function returns zero on successful deletion of file and returns -1 on error and errno is set approximately.

Example:

```

#include<stdio.h>
int main()
{
if(remove(abc.txt)==0)
printf("deleted successfully");
else
printf("unable to delete file");
return 0;
}

```

RENAMING A FILE:

In the C programming language, the rename function changes the name of a file. You must close the file before renaming, as a file cannot be renamed if it is open.

Syntax: int rename(const char *old_filename, const char *new_file name);

New_filename – this is the c string containing the new name for the file.

This function returns zero on successful rename, retrn -1 on error, and errno is set approximately.

```

charoldname[ ]= "sample.txt";
char newname[ ]="newsample.txt";
ref=rename(oldname,newname);

```

Q: Explain about C - Error Handling?

C programming does not provide direct support for error handling but being a system programming language, it provides you access at lower level in the form of return values. Most of the C or even Unix function calls return -1 or NULL in case of any error and set an error code **errno**. It is set as a global variable and indicates an error occurred during any function call. You can find various error codes defined in `<error.h>` header file.

So a C programmer can check the returned values and can take appropriate action depending on the return value. It is a good practice, to set `errno` to 0 at the time of initializing a program. A value of 0 indicates that there is no error in the program.

perror() and **strerror()** functions which can be used to display the text message associated with **errno**.

- ✓ The **perror()** function displays the string you pass to it, followed by a colon, a space, and then the textual representation of the current `errno` value.
- ✓ The **strerror()** function, which returns a pointer to the textual representation of the current `errno` value.

Divide by Zero Errors:

Programmers do not check if a divisor is zero and finally it creates a runtime error.

The code below fixes this by checking if the divisor is zero before dividing –

```
#include <stdio.h>
#include <stdlib.h>
main()
{
    int dividend = 20;
    int divisor = 0;
    int quotient;
    if( divisor == 0){
        fprintf(stderr, "Division by zero! Exiting...\n");
        exit(-1);
    }
    quotient = dividend / divisor;
    fprintf(stderr, "Value of quotient : %d\n", quotient );
    exit(0);
}
```

Program Exit Status

It is a common practice to exit with a value of EXIT_SUCCESS in case of program coming out after a successful operation. Here, EXIT_SUCCESS is a macro and it is defined as 0.

If you have an error condition in your program and you are coming out then you should exit with a status EXIT_FAILURE which is defined as -1.

```
#include <stdio.h>
#include <stdlib.h>
main()
{
    int dividend = 20;
    int divisor = 5;
    int quotient;
    if( divisor == 0)
    {
        fprintf(stderr, "Division by zero! Exiting...\n");
        exit(EXIT_FAILURE);
    }
    quotient = dividend / divisor;
    fprintf(stderr, "Value of quotient : %d\n", quotient );
    exit(EXIT_SUCCESS);
}
```