Module I: Introduction

Notes

Course Content:

- Basic Computer Organization
- Computer Hardware Components
- Primary Memory – RAM, ROM, Secondary Memory
- Types of Software
- Introduction to Compilers, Interpreters, Assembler, Linker, Loader
- Introduction to C compiler and its different versions
- Basic Operating System Concepts, Functions of Operating system
- Types of Operating System.

Key Learning Objective :

At the end of this module, you will be able to:

1. Understand and apply basic structure of computers
2. Understand and apply various types of Software and usage
3. Concept of Compiler, Interpreter and Assembler
4. Get familiar with C compiler
5. Concept of operating system, its functions and types

Notes

UNIT 1.1: Basic Computer Organization

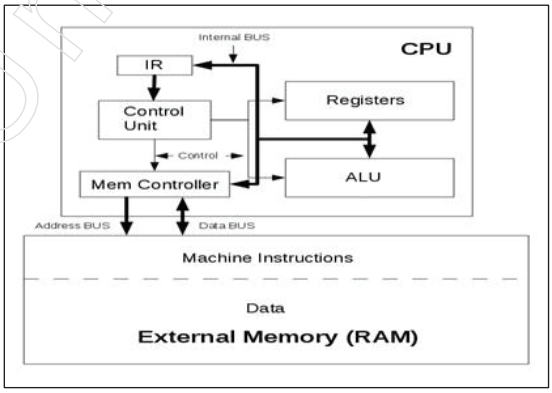
Unit Outcomes :

At the end of this unit, you will be able to:

1. Define basic structure of computers
2. Categorizes the types of memory
3. Define the types of software and usage
4. Define compiler, interpreter, assembler
5. Define linker and loader
6. Differentiate C Compiler from other compilers
7. Define Operating System
8. Summaries types of Operating Systems and its functionalities

1.1.1 Basic Computer Organization

Computer instructions are the basic components of a machine language program. They are also known as macro operations, since each one is comprised of a sequences of microoperations. Each instruction instructs a series of microoperations that carry operands from registers or memory, perhaps perform arithmetic, logic, or shift operations, and store results in registers or memory. Instructions are encoded as binary instruction codes. Every instruction code includes of an operation code, or opcode, which specifies the whole purpose of the instruction. The control unit is accountable for decoding the opcode and operand bits in the instruction register, and producing the control signals required to drive all other hardware in the CPU to execute the series of microoperations that encompass the instruction



1.1.2 Computer Hardware Components

Computer has five units.

They are:

1. Input Devices
2. Memory Unit
3. Arithmetic & Logic Unit
4. Control Unit.
5. Output Devices

Input Devices :

Computers take information through input unit. Keyboard, mouse, Joystick, Light pen, Scanner, Bar code reader are few examples of Input unit. Whenever we press a key, through cable it is translated to binary code as computers only understand binary format (i.e. 0 and 1) and then transmitted to either memory or processor.

The standard data input unit for a computer is Keyboard . It comprises of several keys which follows the standard QWERTY layout with numeric keypad and additional function keys for control purposes.

Another prevalent input device is the Mouse. Movement of mouse is shown on the screen and an arrow is seen during mouse movement which is known as a 'cursor'. There are two buttons and a scroll ball present in a mouse. You will require to click the buttons of the mouse to pick an option and scroll the ball to make the page move up and down.

Joystick is basically used for gaming purpose. It is a rotary lever. It enables you to move within the screen's environment, and is widely used in the computer games industry.

An Alternative input device called a Light Pen is a pointing device modeled like a pen and is linked to a Visual Display Unit (VDU). The tip of the light pen contains a light-sensitive element which, when placed against the screen, detects the light from the screen facilitating the computer to identify the location of the pen on the screen.

A photo or text to be input into a computer through a device called Scanners which is an input device. Two types of Scanners are there; one is flatbed scanner and another is hand-held scanner.

A Bar Code reader is another input device. A Bar Code is a pattern printed in lines of contrasting thickness. You might have seen bar codes on goods in libraries, in supermarkets, on magazines etc. Bar codes provide a quick method of recording the sale of items.

Memory Unit :

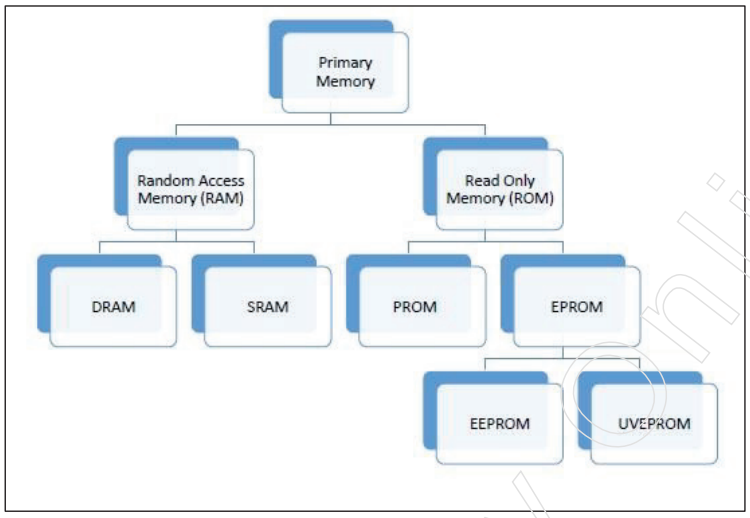
The accountability of memory unit is to stores data as well as programs. Two categories of memory units are there , Primary Memory and Secondary Memory. The capability to store instructions that form a computer program, and the information that the instructions manipulate is what makes stored program architecture computers versatile.

1.1.3 Primary Memory:

Memory is needed in computers to store data and instructions. Memory is physically structured as a large number of cells that are competent of storing one bit each. Logically they are structured as groups of bits called words that are assigned an address. Data and instructions are accessed through these memory address. The rapidity with which these memory addresses can be accessed determines the cost of the memory. Quicker the memory speed, higher the price.

Computer memory can be believed to be arranged in a hierarchical way where memory with the fastest access speeds and highest costs lies at the top although those with lowest speeds and hence lowest costs lie at the bottom. Based on this criteria memory is of two categories – primary and secondary. Here we will look at primary memory in detail.

Notes



Purpose of Storage

The essential components of a general-purpose computer are arithmetic and logic unit, control circuitry, storage space, and input/output devices. If storage were removed, the device we had would be a simple calculator instead of a computer. The ability to store instructions that form a computer program, and the information that the instructions manipulate is what makes stored program architecture computers versatile.

Primary memory is very fast which works at electronic speed. Generally, before getting executed, programs should be stored in memory. Primary memory is essential but expensive. Primary storage is precisely connected to the central processing unit (CPU) of the computer. It is used when large number of data & programs are needed to store, particularly the information that we don't access very frequently. We have two types of memory like RAM and ROM.

RAM (Random Access Memory):

RAM is a volatile memory. It is also called temporary memory. when you turn off your computer, the data that are there inside RAM got obliterated. So, before turning off your computer you must save your work to permanent storage device. If you have more RAM, the less frequently the computer must access instructions and data from the sluggishly accessed hard disk.

Types of RAM:

Two types of RAM are used in computers: Dynamic RAM and Static RAM.

Dynamic RAM (DRAM): The information stored in Dynamic RAM has to be refreshed after every few milliseconds otherwise it will get erased. DRAM has higher storage capacity and is cheaper than Static RAM.

Static RAM (SRAM): The information stored in Static RAM need not be refreshed, but it remains stable as long as power supply is provided. SRAM is costlier but has higher speed than DRAM.

Additional kinds of integrated and quickly accessible memory are Read Only Memory (ROM), Programmable ROM (PROM), and Erasable Programmable ROM (EPROM). These are used to maintain special programs and data, such as the BIOS,

that need to be in your computer all the time. ROM is “built-in” computer memory comprising data that normally can only be read, not written to (hence the name read only).

ROM (Read Only Memory):

ROM is non volatile memory i.e. when you switch off your computer, the data and information which are stored does not get erased. ROM includes the programming that permits computer to be “boot up” each time you turn it on. The ROM is sustained by a small long-life battery in the computer called the CMOS battery. It is non volatile, but not suited to storage of large quantities of data because it is costly to produce. Typically, ROM ought to be completely expunged before it can be rewritten and that can be done only by the manufacturer.

PROM (Programmable Read Only Memory):

A deviation of the ROM chip is programmable read only memory. PROM can be programmed to record information using a provision known as PROM-programmer. Nevertheless, the chip has been programmed the recorded information cannot be changed, i.e. the PROM becomes a ROM and the information can only be read.

EPROM (Erasable Programmable Read Only Memory):

As the name suggests the Erasable Programmable Read Only Memory, information can be removed, and the chip programmed a new to record different information using a special PROM-Programmer. When EPROM is in use information can only be read and the information persists on the chip until it is erased.

1.1.4 Secondary Memory:

Storage devices consist of hard disks, floppy disks, CDROMs, and tape backup systems. The phrases auxiliary storage, auxiliary memory, and secondary memory have also been used for this kind of data repository.

STORAGE DEVICES

The objective of storage in a computer is to keep data or information and get that data to the CPU as swiftly as possible when it is needed. Computers utilize disks for storage: hard disks that are placed inside the computer, and floppy or compact disks that are utilized externally.

- Computers Method of storing data & information for long term basis i.e. even after PC is switched off.
- It is non - volatile
- Can be easily removed and moved & attached to some other device
- Memory capacity can be extended to a greater extent
- Cheaper than primary memory

Storage Involves two processes:

a. Writing data
b. Reading data

Notes

Floppy Disks

The floppy disk drive (FDD) was developed at IBM by Alan Shugart in 1967. The first floppy drives utilized an 8-inch disk (later called a “diskette” as it got smaller), which progressed into the 5.25-inch disk that was used on the first IBM Personal Computer in August 1981. The 5.25-inch disk held 360 kilobytes compared to the 1.44 megabyte capacity of today’s 3.5-inch diskette.

The 5.25-inch disks were dubbed “floppy” since the diskette packaging was a very flexible plastic envelope, unlike the rigid case used to hold today’s 3.5-inch diskettes.

Hard Disks

Computer uses two kinds of memory: primary memory which is stored on chips located on the motherboard, and secondary memory that is stowed in the hard drive. Primary memory holds all the vital memory while Secondary memory owns the information that you store in the computer.

Within the hard disk drive case, circular disks are placed that are made from polished steel. On the disks, there are several tracks or cylinders. Inside the hard drive, an electronic reading/writing device termed the head passes back and forth over the cylinders, reading information from the disk or writing information to it. Hard drives roll at 3600 or more rpm (Revolutions Per Minute) - that means that in one minute, the hard drive spins around over 7200 times!

Optical Storage

- Compact Disk Read-Only Memory (CD-ROM)
- CD-Recordable (CD-R)/CD-Rewritable (CD-RW)
- Digital Video Disk Read-Only Memory (DVD-ROM)
- DVD Recordable (DVD-R/DVD Rewritable (DVD-RW)
- Photo CD

Optical Storage Devices Data is deposited on a reflective surface so it can be read by a beam of laser light. Two Kinds of Optical Storage Devices

- CD-ROM (compact disk read-only memory)
- DVD-ROM (digital video disk read-only memory)

Compact Disks

CDs use pits (microscopic indentations) and lands (flat surfaces) to store information considerably the same way floppies and hard disks use magnetic and non-magnetic storage. Inside the CD-Rom, a laser that reflects light off the surface of the disk to an electric eye. The pattern of reflected light (pit) and no reflected light (land) establishes a code that represents data.

CDs usually store about 650MB. This is quite a bit more than the 1.44MB that a floppy disk stores. A DVD or Digital Video Disk holds even additional information than a CD, because the DVD can store information on two levels, in smaller pits or sometimes on both sides.

Recordable Optical Technologies

- CD

Notes

- DVD
- Pen Drives / Flash Drives

CD ROM - Compact Disc Read Only Memory.

Contrasting magnetic storage device which store data on multiple concentric tracks, the entire CD formats store data on one physical track, which spirals continuously from the center to the outer edge of the recording area. Data dwells on the thin aluminium substrate instantly beneath the label. The data on the CD is recorded as a sequence of microscopic pits and lands physically embossed on an aluminium substrate. Optical drives use a low power laser to read data from those discs without physical contact between the head and the disc which contributes to the high trustworthiness and permanence of storage device.

In order to write the data on a CD a higher power laser are used to record the data on a CD. It creates the pits and land on aluminium substrate. The data is stored perpetually on the disc. These types of discs are called as WORM (Write Once Read Many). Data written to CD cannot consequently be deleted or overwritten which can be classified as advantage or disadvantage depending upon the requirement of the user. Nevertheless if the CD is moderately filled then the more data can be added to it later on till it is full. CDs are usually inexpensive and cost effective in terms of storage capacity and transferring the data.

The CD‘s were further developed where the data could be deleted and re written. These types of

CDs are called as CD Rewritable. These types of discs can be used by deleting the data and making the space for new data. These CD‘s can be written and rewritten at least 1000 times.

CD ROM Drive

CD ROM drives are so well consistent and have become so ubiquitous that many treat them as commodity items. Although CD ROM drives differ in reliability, which standards they support and numerous other respects, there are two important performance measures.

- Data transfer rate
- Average access

The data on a CD is saved on tracks, which spirals from the centre of the CD to outer edge. The portions of the tracks towards centre are shorter than those towards the edge. Relocating the data under the head at a constant rate requires spinning the disc faster as the head moves from the centre where there is less data per revolution to the edge where there is more data. Therefore, the rotation rate of the disc adjustments as it progresses from inner to outer portions of the disc.

Pen Drives / Flash Drives

- Pen Drives / Flash Drives are flash memory storage devices.
- They are faster, portable and have a capability of storing large data.
- It consists of a small printed circuit board with a LED encased in a robust plastic
- The male type connector is used to connect to the host PC
- They are also used a MP3 players

Notes

A Computer has five operational autonomous units like Input Unit, Memory Unit, Arithmetic & Logic Unit, Output Unit, Control Unit.

Arithmetic & Logic Unit :

ALU (Arithmetic & Logical Unit) performs Arithmetic and Logic operations. ALU and this operation are initiated once the operands are brought into the processor.

Output Unit : It displays the processed result to outside world.

Control Unit:

Control Unit is the portion of the computer's central processing unit (CPU), which manages the operation of the processor. It was contained as part of the Von Neumann Architecture by John von Neumann. It is the accountability of the Control Unit to tell the computer's memory, arithmetic/logic unit and input and output devices how to react to the instructions that have been sent to the processor. It brings internal instructions of the programs from the main memory to the processor instruction register, and based on this register contents, the control unit generates a control signal that supervises the execution of these instructions. A control unit works by receiving input information to which it transforms into control signals, which are then sent to the central processor. The computer's processor then tells the enclosed hardware what operations to perform. The functions that a control unit performs are dependent on the type of CPU because the architecture of CPU varies from manufacturer to manufacturer.

OUTPUT DEVICES :

An output device is computer hardware that receives data from a computer and translates that data into either text, audio, video or hard copy such as a printed document.

The key difference between an input device and an output device is that an input device takes data from user and an output device deliver output either on monitor or printer etc.

Example of Output devices are Monitor, Printer, Projector, Headphones, Speakers etc.

A **Printer** is another popular part of a computer system. It takes what you look at on the computer screen and prints it on paper.

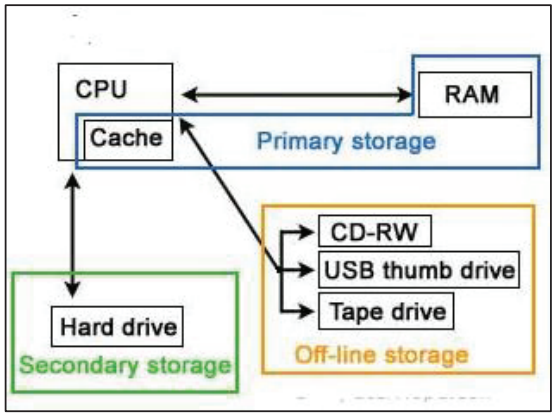
Speakers are output devices that permit you to hear sound from computer. Computer speakers are like stereo speakers. There are mostly two of them and they come in various sizes.

Primary Memory – RAM, ROM:

MEMORY OR PRIMARY STORAGE :

Primary Storage

Primary storage is precisely connected to the central processing unit of the computer. It ought to be present for the CPU to function correctly, just as in a biological analogy the lungs must be present (for oxygen storage) for the heart to function (to pump and oxygenate the blood). As shown in the diagram, primary storage typically consists of three kinds of storage:



Processors Register

Processors Registers is the internal to the central processing unit. Registers include information that the arithmetic and logic unit needs to carry out the current instruction. They are technically the fastest of all forms of computer storage.

Main memory

Main memory contains the programs that are presently being run and the data the programs are operating on. The arithmetic and logic unit can very rapidly transfer information between a processor register and locations in main storage, also known as a “memory addresses”. In modern computers, electronic solid-state random-access memory is employed for main storage, and is precisely connected to the CPU via a “memory bus” and a “data bus”.

Cache memory

Cache memory is a unique type of internal memory used by several central processing units to enhance their performance or “throughput”. Some of the information in the main memory is duplicated in the cache memory, which is marginally slower but of much superior capacity than the processor registers, and faster but much tinier than main memory.

1.1.5 Types of Software:

Generally, there are two main classifications of software, which are namely, System Software along with the Application Software.

1.1.5.1 System Software:

System Software In case of a system software, it helps the user as well as the hardware to function and even interact with each other easily. Essentially, it is a software which is used to manage the behaviour of the computer hardware in order to offer basic functionalities which are needed by the user. In simpler word, it can be said that system software is essentially an intermediator or even a middle layer between the user as well as the hardware. These software sanction an environment or platform for the other software to easily work in. Hence, it is the reason why the system software is quite important in the management of the entire computer system. Whenever you turn on the computer first, it is this system software which gets initialized and then gets loaded in the system’s memory. A system software essentially runs in the background, and it is not actually utilized by the end users.

Notes

1.1.5.2 Application Software :

They are also popularly known as end-user programs or even productivity programs which assist the user in completing various tasks like conducting online research, making notes, designing graphics, maintaining accounts, carrying out calculations or even playing computer games. They essentially lie above the system software. They are actually used by the end-user as well as have specific functionality or tasks which they are designed to perform. These software are often developed through custom software development, based on the requirements of the users.

1.1.6 Introduction to Computers

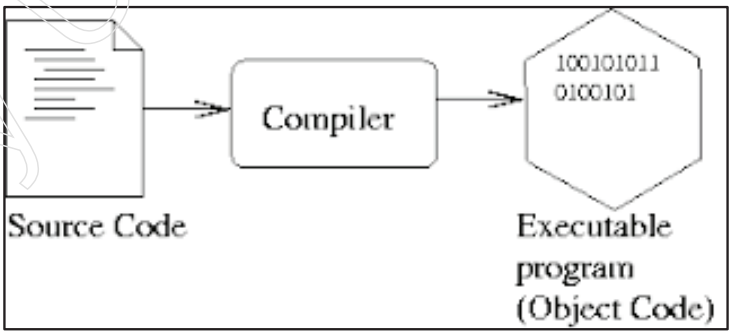
Computers are a reasonable mix of software and hardware. Hardware is simply a piece of mechanical device and its functions are being monitored by a compatible software. Hardware recognizes instructions in the form of electronic charge, which is the counterpart of binary language in software programming. Binary language has only two alphabets, 0 and 1. To instruct, the hardware codes must be written in binary format, which is simply a sequence of 1s and 0s. It would be a challenging and cumbersome task for computer programmers to write such codes, which is why we have compilers to write such codes.

Compiler

A Compiler is a program that converts a number of statement of program into binary language, but it is more intelligent than interpreter because it goes through the entire code at once and can tell the possible errors and limits and ranges. But this makes it's operating time a little slower. it is platform-dependent. its help to detect error and get displayed after reading the entire code by compiler.

Cross-compiler

A compiler that runs on platform (A) and is capable of generating executable code for platform (B) is called a cross-compiler.



Source-to-source Compiler

A compiler that takes the source code of one programming language and translates it into the source code of another programming language is called a source-to-source compiler.

Interpreter

An interpreter, like a compiler, translates high-level language into low-level machine language. The distinction lies in the way they read the source code or input. A compiler reads the whole source code at once, creates tokens, checks semantics,

generates intermediate code, executes the whole program and may involve many passes. In contrast, an interpreter reads a statement from the input, transforms it to an intermediate code, executes it, then takes the next statement in sequence. If an error occurs, an interpreter stops execution and reports it. though a compiler reads the whole program even if it encounters several errors.

There are several types of interpreter:

- Syntax-directed interpreter
- Threaded interpreter
- Bytecode interpreter

Assembler

An assembler translates assembly language programs into machine code. The output of an assembler is called an object file, which contains a combination of machine instructions as well as the data required to place these instructions in memory.

Linker

Linker is a computer program that links and combines various object files together in order to make an executable file. All these files might have been compiled by independent assemblers. The key task of a linker is to search and locate referenced module/routines in a program and to ascertain the memory location where these codes will be loaded, making the program instruction to have absolute references.

1. Dynamic Linker:-

- It is employed during run time.
- It requires less memory.
- In dynamic linking there are many chances of error and failure chances.
- Linking stored the program in virtual memory to save RAM, need shared library

2. Static Linker:-

- It is implemented during compilation of source program.
- It requires more memory.
- Linking is implemented before execution in static linking.
- It is faster and portable.
- In static linking there are less chances to error and No chances to failure.

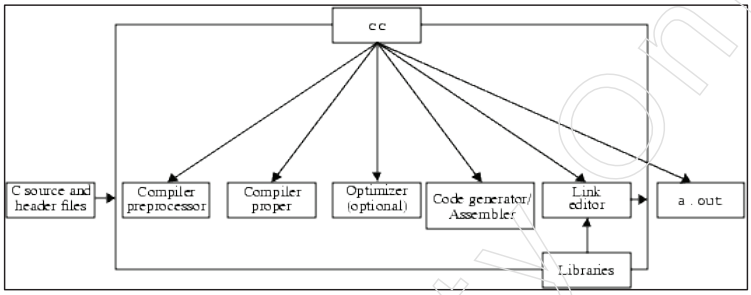
Loader

Loader is a element of operating system and is accountable for loading executable files into memory and execute them. It computes the size of a program (instructions and data) and creates memory space for it. It initializes various registers to initiate execution.

Notes

1.1.7 Introduction to C compiler and its different versions:

The C compilation system comprises of a compiler, an assembler, and a link editor. The cc command invokes each of these components automatically unless you use command-line options to specify otherwise. The C compiler optimizer removes redundancies, optimally allocates registers, schedules instructions, and reorganizes code. Select from multiple levels of optimization to obtain the best balance between application speed and use of memory.



1.1.8 Basic Operating System Concepts:

An Operating system is essentially an intermediary agent in the middle of the user and the computer hardware. It Manages the computer's resources. Simplifies hardware control for applications.

All operating systems use one fundamental abstraction: the process. A process can be described either as "an instance of a program in execution" or as the "execution context" of a running program. In traditional operating systems, a process executes a specific sequence of instructions in an address space; the address space is the set of memory addresses that the process is permissible to reference. Modern operating systems allow processes with multiple execution flows — that is, multiple sequences of instructions executed in the same address space.

1.1.9 Functions of Operating system:

The operating system utilizes password protection to protect user data and similar other techniques. it also precludes unauthorized access to programs and user data. Monitors overall system health to help improve performance. records the response time between service requests and system response to have a comprehensive view of the system health .Operating system Keeps track of time and resources employed by various tasks and users, this information can be used to track resource usage for a particular user or group of user. Operating system frequently monitors the system to detect errors and avoid the malfunctioning of computer system.

1.1.9.1 Types of Operating System:

1. Simple Batch System
2. Multiprogramming Batch System
3. Multiprocessor System
4. Desktop System
5. Distributed Operating System
6. Clustered System
7. Realtime Operating System
8. Handheld System

Notes

1.1.10 The advantage of using Operating System

- Allows you to conceal details of hardware by establishing an abstraction
- Effortless to use with a GUI
- Presents an environment in which a user may execute programs/applications
- The operating system must make sure that the computer system convenient to use
- Operating System acts as an mediator among applications and the hardware components
- It affords the computer system resources with easy to use format
- Plays as an intermediator between all hardware's and software's of the system

Activity :

1. The simultaneous execution of two or more programs in one computer is called_____
2. What is the function of the controller unit?
3. Give example for a multiuser operating system.
4. Give examples for single user operating systems?
5. Which smaller unit of the CPU performs all arithmetic and logic functions in a computer?
6. Which register example holds the address of the current instruction being processed?
7. Distinguish RAM and ROM.
8. What are the advantages of Operating Systems
9. What are The two kind of main memory
10. Give an example of non-volatile memory.
11. The storage that supplements the primary internal storage of a computer is known as ?
12. What is the name given to the memory which works on time sharing principle in order to create an illusion of infinite memory space?
13. What is the main advantage of semiconductor RAM
14. What is the basic characteristics of DRAMs?
15. Differentiate Loader and Linker
16. Distinguish Compiler and Interpreter

Notes

Model II: Programming in C

Course Content:

- History of C
- Introduction of C
- Basic structure of C program
- Concept of variables, constants and data types in C
- Operators and expressions: Introduction, arithmetic, relational, Logical, Assignment, Increment and decrement operator, Conditional, bitwise operators, Expressions
- Operator precedence and associativity
- Managing Input and output Operation, formatting I/O.

Key Learning Objective :

1. Understanding of basic C programs
2. Concept of variables and data types
3. Understanding of different types of operators
4. Understand and apply the concept to develop applications

UNIT 2.1: History of C

Notes

Unit Outcomes :

At the end of this unit, you will be able to:

1. Understand syntax and semantics - variables, data types , operators
2. Get Familiar with basic structure of C Programs
3. State different types of operators
4. Understand how to work with input and output

2.1.1 History of C

The C programming language came out of Bell Labs in the initial 1970s. Rendering to the Bell Labs paper “The Development of the C Language” by Dennis Ritchie. “The C programming language was devised in the early 1970s as a system implementation language for the nascent Unix operating system.

Stemmed from the type less language BCPL, it evolved a type structure; built on a tiny machine as a tool to enhance a meager programming environment.”

Formerly, Ken Thompson, a Bell Labs employee, preferred to make a programming language for the new Unix platform. Thompson adapted the BCPL system language and invented B.

However, not many utilities were ever written in B due to its sluggish nature and inability to take advantage of PDP-11 features in the operating system. This led to Ritchie bettering on B, and thus producing C.

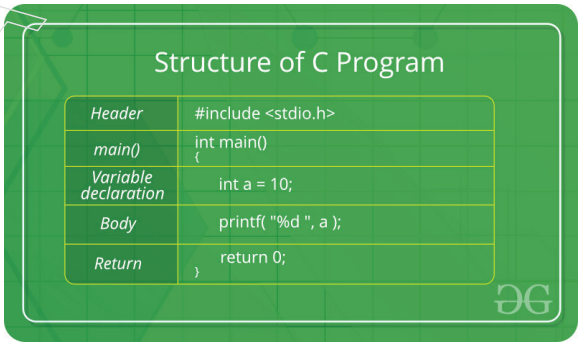
2.1.2 Introduction of C

C is a general-purpose programming language that is exceptionally popular, simple and flexible. It is machine-independent, structured programming language which is used significantly in numerous applications.

C was the fundamental language to write the whole lot from operating systems (Windows and many others) to complex programs resembling the Oracle database, Git, Python interpreter and more.

It is believed that ‘C’ is a divinity’s programming language. One can say, C is a center for the programming. If you understand ‘C,’ you can easily grasp the expertise of the other programming languages that uses the concept of ‘C’

2.1.3 Basic structure of C program



Notes

The components of the above structure are:

1. **Header Files Inclusion:** The first and foremost element is the inclusion of the Header files in a C program. A header file is a file with extension .h which comprises C function declarations and macro definitions to be shared between several source files.

Some of C Header files:

- stddef.h – Describes several useful types and macros.
- stdint.h – Identifies precise width integer types.
- stdio.h – Specifies core input and output functions
- stdlib.h – Defines numeric conversion functions, pseudo-random network generator, memory allocation
- string.h – Defines string handling functions
- math.h – Defines common mathematical functions

Syntax to include a header file in C:

#include

Main Method Declaration: The next part of a C program is to declare the main() function. The syntax to declare the main function is:

Syntax to Declare main method:

int main()

{}

Variable Declaration:

The following part of any C program is the variable declaration. It implies to the variables that are to be used in the function. Please note that in the C program, no variable can be used deprived of being declared. Also in a C program, the variables are to be declared prior to any operation in the function.

Example:

```
int main()
{
   int a;
.
.
```

Body:

Body of a function in C program, refers to the operations that are staged in the functions. It can be anything like manipulations, searching, sorting, printing, etc.

Example:

```
int main()
{
   int a;

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

Return Statement:

The preceding part in any C program is the return statement. The return statement describes to the returning of the values from a function. This return statement and return value hinge on upon the return type of the function. For example, if the return type is void, then there will be no return statement. In any new case, there will be a return statement and the return value will be of the type of the stipulated return type.

Example:

```
int main()
{
   int a;

   printf("%d", a);

   return 0;
}
```

2.4 Concept of variables

A variable is a name given to a storage area that our programs can manipulate. Each variable in C has a specific type, which regulates the size and layout of the variable's memory; the range of values that can be stored inside that memory; and the set of operations that can be applied to the variable.

The name of a variable can be comprised of letters, digits, and the underscore character. It must begin with either a letter or an underscore. Upper and lowercase letters are distinct because C is case-sensitive. Based on the basic types explained in the previous chapter, there will be the following basic variable types −

char

Typically a single octet(one byte). It is an integer type.

int

The most natural size of integer for the machine.

Notes

float

A single-precision floating point value.

double

A double-precision floating point value.

void

Represents the absence of type.

A variable definition identifies the compiler where and how considerably storage to create for the variable. A variable definition specifies a data type and contains a list of one or more variables of that type as follows −

type variable_list;

Here, type must be a valid C data type including char, w_char, int, float, double, bool, or any user-defined object; and variable_list may comprise of one or more identifier names divided by commas. Some valid declarations are shown here −

int i, j, k;

char c, ch;

float f, salary;

double d;

The line int i, j, k; declares and defines the variables i, j, and k; which instruct the compiler to create variables named i, j and k of type int.

Variables can be initialized (assigned an initial value) in their declaration. The initializer consists of an equal sign followed by a constant expression as follows −

type variable_name = value;

2.1.5 Constants and data types in C

In order to define a variable whose value cannot be altered; you can utilize the const keyword. This will create a constant. For example,

const double PI = 3.14;

In C programming, data types are declarations for variables. This affects the type and size of data associated with variables. For example,

int myVar;

Type	Size (bytes)	Format Specifi-er
int	at least 2, usually 4	%d, %i
char	1	%c
float	4	%f
double	8	%lf
short int	2 usually	%hd
unsigned int	at least 2, usually 4	%u

long int	at least 4, usually 8	%ld, %li
long long int	at least 8	%lld, %lli

2.1.6 Operators and expressions

An operator is a symbol that instructs the compiler to perform specific mathematical or logical functions. C language is rich in built-in operators and provides the following types of operators −

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators

2.6.1 Arithmetic Operators

These operators do only basic arithmetic operations.

Operator	Description	Example
+	Adds two operands.	A + B = 30
−	Subtracts second operand from the first.	A − B = -10
*	Multiplies both operands.	A * B = 200
/	Divides numerator by de-numerator.	B / A = 2
%	Modulus Operator and remainder of after an integer division.	B % A = 0
++	Increment operator increases the integer value by one.	A++ = 11
--	Decrement operator decreases the integer value by one.	A-- = 9

2.1.6.2 Relational Operators

These operators checks the relations between two operands. If result is true then returns 1 otherwise returns 0.

Operator	Description	Example
==	Checks if the values of two operands are equal or not. If yes, then the condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not. If the values are not equal, then the condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand. If yes, then the condition be-comes true.	(A > B) is not true.

Notes

<	Checks if the value of left operand is less than the value of right operand. If yes, then the condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand. If yes, then the condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand. If yes, then the condition becomes true.	(A <= B) is true.

2.1.6.3 Logical Operators

These operators performs logical operations.

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands is non-zero, then the condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. It is used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make it false.	!(A && B) is true.

2.1.6.4 Bitwise Operators

1. In C, the bitwise AND (&) takes two numbers as operands and does AND operation on every bit of two numbers. If both bits are 1then The result comes as 1 in this operation.
2. In C, the bitwise OR (|) takes two numbers as operands and does OR operation on every bit of two numbers. If any of the two bits is 1 then the result comes as 1 in this operation.
3. The bitwise XOR (^) in C takes two numbers as operands and perform XOR operation on every bit of two numbers. If the two bits are different then the result of XOR comes as 1.
4. The left shift (<<) operator in C takes two numbers and left shifts the bits of the first operand, the second operand decides the number of places to shift.
5. The right shift (>>) in C takes two numbers and right shifts the bits of the first operand, the second operand decides the number of places to shift.
6. The bitwise NOT (~) in C takes one number and inverts all bits of it

p	q	p & q	p | q	p ^ q
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

Notes

2.1.6.5 Assignment Operators

Operator	Description	Example
=	Simple assignment operator. Assigns values from right side operands to left side operand	C = A + B will assign the value of A + B to C
+=	Add AND assignment operator. It adds the right op-erand to the left operand and assign the result to the left operand.	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator. It subtracts the right operand from the left operand and assigns the result to the left operand.	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator. It multiplies the right operand with the left operand and assigns the result to the left operand.	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator. It divides the left operand with the right operand and assigns the result to the left operand.	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator. It takes modulus using two operands and assigns the result to the left operand.	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator.	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator.	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator.	C &= 2 is same as C = C & 2

Example 1: Arithmetic Operators

```
// Working of arithmetic operators

#include <stdio.h>

int main()

{

   int a = 9,b = 4, c;


   c = a+b;

   printf("a+b = %d \n",c);

   c = a-b;

   printf("a-b = %d \n",c);
```

Notes

```
    c = a*b;
    printf("a*b = %d \n",c);
    c = a/b;
    printf("a/b = %d \n",c);
    c = a%b;
    printf("Remainder when a divided by b = %d \n",c);

    return 0;

}
```

Output :

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

Example 2: Increment and Decrement Operators

```
// Working of increment and decrement operators
#include <stdio.h>
int main()
{
    int a = 10, b = 100;
    float c = 10.5, d = 100.5;
    printf("++a = %d \n", ++a);
    printf("--b = %d \n", --b);
    printf("++c = %f \n", ++c);
    printf("--d = %f \n", --d);

    return 0;
}
```

Output :

```
++a = 11
```

```
--b = 99
++c = 11.500000
--d = 99.500000
```

Example 3: Assignment Operators

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

Example 4: Relational Operators

```
// Working of relational operators
#include <stdio.h>
int main()
{
    int a = 5, b = 5, c = 10;

    printf("%d == %d is %d \n", a, b, a == b);
```

Notes

```
    printf("%d == %d is %d \n", a, c, a == c);
    printf("%d > %d is %d \n", a, b, a > b);
     printf("%d > %d is %d \n", a, c, a > c);
    printf("%d < %d is %d \n", a, b, a < b);
    printf("%d < %d is %d \n", a, c, a < c);
    printf("%d != %d is %d \n", a, b, a != b);
    printf("%d != %d is %d \n", a, c, a != c);
    printf("%d >= %d is %d \n", a, b, a >= b);
    printf("%d >= %d is %d \n", a, c, a >= c);
    printf("%d <= %d is %d \n", a, b, a <= b);
    printf("%d <= %d is %d \n", a, c, a <= c);

    return 0;
}
```

Output:

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

Example 5: Logical Operators

```
// Working of logical operators

#include <stdio.h>
int main()
{
    int a = 5, b = 5, c = 10, result;

    result = (a == b) && (c > b);
    printf("(a == b) && (c > b) is %d \n", result);
```

```
    result = (a == b) && (c < b);
    printf("(a == b) && (c < b) is %d \n", result);
    result = (a == b) || (c < b);
    printf("(a == b) || (c < b) is %d \n", result);
    result = (a != b) || (c < b);
    printf("(a != b) || (c < b) is %d \n", result);
    result = !(a != b);
    printf("!(a == b) is %d \n", result);
    result = !(a == b);
    printf("!(a == b) is %d \n", result);

    return 0;
}
```

Output:

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

Example 6: sizeof Operator

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

Notes

```
    float b;
    double c;
    char d;
    printf("Size of int=%lu bytes\n",sizeof(a));
    printf("Size of float=%lu bytes\n",sizeof(b));
    printf("Size of double=%lu bytes\n",sizeof(c));
    printf("Size of char=%lu byte\n",sizeof(d));
    return 0;
}
```

Output :

Size of int = 4 bytes

Size of float = 4 bytes

Size of double = 8 bytes

Size of char = 1 byte

2.1.7 Managing Input and output Operation

Input means to provide the program with some data to be utilized in the program and Output means to display data on screen or write the data to a printer or a file.

C programming language provides many built-in functions to read any given input and to demonstrate data on screen when there is a need to output the result.

In this tutorial, we will understand about such functions, which can be utilized in our program to take input from user and to output the result on screen.

All these built-in functions are present in C header files, will also specify the name of header files in which a specific function is defined while reviewing about it.

scanf() and printf() functions

The standard input-output header file, named stdio.h encompasses the definition of the functions printf() and scanf(), which are used to display output on screen and to take input from user correspondingly.

```
#include <stdio.h>
int main()
{
    float num1;
    double num2;
```

```
    printf("Enter a number: ");
    scanf("%f", &num1);
    printf("Enter another number: ");
    scanf("%lf", &num2);

    printf("num1 = %f\n", num1);
    printf("num2 = %lf", num2);

    return 0;
}
```

Output:

```
Enter a number: 12.523
Enter another number: 10.2
num1 = 12.523000
num2 = 10.200000
```

When you will compile the above code, it will require you to enter a value. When you will enter the value, it will display the value you have entered on screen.

You must be wondering what is the purpose of %d inside the scanf() or printf() functions. It is known as **format string** and this informs the scanf() function, what type of input to expect and in printf() it is used to give a heads up to the compiler, what type of output to expect.

Format String	Meaning
%d	Scan or print an integer as signed decimal number
%f	Scan or print a floating point number
%c	To scan or print a character
%s	To scan or print a character string. The scanning ends at whitespace.

getchar() & putchar() functions

The getchar() function takes a character from the terminal and returns it as an integer. This function takes only single character at a time. This method can be utilized in a loop in case you want to read more than one character. The putchar() function displays the character passed to it on the screen and returns the same character. This function too exhibits only a single character at a time. In case you want to exhibit more than one characters, apply putchar() method in a loop.

Notes

gets() & puts() functions

The gets() function recites a line from stdin(standard input) into the buffer pointed to by str pointer, until either a terminating newline or EOF (end of file) occurs. The puts() function transcribes the string str and a trailing newline to stdout.

formatting I/O

C language extend console input/output functions. As the name reveals, the console input/output functions permit us to -

- Read the input from the keyboard by the user accessing the console.
- Exhibit the output to the user at the console.

Note : These input and output values could be of any primitive data type.

There are two kinds of console input/output functions -

- Formatted input/output functions.
- Unformatted input/output functions.

Functions	Description
scanf()	This function is utilized to read one or multiple inputs from the user at the console.
printf()	This function is utilized to display one or multiple values in the output to the user at the console.
sscanf()	This function is utilized to read the characters from a string and stores them in variables.
sprintf()	This function is utilized to read the values stored in various varia-bles and store these values in a character array.

Activity :

1. What would be the output of the program ?

```
int main()
{
   int a = 1;
   int b = 1;
   int c = a || --b;
   int d = a-- && --b;
   printf("a = %d, b = %d, c = %d, d = %d", a, b, c, d);
   return 0;
}
```

Answer : a = 0, b = 0, c = 1, d = 0

2. What will be the output of the C program?

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

Notes

```
int main()
{
        int i = 3;
        printf("%d", (++i)++);
        return 0;
}
```

Answer : Compile-time error

3. What will be the output of the C program?

```
#include <stdio.h>
int main()
{
int a = 10, b = 20, c = 30;
if (c > b > a)
        printf("TRUE");
else
        printf("FALSE");
return 0;
}
```

Answer : FALSE

4. Assume size of an integer as 4 bytes. What will be the output of the C program?

```
#include <stdio.h>
int main()
{
        int i = 5, j = 10, k = 15;
        printf("%d ", sizeof(k /= i + j));
        printf("%d", k);
        return 0;
}
```

Answer : 4 15

1. What will be the output of the C program?

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

Notes

```
int public = 5;
int private = 10;
int protected = 15;
class = public + private + protected;
printf("%d",class);
return 0;
}
```

Answer : 5

2. Write a program in C to show the behavior of relational operators.
3. Write a program in C to show the behavior of logical operators.
4. Write a program in C to show the behavior of assignment operators.

Model III: Fundamental Features in C

Notes

Course Content:

- C Statements
- conditional executing using if, else, nesting of if, switch and break
- Concepts of loops, example of loops in C using for, while and do-while, continue and break
- Storage types (automatic, register etc.)
- predefined processor
- Command Line Argument

Key Learning Objective :

1. Better understanding of C statements.
2. Concept of various control statements of C
3. Concept of different types of loops
4. Understanding of continue and break statements
5. Understanding of concept of Storage Types
6. Concept of fundamentals as pre processors and command line argument.
7. Command line argument concept

Notes

UNIT 3.1: C Statements

Unit Outcomes :

At the end of this unit, you will be able to:

1. Understanding of if-else statement
2. Concept of switch-case and break
3. Concept of different types of loops
4. Understanding of Storage types
5. Concept of predefined processor
6. Understanding of command line arguments
7. Apply the concept in developing applications

3.1.1 C Statements

A statement is a command that user gives to the computer in order to get an output and that command instructs the computer to take a specific action, such as display to the screen, or collect input. A computer program is made up of a series of statements.

C statements can be of Labeled statement, Compound statement, Expression statement, Selection statements, Iteration statements and Jump statements.

Labeled Statement

When a statement is preceded by a label, it is known as labeled statement. A simple identifier followed by : (colon) is a label. Generally we use label in goto statement and switch-case statements

Compound Statement

When multiple statements is grouped into a single statements, it is known as compound statement. For example, body of function can be consider as compund statement.

Expression Statement

Expression statement consists of an expression followed by ; (semi colon), if no expression is there then it is known as null statement. Generally, statement may have a value when expression is present.

Example:

printf(“Hello C language”); is an expression statement.

Selection Statement

Under this division, if, if..else and switch statement fall. Which we are going to learn in the next part.

Iteration Statement

Generally iteration statements consists of for loop, while loop and do..while loop.

Jump Statement

C language has jump statements by which user can jump from one part of the program to another part and to avail this facility C uses goto, break, continue and return expression.

3.1.2 Conditional executing

Conditional statements instructs a computer to execute a certain block of code till a certain condition has been met.

3.1.2.1 Basic if Syntax

The if statement permits to control if a program arrives a section of code or not based on whether a given condition is true or false. One of the crucial functions of the if statement is that it agrees the program to decide on an action based upon the user's input.

The structure of an if statement is as follows:

```
if ( statement is TRUE )

   Execute this line of code
```

Here is a simple example that shows the syntax:

```
if ( 5 < 10 )

   printf( "Five is now less than ten, that's a big surprise" );
```

else syntax

Every So Often when the condition in an if statement calculates to false, it would be wonderful to execute some code instead of the code executed when the statement appraises to true. The "else" statement efficiently declares that whatever code after it (whether a single line or code between brackets) is executed if the if statement is False.

It looks like this:

```
if ( TRUE ) {

   /* Execute these statements if TRUE */

}

else {

   /* Execute these statements if FALSE */

}
```

else if syntax

An Additional usage of else is when there are multiple conditional statements that may all assess to true, yet you want only one if statement's body to execute. You can make use of an "else if" statement following an if statement and its body; that way, if the first statement is true, the "else if" will be overlooked, but if the if statement is false, it will then check the condition for the else if statement. If the if statement was true, the else statement will not be verified. It is conceivable to use numerous else if statements to ensure that only one block of code is executed.

Notes

Let's look at a simple program for you to try out on your own.

```
#include <stdio.h>
int main() {
    int number1, number2;
    printf("Enter two integers: ");
    scanf("%d %d", &number1, &number2);

    //checks if the two integers are equal.
    if(number1 == number2) {
        printf("Result: %d = %d",number1,number2);
    }

    //checks if number1 is greater than number2.
    else if (number1 > number2) {
        printf("Result: %d > %d", number1, number2);
    }

    //checks if both test expressions are false
    else {
        printf("Result: %d < %d",number1, number2);
    }

    return 0;
}
```

Output:

```
Enter two integers: 12
23
```

Result: 12 < 23

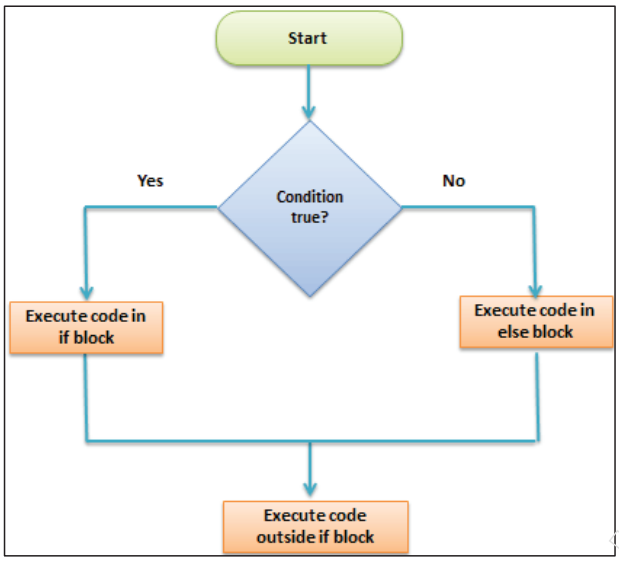


Fig 3.1: Flow chart of conditional statement

```
if (test-expression)
{
   //will be executed for True block of statements
}
else
{
   //will be executed for False block of statements
}
Statements;
```

Nested if-else Statements

When a sequence of decision is required, nested if-else is applied. Nesting implies using one if-else construct within another one.

Let us create a program to illustrate the use of nested if-else.

```
#include<stdio.h>
int main()
{
        int num=1;
        if(num<10)
        {
```

Notes

```
                    if(num==1)
                    {
                        printf("The value is:%d\n",num);
                    }
                    else
                    {
                        printf("The value is greater than 1");
                    }
            }
            else
            {
                    printf("The value is greater than 10");
            }
            return 0;
}
```

3.2.2 switch statement

A switch statement permits a variable to be tested against numerous values for equality. Each of these values is called a case. As Soon As a matching case is found, its block of code is executed. It is an complementary to the more common if-else statement

```
switch(expression)
{
  case constant_1 :
    // Code to be executed if expression == constant_1
    break;

  case constant_2 :
    // Code to be executed if expression == constant_2;
    break;
   default : // the default case is optional
    // Code to be executed if none of the cases match.
}
```

There are several rules to keep in mind while writing switch statements:

- The expression in the switch can be a variable or an expression - but it ought to be an integer or a character.
- You can have any quantity of cases however there should not be any reproductions. Switch statements can also be nested within each other.
- The optional default case is executed when none of the cases above match.
- The break statement is utilized to break the flow of control once a case block is executed. While it is not obligatory, without it, all subsequent cases after the matching case will also get executed. Think About the code below to get a clearer idea:

Example:

```
int main() {
  int var = 10;
  switch (var)
    {
      case 5:
         printf(“Case 1 executed.”);
         // break;
      case 10:
         printf(“Case 2 executed. “);
         // break;
      case 15:
         printf(“Case 3 executed. “);
         // break;
      case 20:
         printf(“Case 4 executed. “);
         // break;
      default:
         printf(“Default case executed. “);
    }
}
```

Output : Case 2 executed

```
// Program to create a simple calculator
#include <stdio.h>
```

Notes

```
int main() {
   char operator;
   double n1, n2;
   printf("Enter an operator (+, -, *, /): ");
   scanf("%c", &operator);
   printf("Enter two operands: ");
   scanf("%lf %lf",&n1, &n2);
   switch(operator)
   {
      case '+':
         printf("%.1lf + %.1lf = %.1lf",n1, n2, n1+n2);
         break;

      case '-':
         printf("%.1lf - %.1lf = %.1lf",n1, n2, n1-n2);
         break;

      case '*':
         printf("%.1lf * %.1lf = %.1lf",n1, n2, n1*n2);
         break;

      case '/':
         printf("%.1lf / %.1lf = %.1lf",n1, n2, n1/n2);
         break;
      // operator doesn't match any case constant +, -, *, /
      default:
         printf("Error! operator is not correct");
   }
   return 0;
}
```

Output:

Enter an operator (+, -, *,): -

Enter two operands: 32.5

12.4

32.5 - 12.4 = 20.1

3.1.3 Concept of Loops

A Loop implements the sequence of statements several times until the stated condition becomes false. A loop comprises of two parts, a body of a loop and a control statement. The control statement is a blend of some conditions that direct the body of the loop to execute until the specified condition becomes false. The objective of the loop is to replicate the same code a number of times.

Hinge On the position of a control statement in a program, a loop is categorized into two types:

1. Entry controlled loop
2. Exit controlled loop

In an entry-controlled loop, a condition is examined before executing the body of a loop. It is also termed as a pre-checking loop.

In an exit-controlled loop, a condition is verified after executing the body of a loop. It is also labeled as a post-checking loop.

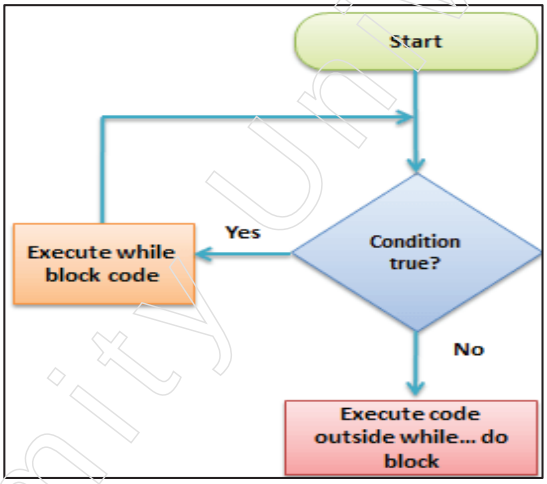


Fig 3.2: Flow chart of while loop execution

'C' programming language affords us with three types of loop constructs:

1. The while loop

2. The do-while loop

3. The for loop

Notes

3.1.3.1 Examples of Loops

Let us look some examples of loops in order to make our concept more clear.

3.1.3.1.1 while Loop and do..while loop

A while loop is the highly straightforward looping structure. The essential format of while loop is as follows:

```
while (condition) {
        statements;
}
```

It is an entry-controlled loop. In while loop, a condition is calculated before processing a body of the loop. If a condition is true then and only then the body of a loop is completed. After the body of a loop is executed then control again goes back at the beginning, and the condition is checked if it is true, the identical process is executed up until the condition becomes false. Once the condition becomes false, the control turns out of the loop.

Example:

```
#include<stdio.h>
#include<conio.h>
int main()
{
        int num=1;              //initializing the variable
        while(num<=10)          //while loop with condition
        {
                                printf("%d\n",num);
                                num++;                          //incrementingoperation
        }
        return 0;
}
```

do..while loop

A do-while loop is like the while loop apart from that the condition is constantly executed after the body of a loop. It is also termed an exit-controlled loop.

The basic format of while loop is as follows:

```
 do {
```

```
  statements
} while (expression);
```

As we noticed in a while loop, the body is executed if and only if the condition is true. In certain cases, we must execute a body of the loop at least one time even if the condition is false. This kind of operation can be achieved by using a do-while loop.

In the do-while loop, the body of a loop is continually executed at least once. After the body is executed, then it verifies the condition. If the condition is true, then it will once again execute the body of a loop otherwise control is shifted out of the loop.

Example:

```
#include<stdio.h>
#include<conio.h>
int main()
{
        int num=1;              //initializing the variable
        do                      //do-while loop
        {
                                printf("%d\n",2*num);
                                num++;                  //incrementingoperation
        }while(num<=10);
        return 0;
}
```

3.1.3.1.2 for loop

A for loop is a further effective loop structure in 'C' programming. The general structure of for loop is as follows:

```
for (initial value; condition; incrementation or decrementation )
{
  statements;
}
```

- The initial value of the for loop is staged only once.
- The condition is a Boolean expression that scans and contrasts the counter to a fixed value following each iteration, stopping the for loop when false is returned.
- The incrementation/decrementation boosts (or decreases) the counter by a set value.

Notes

Example:

```
            // Print numbers from 1 to 10
#include <stdio.h>


int main() {
  int i;


  for (i = 1; i < 11; ++i)
  {
    printf("%d ", i);
  }
  return 0;
}
```

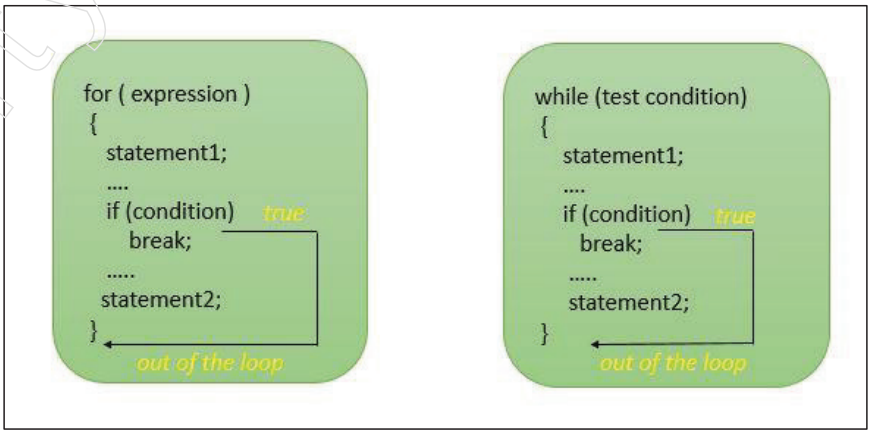
Output:

1 2 3 4 5 6 7 8 9 10

3.1.3.1.3 continue and break statements in C language

In any loop break is utilized to jump out of loop hopping the code below it without caring about the test condition.

It disrupts the flow of the program by breaking the loop and resumes the execution of code which is outside the loop.



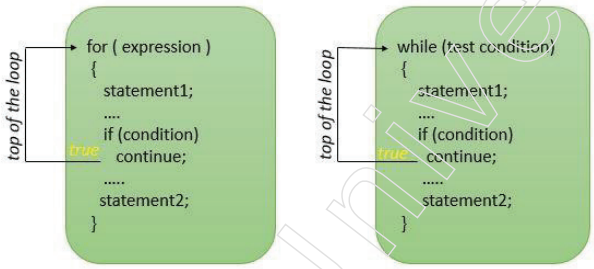
Syntax:

```
while (test_condition)
{
  statement1;
  if (condition )
     break;
  statement2;
}
```

Like a break statement, continue statement is also utilized with if condition inside the loop to alter the flow of control.

When used in while, for or do...while loop, it avoids the remaining statements in the body of that loop and performs the next iteration of the loop.

Unlike break statement, continue statement when encountered does not cease the loop, rather interrupts a specific iteration.




```
while (test_condition)
{
  statement1;
  if (condition )
     continue;
  statement2;
}
```

3.1.4 Storage Classes in C

A storage class exemplifies the visibility and a location of a variable. It tells from what part of code we can access a variable. A storage class is utilized to describe the following things:

- The variable scope.

Notes

- The place where the variable will be stored.
- The initialized value of a variable.
- A lifespan of a variable.

3.1.4.1 Automatic

- Automatic variables are allocated memory automatically at runtime.
- The visibility of the automatic variables is limited to the block in which they are defined.
- The scope of the automatic variables is limited to the block in which they are defined.
- The automatic variables are initialized to garbage by default.
- The memory assigned to automatic variables gets freed upon exiting from the block.
- The keyword used for defining automatic variables is auto.
- Every local variable is automatic in C by default.

3.1.4.2 Static

- The variables defined as static specifier can hold their value between the multiple function calls.
- Static local variables are visible only to the function or the block in which they are defined.
- A same static variable can be declared many times but can be assigned at only one time.
- Default initial value of the static integral variable is 0 otherwise null.
- The visibility of the static global variable is limited to the file in which it has declared.
- The keyword used to define static variable is static.

3.1.4.3 Register

- The variables defined as the register is allocated the memory into the CPU registers depending upon the size of the memory remaining in the CPU.
- We can not dereference the register variables, i.e., we can not use &operator for the register variable.
- The access time of the register variables is faster than the automatic variables.
- The initial default value of the register local variables is 0.
- The register keyword is used for the variable which should be stored in the CPU register. However, it is compilers choice whether or not; the variables can be stored in the register.
- We can store pointers into the register, i.e., a register can store the address of a variable.
- Static variables can not be stored into the register since we can not use more than one storage specifier for the same variable.

3.1.4.4 External

- The external storage class is used to tell the compiler that the variable defined as extern is declared with an external linkage elsewhere in the program.
- The variables declared as extern are not allocated any memory. It is only declaration and intended to specify that the variable is declared elsewhere in the program.
- The default initial value of external integral type is 0 otherwise null.
- We can only initialize the extern variable globally, i.e., we can not initialize the external variable within any block or method.
- An external variable can be declared many times but can be initialized at only once.
- If a variable is declared as external then the compiler searches for that variable to be initialized somewhere in the program which may be extern or static. If it is not, then the compiler will show an error.

3.1.5 Pre-processor in C

Pre-processor processes program prior to going to the compiler. You may not get it right now but will soon comprehend what it means.

Pre-processor directives commence with a hash symbol (#). Preprocessor directives are preprocessor commands.

The most common preprocessor directive is

#include

Here, #include is a preprocessor directive and it renders header file like “stdio.h” available for us.

Let us take one more example of a pre-processor directive.

Here, #define is a pre-processor directive. It is employed to define something, like here we have defined that PI is 3.14. So, before going to the compiler, if ‘PI’ comes anywhere in the program, it will be replaced by 3.14.

It is dissimilar from normal variables because normal variables are processed in the course of compilation. However, anything defined in #define is substituted with its value before compilation.

3.1.6 Command Line Arguments

It is feasible to pass some values from the command line to your C programs when they are executed. These values are labeled command line arguments and several times they are vital for the program notably when you want to control your program from elsewhere instead of hard coding those values within the code.

The command line arguments are processed using main() function arguments where argc refers to the number of arguments passed, and argv[] is a pointer array which points to every single argument passed to the program. Following is a simple example which verifies if there is any argument delivered from the command line and take action appropriately.

Example:

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

Notes

```
int main( int argc, char *argv[] )  {

  if( argc == 2 ) {
    printf("The argument supplied is %s\n", argv[1]);
  }
  else if( argc > 2 ) {
    printf("Too many arguments supplied.\n");
  }
  else {
    printf("One argument expected.\n");
  }
}
```

Storage class	Purpose
auto	It is a default storage class.
extern	It is a global variable.
static	It is a local variable which can return a value even when control is transferred to the function call.
register	It is a variable which is stored inside a Register.

Activities :

1. What will be the output of the C program?

```
int main()
{
  int a = 0, i = 0, b;
  for (i = 0;i < 5; i++)
  {
    a++;
    continue;
  }
}
```

Output : 5

2. What will be the output of the C program?

```
int main()
  {
     int a = 0, i = 0, b;
     for (i = 0;i < 5; i++)
     {
        a++;
        if (i == 3)
           break;
     }
  }
```

Output : 4

3. What will be the output of the C program?

```
#include<stdio.h>
int main()
{
        int i;
        while(0, i < 4)
        {
                                printf(“Loop “);
                                i++;
        }
        return 0;
}
```

4. What will be the output of the C program?

```
int main()
  {
    int i = 0;
    do
    {
```

Notes

Notes

```
        i++;
        if (i == 2)
            continue;
            printf("In while loop ");
    } while (i < 2);
    printf("%d\n", i);
}
```

Output : In while loop 2

5. Write a short note on pre processor directives in C.
6. Define different types of registers.
7. What do you mean by command line arguments.

Model IV: Arrays and Functions

Notes

Course Content:

- Basic concept of array
- Use of two dimensional arrays in matrix
- Concept of sub programming
- Concept of functions
- Function prototype
- Concept of returning of values from functions
- Passing of arguments in a function
- Recursion

Key Learning Objective :

1. Understanding of concept of Arrays
2. Better understanding of one dimensional and two dimensional array
3. Understanding of functions
4. Concept of function prototypes
5. Understanding of how function is returning values
6. Concept of passing arguments
7. Concept of recursive function

Notes

UNIT 4.1: Arrays and Functions

Unit Outcomes :

At the end of this unit, you will be able to:

1. Define arrays
2. Define function declaration
3. State function prototype
4. Define function with various types of arguments
5. State recursion

4.1.1 Arrays and Functions

An array is a assemblage of one or more values of the identical type. Each value is recognized as an element of the array. The elements of the array contribute to the same variable name but each element has its own distinctive index number (also known as a subscript). An array can be of any type, For example: int, float, char etc. If an array is of type int then it's elements must be of type int only.

To store roll no. of 100 students, we have to declare an array of size 100 i.e roll_no[100]. Here size of the array is 100 , so it is efficient of storing 100 values. In C, index or subscript starts from 0, so roll_no[0] is the first element, roll_no[1] is the second element and so on. Note that the very last element of the array will be at roll_no[99] not at roll_no[100] because the index starts at 0.

Arrays can be single or multidimensional. The number of subscript or index defines the dimensions of the array. An array of one dimension is recognized as a one-dimensional array or 1-D

4.1.2 One-dimensional array

Theoretically you can think of a one-dimensional array as a row, where elements are stored one after another.

Syntax: datatype array_name[size];

datatype: It implies the type of the elements in the array.

array_name: Name of the array. It must be a valid identifier.

size: Number of elements an array can hold. here is some example of array declarations:

1

2

3 int num[100];

float temp[20];

char ch[50];

The elements of an array can be retrieved by specifying array name followed by subscript or index inside square brackets (i.e []). Array subscript or index begins at 0.

If the size of an array is 10 then the first element is at index 0, while the last element is at index 9. The first legitimate subscript (i.e 0) is known as the lower bound, while last legitimate subscript is known as the upper bound.

int my_arr[5];

The following program utilizes for loop to take input and print elements of a 1-D array.

Example:

```
#include<stdio.h>

int main()

{

   int arr[5], i;


   for(i = 0; i < 5; i++)

   {

      printf(“Enter a[%d]: “, i);

      scanf(“%d”, &arr[i]);

   }

   printf(“\nPrinting elements of the array: \n\n”);

   for(i = 0; i < 5; i++)

   {

      printf(“%d “, arr[i]);

   }

   // signal to operating system program ran fine

   return 0;

}
```

Output:

```
Enter a[0]: 11

Enter a[1]: 22

Enter a[2]: 34

Enter a[3]: 4

Enter a[4]: 34

Printing elements of the array:

11 22 34 4 34
```

Notes

When an array is proclaimed inside a function, the elements of the array have garbage value. If an array is global or static, then its elements are automatically initialized to 0. We can unequivocally initialize elements of an array at the time of declaration using the following syntax:

Syntax:

datatype array_name[size] = { val1, val2, val3, ... valN };

datatype is the type of elements of an array.

array_name is the variable name, which ought to be any valid identifier.

size is the size of the array.

val1, val2 ... are the constants known as initializers. Each value is split up by a comma(,) and then there is a semi-colon (;) after the closing curly brace (}).

Example:

The following program finds the highest and lowest elements in an array.

```
#include<stdio.h>

#define SIZE 10

int main()

{

   int my_arr[SIZE] = {34,56,78,15,43,71,89,34,70,91};

   int i, max, min;

   max = min = my_arr[0];

   for(i = 0; i < SIZE; i++)

   {

      // if value of current element is greater than previous value

      // then assign new value to max

      if(my_arr[i] > max)

      {

         max = my_arr[i];

      }


      // if the value of current element is less than previous element

      // then assign new value to min

      if(my_arr[i] < min)

      {

         min = my_arr[i];
```

```
    }
  }
printf("Lowest value = %d\n", min);
printf("Highest value = %d", max);
  // signal to operating system everything works fine
  return 0;
}
```

Output :

Lowest value = 15

Highest value = 91

4.1.3 Two Dimensional Array in C

The two-dimensional array can be described as an array of arrays. The 2D array is organized as matrices which can be exemplified as the collection of rows and columns. However, 2D arrays are established to implement a relational database lookalike data structure. It offers ease of holding the bulk of data at once which can be passed to any number of functions wherever needed.

The syntax to declare the 2D array is given below.

1. data_type array_name[rows][columns];
2. int twodimen[4][3];

Here, 4 is the number of rows, and 3 is the number of columns.

Example:

Two-dimensional array example in C

```
#include<stdio.h>
int main(){
int i=0,j=0;
int arr[4][3]={{1,2,3},{2,3,4},{3,4,5},{4,5,6}};
//traversing 2D array
for(i=0;i<4;i++){
 for(j=0;j<3;j++){
   printf("arr[%d] [%d] = %d \n",i,j,arr[i][j]);
 }//end of j
}//end of i
return 0;
}
```

Notes

4.1.3.1 2D array example:

```
#include<stdio.h>
#define ROW 2
#define COL 3

int main()
{
   int mat1[ROW][COL], mat2[ROW][COL], mat3[ROW][COL];
   int i, j;

   printf("Enter first matrix: \n\n");

   for(i = 0; i < ROW; i++)
   {
      for(j = 0; j < COL; j++)
      {
         printf("Enter a[%d][%d]: ", i, j);
         scanf("%d", &mat1[i][j]);
      }
   }

   printf("\nEnter Second matrix: \n\n");

   for(i = 0; i < ROW; i++)
   {
      for(j = 0; j < COL; j++)
      {
         printf("Enter a[%d][%d]: ", i, j);
         scanf("%d", &mat2[i][j]);
      }
   }
```

Notes

```
    // add mat1 and mat2

    for(i = 0; i < ROW; i++)
    {
        for(j = 0; j < COL; j++)
        {
            mat3[i][j] = mat1[i][j] + mat2[i][j] ;
        }
    }

    printf("\nResultant array: \n\n");

    // print resultant array

    for(i = 0; i < ROW; i++)
    {
        for(j = 0; j < COL; j++)
        {
            printf("%5d ", mat3[i][j]);
        }
        printf("\n");
    }
    // signal to operating system program ran fine
    return 0;
}
```

Output:

Enter first matrix:

Enter a[0][0]: 12

Enter a[0][1]: 32

Enter a[0][2]: 13

Enter a[1][0]: 35

Notes

Enter a[1][1]: 54

Enter a[1][2]: 35

Enter Second matrix:

Enter a[0][0]: 57

Enter a[0][1]: 64

Enter a[0][2]: 58

Enter a[1][0]: 72

Enter a[1][1]: 84

Enter a[1][2]: 29

Resultant array:

mat1 + mat2 =

69 96 71

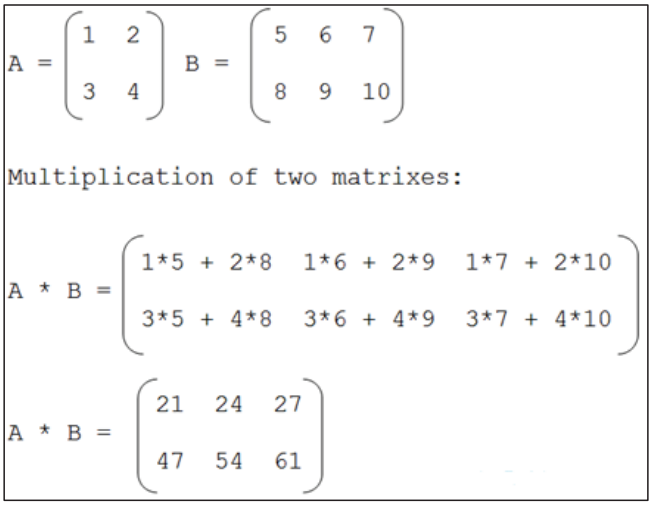
107 138 64

Matrix multiplication in C

We can add, subtract, multiply and divide 2 matrices. To do so, we are taking input from the user for row number, column number, first matrix elements and second matrix elements. Then we are performing multiplication on the matrices entered by the user.

In matrix multiplication first matrix one row element is multiplied by second matrix all column elements.

Let's try to understand the matrix multiplication of 2*2 and 3*3 matrices by the figure given below:



Example:

```
#include<stdio.h>
#include<stdlib.h>
int main(){
int a[10][10],b[10][10],mul[10][10],r,c,i,j,k;
system("cls");
printf("enter the number of row=");
scanf("%d",&r);
printf("enter the number of column=");
scanf("%d",&c);
printf("enter the first matrix element=\n");
for(i=0;i<r;i++)
{
for(j=0;j<c;j++)
{
scanf("%d",&a[i][j]);
}
}
printf("enter the second matrix element=\n");
for(i=0;i<r;i++)
{
for(j=0;j<c;j++)
{
scanf("%d",&b[i][j]);
}
}

printf("multiply of the matrix=\n");
for(i=0;i<r;i++)
{
for(j=0;j<c;j++)
{
```

Notes

Notes

```
mul[i][j]=0;
for(k=0;k<c;k++)
{
mul[i][j]+=a[i][k]*b[k][j];
}
}
}
//for printing result
for(i=0;i<r;i++)
{
for(j=0;j<c;j++)
{
printf("%d\t",mul[i][j]);
}
printf("\n");
}
return 0;
}
```

4.1.4 Function

A function is a cluster of statements that together accomplish a task. Every C program has at least one function, which is main(), and all the most trivial programs can define additional functions.

Code can be separated into separate functions. How you divide up your code among various functions is up to you, but rationally the division is such that each function presents a specific task.

A function declaration informs the compiler about a function's name, return type, and parameters. A function definition delivers the actual body of the function.

Functions are used because of following reasons –

a. To enhance the readability of code.
b. Expands the reusability of the code, same function can be utilized in any program rather than writing the equivalent code from scratch.
c. Debugging of the code would be simpler if you use functions, as errors are uncomplicated to be traced.
d. Decreases the size of the code, identical set of statements are substituted by function calls.

The general form of a function definition in C programming language is as follows:

```
return_type function_name( parameter list ) {
   body of the function
}
```

- **Return Type −** A function may send back a value. The return_type is the data type of the value the function returns. Some functions present the desired operations without returning a value. In this case, the return_type is the keyword void.
- **Function Name −** This is the authentic name of the function. The function name and the parameter list collectively constitute the function signature.
- **Parameters −** A parameter is like a placeholder. When a function is stated, you pass a value to the parameter. This value is referred to as actual parameter or argument. The parameter list implies to the type, order, and number of the parameters of a function. Parameters are not obligatory; that is, a function may contain no parameters.
- **Function Body −** The function body encompasses a collection of statements that describe what the function does.

4.1.4.1 Function Declarations

A function declaration tells the compiler about a function name and how to call the function. The actual body of the function can be defined separately.

A function declaration has the following parts −

```
return_type function_name( parameter list );
```

For the above defined function max(), the function declaration is as follows −

```
int max(int num1, int num2);
```

4.1.4.2 Function Prototype

The Function prototype serves the following purposes –

1. It reveals the return type of the data that the function will return.
2. It informs the number of arguments passed to the function.
3. It informs the data types of the each of the passed arguments.
4. Also it describes the order in which the arguments are passed to the function.

4.1.4.3 Calling function

While designing a C function, provide a definition of what the function must do. To use a function, you will have to call that function to perform the defined task.

When a program requests a function, the program control is shifted to the called function. A called function implements a defined task and when its return statement is executed or when its function-ending closing brace is reached, it returns the program control back to the main program.

To call a function, pass the requisite parameters along with the function name, and if the function returns a value, then you can store up the returned value.

Notes

Example:

```
#include <stdio.h>

/* function declaration */
int max(int num1, int num2);

int main () {

   /* local variable definition */
   int a = 100;
   int b = 200;
   int ret;

   /* calling a function to get max value */
   ret = max(a, b);

   printf( “Max value is : %d\n”, ret );

   return 0;
}

/* function returning the max between two numbers */
int max(int num1, int num2) {

   /* local variable declaration */
   int result;

   if (num1 > num2)
      result = num1;
   else
      result = num2;
```

```
  return result;
}
```

Output:

Max Value is: 200

4.1.4.4 Function Arguments

If a function is to make use of arguments, it must declare variables that accept the values of the arguments. These variables are termed the formal parameters of the function.

Formal parameters perform like other local variables inside the function and are formed upon entry into the function and demolished upon exit.

While calling a function, there are two ways in which arguments can be passed to a function

Call by value

This method copies the actual value of an argument into the formal parameter of the function. In this case, changes made to the parameter inside the function have no effect on the argument.

Call by reference

This method duplicates the address of an argument into the formal parameter. Inside the function, the address is utilized to access the actual argument used in the call. This suggests that changes made to the parameter influence the argument.

Difference between Call by value and Call by Reference

- In Call by value method original value is not altered whereas, in Call by reference method, the original value is altered.
- In Call by value, a copy of the variable is passed whereas in Call by reference, a variable itself is passed.
- In Call by value, actual and formal arguments will be generated in different memory locations whereas in Call by reference, actual and formal arguments will be created in the same memory location.
- Call by value is the default method in programming languages like C++, PHP, Visual Basic NET, and C# whereas Call by reference is supported only Java language.
- Call by Value, variables are passed using a straightforward method whereas Call by Reference, pointers are required to store the address of variables.

Local Variables

Variables that are announced inside a function or block are termed local variables. They can be utilized only by statements that are inside that function or block of code. Local variables are not known to functions beyond their own. The following example discloses how local variables are used. Here all the variables a, b, and c are local to main() function.

Notes

Global Variables

Global variables are specified outside a function, generally on top of the program. Global variables retain their values throughout the life cycle of the program and they can be retrieved inside any of the functions identified for the program.

A global variable can be retrieved by any function. That is, a global variable is obtainable for use throughout the complete program after its declaration.

Swapping numbers using Function Call by Value

```
#include <stdio.h>

void swapnum( int var1, int var2 )

{

  int tempnum ;


  /*Copying var1 value into temporary variable */

  tempnum = var1 ;


  /* Copying var2 value into var1*/

  var1 = var2 ;


  /*Copying temporary variable value into var2 */

  var2 = tempnum ;


}


int main( )

{

   int num1 = 35, num2 = 45 ;

   printf("Before swapping: %d, %d", num1, num2);


   /*calling swap function*/

   swapnum(num1, num2);

   printf("\nAfter swapping: %d, %d", num1, num2);

}
```

Output:

Before Swapping: 35 , 45

After Swapping: 45 ,35

Example of Function call by Reference

```
#include <stdio.h>
void increment(int  *var)
{
    /* Although we are achieving the increment on variable
     * var, however the var is a pointer that holds the address
     * of variable num, which means the increment is actually done
     * on the address where value of num is stored.
     */
    *var = *var+1;
}
int main()
{
     int num=20;
     /* This way of calling the function is known as call by
      * reference. Instead of passing the variable num, we are
      * passing the address of variable num
      */
     increment(&num);
     printf("Value of num is: %d", num);
   return 0;
}
```

4.1.4.5 Recursion

Recursion is the process of duplicating items in a self-similar way. In programming languages, if a program permits you to call a function inside the same function, then it is described a recursive call of the function. Recursive functions are very beneficial to solve many mathematical problems, such as calculating the factorial of a number, generating Fibonacci series, etc.

Notes

The following example calculates the factorial of a given number using a recursive function.

```
#include <stdio.h>


unsigned long long int factorial(unsigned int i) {


  if(i <= 1) {
    return 1;
  }
  return i * factorial(i - 1);
}


int  main() {
  int i = 3;
  printf(“Factorial of %d is %d\n”, i, factorial(i));
  return 0;
}
```

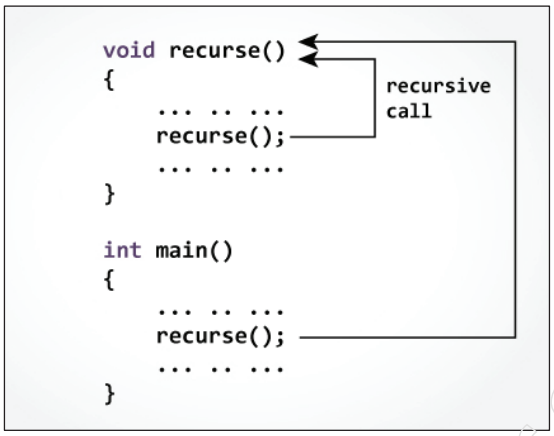
Output :

```
Factorial of 3 is : 6
```

4.3.5.1 How recursion works?

```
void recurse()
{
   ... .. ...
   recurse();
   ... .. ...
}


int main()
```

```
{
   ... .. ...
   recurse();
   ... .. ...
}
```



Activities :

1. What will be the output of the C program?

```
#include <stdio.h>
void foo(int*);
int main()
{
   int i = 10, *p = &i;
   foo(p++);
}
void foo(int *p)
{
   printf("%d\n", *p);
}
```

Answer : 10

Notes

2. What will be the output of the C program?

```
#include <stdio.h>
int main()
{
    int i = 97, *p = &i;
    foo(&i);
    printf("%d ", *p);
}
void foo(int *p)
{
    int j = 2;
    p = &j;
    printf("%d ", *p);
}
```

Output: 2 97

3. What will be the code to print 5 contains in a[4][1][0]?

```
#include<stdio.h>
int main()
{
        int a[1][2][3] = {0};
        a[0][1][2] = 5;
        printf("%d",*(*(*(a+0)+1)+2));
        return 0;
```

4. What will be the output of the C program?

```
#include<stdio.h>
int main()
{
        char *ptr;
        char string[] = "This is C Programming ";
```

Notes

```
        ptr = string;
        ptr += 6;
        printf("%s",ptr);
        return 0;
}
```

Answer : C Programming

5. What will be the output of the C program?

```
#include<stdio.h>
int main()
{
        function();
        return  0;
}
void function()
{
        printf("Function in C is awesome");
}
```

Answer : Function in C is awesome

6. What will be the output of the C program?

```
#include <stdio.h>
enum m{JAN, FEB, MAR};
enum m foo();
int main()
{
    enum m i = foo();
    printf("%d\n", i);
}
int  foo()
```

Notes

```
{
    return JAN;
}
```

Answer : Compile Time Error

7. What will be the output of the C program?

```
#include <stdio.h>
void m(int k)
{
    printf("hi");
}
void m(double k)
{
    printf("hello");
}
void main()
{
    m(3);
}
```

Answer : Compile Time Error

8. What will be the output of the C program?

```
void foo();
int main()
{
void foo(int);
foo(1);
return 0;
}
void foo(int i)
{
```

printf(“2 “);

}

Answer : 2

1. Write a program in C to find the square of any number using the function
2. Write a program in C to check a given number is even or odd using the function.
3. Write a program in C to convert decimal number to binary number using the function
4. Write a program in C to get the largest element of an array using the function
5. Write a program in C to check armstrong and perfect numbers using the function

Notes

Notes

Model V: Advanced features in C

Course Content:

- Pointers
- Relationship between arrays and pointers
- Argument passing using pointers
- Array of pointers
- Passing arrays as arguments
- Strings and C string library
- Structure and Union
- Defining C structures
- Placing values to members
- Array of structure
- Nested structure
- Passing strings as arguments
- File Handling

Key Learning Objective :

1. Understanding of basic concept of pointers and structures
2. Concept of passing array as argument
3. Understanding of String handling in C language
4. Concept of structures
5. Understand and apply various types file handling mechanism
6. Get familiar with Structure and Union

UNIT 5.1: Pointers

Notes

Unit Outcomes :

At the end of this unit, you will be able to:

1. Define pointers
2. Define pointer as argument
3. State function returning pointer variable
4. Use pointer to functions
5. Define how to create and use structure
6. State how to create union
7. Define the types of Files and usage

5.1.1 Pointers

A pointer is a variable that stores the address of another variable. Dissimilar other variables that hold values of a certain type, pointer occupies the address of a variable. For instance, an integer variable possesses (or you can say stores) an integer value, however an integer pointer holds the address of a integer variable.

In the following program, we have a variable num of int type. The value of num is 10 and this value ought to be stored somewhere in the memory, right? A memory space is assigned for each variable that retains the value of that variable, this memory space has an address.

For example we stay in a house and that has an address, which facilitates other people to find our house. The similar way the value of the variable is stored in a memory address, which facilitates the C program to find that value when it is required.

Example:

```
#include <stdio.h>

int main()

{

  int num = 12;

  printf(“Value of variable num is: %d”, num);

 /* To print the address of a variable we use %p format specifier and ampersand (&) sign just before the variable name like &num.

  */

  printf(“\nAddress of variable num is: %p”, &num);

  return 0;

}
```

Notes

Output:

Value of variable num is: 12

Address of variable num is : 2293340

5.1.1.1 Operators that are used with Pointers

"Address of"(&) Operator

We have previously seen in the first example that we can exhibit the address of a variable using ampersand sign. &num is used to retrieve the address of variable num. The & operator is also identified as "Address of" Operator.

printf("Address of var is: %p", &num);

"Value at Address"(*) Operator

The * Operator is also known as Value at address operator.

5.1.1.2 How to declare a pointer?

```
int *p1     /*Pointer to an integer variable*/
double *p2                    /*Pointer to a variable of data type double*/
char *p3                      /*Pointer to a character variable*/
float *p4                     /*pointer to a float variable*/
```

Example of Pointer demonstrating the use of & and *

```
#include <stdio.h>
int main()
{
  //Pointer of integer type, this can hold the address of a integer type variable.
  int *p;

  int var = 10;

  /* Assigning the address of variable var to the pointer * p. The p can hold the address of var because var is an integer type variable.
 */
  p= &var;
  printf("Value of variable var is: %d", var);
```

```
   printf("\nValue of variable var is: %d", *p);
   printf("\nAddress of variable var is: %p", &var);
   printf("\nAddress of variable var is: %p", p);
   printf("\nAddress of pointer p is: %p", &p);
   return 0;
}
```

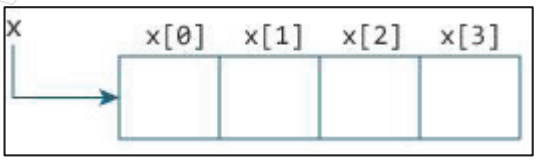
5.1.2 Arrays and Pointers

An array is a block of sequential data. Let us write a program to publish addresses of array elements.

```
#include <stdio.h>
int main() {
   int x[4];
   int i;

   for(i = 0; i < 4; ++i) {
      printf("&x[%d] = %p\n", i, &x[i]);
   }

   printf("Address of array x: %p", x);

   return 0;
}
```

There is a disparity of 4 bytes between two consecutive elements of array x. It is because the size of int is 4 bytes (on our compiler).

Notice that, the address of &x[0] and x is the same. It's because the variable name x points to the first element of the array.



Example: Pointers and Arrays

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

Notes

```
int main() {
  int i, x[6], sum = 0;
  printf("Enter 6 numbers: ");
  for(i = 0; i < 6; ++i) {
  // Equivalent to scanf("%d", &x[i]);
      scanf("%d", x+i);

  // Equivalent to sum += x[i]
      sum += *(x+i);
  }
  printf("Sum = %d", sum);
  return 0;
}
```

Output:

```
Enter 6 numbers:  2
 3
 4
 4
 12
 4
Sum = 29
```

5.1.3 Pointers as Function Argument

Pointer as a function parameter is utilized to hold addresses of arguments passed through function call. This is also identified as call by reference. When a function is called by reference any adjustment made to the reference variable will alter the original variable.

Swapping two numbers using Pointer

Example:

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

```
void swap(int *a, int *b);


int main()
{
   int m = 10, n = 20;
   printf("m = %d\n", m);
   printf("n = %d\n\n", n);


   swap(&m, &n);    //passing address of m and n to the swap function
   printf("After Swapping:\n\n");
   printf("m = %d\n", m);
   printf("n = %d", n);
   return 0;
}


//    pointer 'a' and 'b' holds and points to the address of 'm' and 'n'


void swap(int *a, int *b)
{
   int temp;
   temp = *a;
   *a = *b;
   *b = temp;
}
```

Output:

After Swapping : 20 10

5.1.4 Functions returning Pointer variables

A function can likewise return a pointer to the calling function. In this case you must be cautious, because local variables of function doesn't exist outside the function. They have scope only inside the function. Therefore if you return a pointer linked to a local

Notes

variable, that pointer will be pointing to nothing when the function culminates.

Example:

```
#include <stdio.h>


int* larger(int*, int*);


void main()
{
   int a = 15;
   int b = 92;
   int *p;
   p = larger(&a, &b);
   printf("%d is larger",*p);
}


int* larger(int *x, int *y)
{
   if(*x > *y)
      return x;
   else
      return y;
}
```

Output:

92 is larger

5.1.5 Pointer to functions

It is feasible to declare a pointer pointing to a function which can be utilized as an argument in another function. A pointer to a function is proclaimed as follows,

type (*pointer-name)(parameter);

Here is an example :

```
int (*sum)();   //legal declaration of pointer to function

int *sum();     //This is not a declaration of pointer to function
```

A function pointer can point to a specific function when it is assigned the name of that function.

```
int sum(int, int);

int (*s)(int, int);

s = sum;
```

Example of Pointer to Function

```
#include <stdio.h>


int sum(int x, int y)

{

   return x+y;

}


int main( )

{

   int (*fp)(int, int);

   fp = sum;

   int s = fp(10, 15);

   printf("Sum is %d", s);


   return 0;

}
```

Output:

Sum is 25

A string in the C language is merely an array of characters. Strings ought to have a NULL or \0 character after the last character to indicate where the string ends. A string can be asserted as a character array or with a string pointer. Earliest, we take a look at a character array example:

Notes

```
char mystr[20];
```

As you can realize the character array is declared in the identical way as a normal array. This array can take in only 19 characters, because we ought to leave room for the NULL character.

Example:

```
#include<stdio.h>

    int main()
    {
                    char mystring[20];

                    mystring[0] = 'H';
                    mystring[1] = 'E';
                    mystring[2] = 'L';
                    mystring[3] = 'L';
                    mystring[4] = 'O';
                    mystring[5] = '\n';
                    mystring[6] = '\0';

                    printf("%s", mystring);

                    return 0;
    }
```

Output:

HELLO

%s is used to print a string. (The 0 without the " will in most cases also work).

String pointers are proclaimed as a pointer to a char. When there is a value allocated to the string pointer the NULL is put at the end inevitably.

Example:

```
#include<stdio.h>

int main()

{

                    char *ptr_mystring;


                    ptr_mystring = “HELLO”;

                    printf(“%s\n”, ptr_mystring);


                    return 0;

}
```

It is not feasible to read, with scanf, a string with a string pointer. You must utilize a character array and a pointer.

Example:

```
#include<stdio.h>


int main()

{

char my_array[10];

char *ptr_section2;


printf(“Type hello and enter\n”);

      scanf(“%s”, my_array);

ptr_section2 = my_array;

printf(“%s\n”, ptr_section2);


return 0;

}
```

string.h or strings.h

The C language offers no unequivocal support for strings in the language itself. The string-handling functions are applied in libraries. String I/O operations are implemented in <stdio.h> (puts , gets, etc). A set of simple string manipulation functions are employed in <string.h>, or on some systems in <strings.h>.

Notes

The string library (string.h or strings.h) has some handy functions for working with strings, like strcpy, strcat, strcmp, strlen, strcoll, etc. We will look at a few of these string operations.

strcpy

This library function is utilized to copy a string and can be applied like this: strcpy(destination, source). (It is not possible in C to do this: string1 = string2). Glance at the following example:

```
str_one = “abc”;

str_two = “def”;

strcpy(str_one , str_two); // str_one becomes “def”
```

strcmp

This library function is managed to compare two strings and can be applied like this: strcmp(str1, str2).

- If the first string is bigger than the second string a number more than null is returned.
- If the first string is less than the second string a number less than null is returned.
- If the first and the second string are identical, a null is returned.

Take look at an example:

```
printf(“Enter you name: “);

scanf(“%s”, name);

if( strcmp( name, “jane” ) == 0 )

                printf(“Hello, jane!\n”);
```

strcat

This library function concatenates a string onto the end of the new string. The result is returned. Look at the example:

```
printf(“Enter you age: “);

scanf(“%s”, age);

result = strcat( age, “ years old.” ) == 0 )

printf(“You are %s\n”, result);
```

Note: strcat() will not accomplish any boundary checking, and thus there is a threat of overrunning the strings.

strlen

This library function put back the length of a string. (All characters before the null termination.) View at the example:

```
name = “jane”;

result = strlen(name); //Will return size of four.
```

5.1.6 Structure and Union in C

In C we have container for mutually i.e. for same type data and multiple type data. Storage of data of same type, C offers concept of Array which stores data variables of identical type while for storing data of different type C has concept of structure and union that can store data variable of different type as well.

Ever Since both Structure and Union can hold various type of data in them but now based on internal implementation, we can find numerous differences in both containers.

Structure is the container well-defined in C to store data variables of different type and supports for the user defined variables storage.

On other hand Union is also similar kind of container in C which can also holds the different type of variables along with the user defined variable

5.1.6.1 How to create a structure?

‘struct’ keyword is used to create a structure. Following is an example.

```
struct address
{
  char name[50];
  char street[100];
  char city[50];
  char state[20];
  int pin;
};
```

5.1.6.2 How to declare structure variables?

A structure variable can either be declared with structure declaration or as a separate declaration like basic types.

Example:

```
// A variable declaration with structure declaration.
struct Point
```

Notes

```
{
   int x, y;
} p1;  // The variable p1 is declared with 'Point'


// A variable declaration like basic data types
struct Point
{
   int x, y;
};


int main()
{
   struct Point p1;  // The variable p1 is declared like a normal variable
}
```

5.1.6.3 How to initialize structure members?

Structure members cannot be initialized with declaration. For example, the following C program collapses in compilation.

```
struct Point
{
   int x = 0;  // COMPILER ERROR:  cannot initialize members here

 int y = 0;  // COMPILER ERROR:  cannot initialize members here
};
```

5.1.6.4 How to access structure elements?

Structure members are accessed using dot (.) operator.

Example:

```
#include<stdio.h>


struct Point
```

```
{
  int x, y;
};

int main()
{
  struct Point p1 = {0, 1};

  // Accessing members of point p1
  p1.x = 20;
  printf (“x = %d, y = %d”, p1.x, p1.y);

  return 0;
}
```

5.1.6.5 What is an array of structures?

Similar to other primitive data types, we can create an array of structures.

Example:

```
#include<stdio.h>

struct Point
{
  int x, y;
};

int main()
{
  // Create an array of structures
  struct Point arr[10];

  // Access array members
  arr[0].x = 10;
```

Notes

```
  arr[0].y = 20;


  printf("%d %d", arr[0].x, arr[0].y);
  return 0;
}
```

A union is a user-defined type related to structs in C except for one important distinction. Structs allocate sufficient space to store all its members where's unions assign the space to store merely the largest member.

5.1.7 How to define a union?

We use the union keyword to define unions. Here's an example:

```
union car
{
 char name[50];
 int price;
};
```

5.1.7.1 Create union variables

When a union is defined, it produces a user-defined type. However, no memory is allotted. To allocate memory for a given union type and work with it, we require to create variables.

Here's how we create union variables.

```
union car
{
 char name[50];
 int price;
};

int main()
{
 union car car1, car2, *car3;
 return 0;
}
```

Access members of a union

We make use of the . operator to access members of a union. To access pointer variables, we use also use the -> operator.

5.1.7.2 Example: Accessing Union Members

```
#include <stdio.h>

union Job {
  float salary;
  int workerNo;
} j;


int main() {
  j.salary = 12.3;


  // when j.workerNo is assigned a value,
  // j.salary will no longer hold 12.3
  j.workerNo = 100;


  printf("Salary = %.1f\n", j.salary);
  printf("Number of workers = %d", j.workerNo);
  return 0;
}
```

Output:

Salary =12.3

Number of workers =100

5.1.8 C Structure and Function

Here is how you can pass structures to a function

Example:

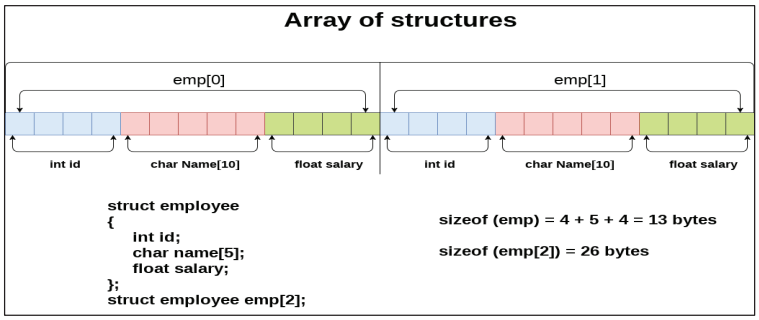
```
#include <stdio.h>
struct student {
  char name[50];
  int age;
```

Notes

```
};

// function prototype
void display(struct student s);

int main() {
   struct student s1;

   printf(“Enter name: “);

   // read string input from the user until \n is entered
   // \n is discarded
   scanf(“%[^\n]%*c”, s1.name);

   printf(“Enter age: “);
   scanf(“%d”, &s1.age);

   display(s1); // passing struct as an argument

   return 0;
}

void display(struct student s) {
   printf(“\nDisplaying information\n”);
   printf(“Name: %s”, s.name);
   printf(“Age: %s”, s.age);
}
Output :
Enter Name : Smith
Enter Age :  15
```

5.1.9 Array of Structures in C

An array of structures in C can be described as the collection of multiple structures variables where each variable comprises information about different entities. The array of structures in C are utilized to store information about multiple entities of different data types. The array of structures is also well-known as the collection of structures.



5.1.10 How Structures are stored in Memory

Members of a structure are always stored in consecutive memory locations but the memory occupied by each member may vary.

Example:

```
#include<stdio.h>


struct book

{

   char title[5];

   int year;

   double price;

};


int main()

{

   struct book b1 = {“Book1”, 1988, 4.51};


   printf(“Address of title = %u\n”, b1.title);

   printf(“Address of year = %u\n”, &b1.year);

   printf(“Address of price = %u\n”, &b1.price);


   printf(“Size of b1 = %d\n”, sizeof(b1));
```

Notes

```
    // signal to operating system program ran fine
    return 0;
}
```

5.1.11 Example using Structure and Pointer : Pointer Within Structure

```
#include<stdio.h>

struct Student
{
  int  *ptr;  //Stores address of integer Variable
  char *name; //Stores address of Character String
}s1;

int main()
{

int roll = 20;
s1.ptr   = &roll;
s1.name  = “Pritesh”;

printf(“\nRoll Number of Student : %d”,*s1.ptr);
printf(“\nName of Student       : %s”,s1.name);
return(0);
}
```

Output :

Name of Student : Pritesh

Roll Number of Student : 20

5.1.12 C File Handling

When a program is completed, the entire data is dropped. Storing in a file will safeguard your data even if the program terminates.

If you have to enter a huge number of data, it will take a lot of time to enter them all.

However, if you come up with a file containing all the data, you can effortlessly access the contents of the file applying a few commands in C.

5.1.12.1 Types of Files

When dealing with files, there are two types of files you should realize about:

1. Text files
2. Binary files

1. Text files

Text files are the normal **.txt** files. You can easily create text files using any unsophisticated text editors such as Notepad.

When you open those files, you'll go to see all the contents within the file as plain text. You can definitely edit or delete the contents.

2. Binary files

Binary files are predominantly 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).

5.1.12.2 File Operations

In C, you can execute four major operations on files, either text or binary:

1. Creating a new file
2. Opening an existing file
3. Closing a file
4. Reading from and writing information to a file

5.1.12.2.1 Mode of operations performed on a file in C language:

There are several modes in opening a file. Based on the mode of file, it can be opened for reading or writing or appending the texts. They are inventoried below.

- r – Opens a file in read mode and sets pointer to the first character in the file. It returns null if file does not exist.
- w – Opens a file in write mode. It returns null if file could not be opened. If file exists, data are overwritten.
- a – Opens a file in append mode. It returns null if file couldn't be opened.
- r+ – Opens a file for read and write mode and sets pointer to the first character in the file.
- w+ – opens a file for read and write mode and sets pointer to the first character in the file.
- a+ -- Opens a file for read and write mode and sets pointer to the first character in the file. But, it can't modify existing contents.

When employed with files, you ought to declare a pointer of type file. This declaration is essential for communication between the file and the program.

```
FILE *fptr;
```

Notes

Opening a file is achieved using the fopen() function described in the stdio.h header file.

The syntax for opening a file in standard I/O is:

ptr = fopen(“fileopen”,”mode”);

Closing a file is completed using the fclose() function.

fclose(fptr);

Example 1: Write to a text file

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

int main()
{
  int num;
  FILE *fptr;

  // make use of appropriate location if you are using MacOS or Linux
  fptr = fopen(“C:\\program.txt”,”w”);

  if(fptr == NULL)
  {
    printf(“Error!”);
    exit(1);
  }

  printf(“Enter num: “);
  scanf(“%d”,&num);

  fprintf(fptr,”%d”,num);
  fclose(fptr);
```

```
  return 0;
}
```

Example 2: Read from a text file

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

int main()
{
  int num;
  FILE *fptr;

  if ((fptr = fopen("C:\\program.txt","r")) == NULL){
      printf("Error! opening file");

      // Program exits if the file pointer returns NULL.
      exit(1);
  }

  fscanf(fptr,"%d", &num);

  printf("Value of n=%d", num);
  fclose(fptr);

  return 0;
}
```

Activity :

1. What will be the output of the C program?

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

Notes

```
int main(){
        int a = 130;
        char *ptr;
        ptr = (char *)&a;
        printf("%d ",*ptr);
        return 0;
}
```

Answer : 130

2. What will be the output of the C program?

```
#include<stdio.h>
int main()
{
        const int a = 5;
        const int *ptr;
        ptr = &a;
        *ptr = 10;
        printf("%d\n", a);
        return 0;
}
```

Answer : Address

3. What will be the output of the C program?

```
#include<stdio.h>
void function(char**);
int main()
{
        char *arr[] = { "ant", "bat", "cat", "dog", "egg", "fly" };
        function(arr);
```

```
        return 0;
}
void function(char **ptr)
{
        char *ptr1;
        ptr1 = (ptr += sizeof(int))[-2];
        printf("%s\n", ptr1);
}
```

Answer : Cat

4. What will be the output of the C program?

```
char p[20];
char *s = "string";
int length = strlen(s);
int i;
for (i = 0; i < length; i++)
    p[i] = s[length — i];
printf("%s",p);
```

Answer : No output printed

5. What will be the output of the C program?

```
#include<stdio.h>
void swap(char *str1, char *str2)
{
  char *temp = str1;
  str1 = str2;
  str2 = temp;
}
```

Notes

```
int main()
{
  char *str1 = “Holiday”;
  char *str2 = “Home”;
  swap(str1, str2);
  printf(“str1 is %s, str2 is %s”, str1, str2);
  return 0;
}
```

Answer : Holiday Home

6. Write a Program using C Structure to perform the following

Details of student 1

Name: Jim

Roll no: 10

Marks: 34.50

Enter name of student2: jack

Enter roll no of student2: 33

Enter marks of student2: 15.21

Details of student 2

Name: jack

Roll no: 33

Marks: 15.21

Details of student 3

Name: King

Roll no: 34

Marks: 25.21

7. Write a C program to create and store information in a text file.
8. Write a C program to read a file and store the lines into an array.
9. Write a program in C to find the content of the file and number of lines in a Text file.
10. Write a program in C to count a number of words and characters in a file.
11. Write a program in C to delete a specific line from a file.

Mini Project in C

Here is a project we developed as mini project in C bank management system during our first semester; it is complete and totally error-free. This project is focused on customer account services in bank, so it is named "Customer Account Bank Management System".

Here, you can create a new account, update information of an existing account, view and manage transactions, check the details of an existing account, remove existing account and view customers' list.

Overall, with this project, you can perform banking activities like in a REAL bank. Bank management mini project in C is a console application without graphics. It is compiled in Code::Blocks with gcc compiler.

```
#include<stdio.h>

#include<stdlib.h>

#include<windows.h>

int i,j;

int main_exit;

void menu();

struct date{

   int month,day,year;


   };

struct {


   char name[60];

   int acc_no,age;

   char address[60];

   char citizenship[15];

   double phone;
```

Notes

```
    char acc_type[10];
    float amt;
    struct date dob;
    struct date deposit;
    struct date withdraw;

    }add,upd,check,rem,transaction;

float interest(float t,float amount,int rate)
{
    float SI;
    SI=(rate*t*amount)/100.0;
    return (SI);

}
void fordelay(int j)
{   int i,k;
    for(i=0;i<j;i++)
        k=i;
}

void new_acc()
{
    int choice;
    FILE *ptr;

    ptr=fopen("record.dat","a+");
    account_no:
    system("cls");
    printf("\t\t\t\xB2\xB2\xB2\ ADD RECORD  \xB2\xB2\xB2\xB2");
    printf("\n\n\nEnter today's date(mm/dd/yyyy):");
```

Notes

```
    scanf("%d/%d/%d",&add.deposit.month,&add.deposit.day,&add.deposit.year);
    printf("\nEnter the account number:");
    scanf("%d",&check.acc_no);
     while(fscanf(ptr,"%d %s %d/%d/%d %d %s %s %lf %s %f %d/%d/%d\n",&add.acc_
no,add.name,&add.dob.month,&add.dob.day,&add.dob.year,&add.age,add.address,add.
citizenship,&add.phone,add.acc_type,&add.amt,&add.deposit.month,&add.deposit.
day,&add.deposit.year)!=EOF)
    {
       if (check.acc_no==add.acc_no)
          {printf("Account no. already in use!");
          fordelay(1000000000);
             goto account_no;

          }
    }
    add.acc_no=check.acc_no;
       printf("\nEnter the name:");
    scanf("%s",add.name);
    printf("\nEnter the date of birth(mm/dd/yyyy):");
    scanf("%d/%d/%d",&add.dob.month,&add.dob.day,&add.dob.year);
    printf("\nEnter the age:");
    scanf("%d",&add.age);
    printf("\nEnter the address:");
    scanf("%s",add.address);
    printf("\nEnter the citizenship number:");
    scanf("%s",add.citizenship);
    printf("\nEnter the phone number: ");
    scanf("%lf",&add.phone);
    printf("\nEnter the amount to deposit:$");
    scanf("%f",&add.amt);
     printf("\nType of account:\n\t#Saving\n\t#Current\n\t#Fixed1(for 1 year)\n\t#Fixed2(for
2 years)\n\t#Fixed3(for 3 years)\n\n\tEnter your choice:");
    scanf("%s",add.acc_type);
```

Notes

```
        fprintf(ptr,"%d %s %d/%d/%d %d %s %s %lf %s %f %d/%d/%d\n",add.acc_no,add.
name,add.dob.month,add.dob.day,add.dob.year,add.age,add.address,add.citizenship,add.
phone,add.acc_type,add.amt,add.deposit.month,add.deposit.day,add.deposit.year);


    fclose(ptr);
    printf("\nAccount created successfully!");
    add_invalid:
    printf("\n\n\n\t\tEnter 1 to go to the main menu and 0 to exit:");
    scanf("%d",&main_exit);
    system("cls");
    if (main_exit==1)
        menu();
    else if(main_exit==0)
            close();
    else
        {
            printf("\nInvalid!\a");
            goto add_invalid;
        }
}
void view_list()
{
    FILE *view;
    view=fopen("record.dat","r");
    int test=0;
    system("cls");
    printf("\nACC. NO.\tNAME\t\t\tADDRESS\t\t\tPHONE\n");


     while(fscanf(view,"%d %s %d/%d/%d %d %s %s %lf %s %f %d/%d/%d",&add.acc_
no,add.name,&add.dob.month,&add.dob.day,&add.dob.year,&add.age,add.address,add.
citizenship,&add.phone,add.acc_type,&add.amt,&add.deposit.month,&add.deposit.
day,&add.deposit.year)!=EOF)
        {
```

Notes

```
            printf("\n%6d\t %10s\t\t\t%10s\t\t%.0lf",add.acc_no,add.name,add.address,add.
phone);
          test++;
      }

   fclose(view);
   if (test==0)
      {   system("cls");
          printf("\nNO RECORDS!!\n");}

   view_list_invalid:
    printf("\n\nEnter 1 to go to the main menu and 0 to exit:");
      scanf("%d",&main_exit);
      system("cls");
      if (main_exit==1)
          menu();
      else if(main_exit==0)
          close();
      else
      {
          printf("\nInvalid!\a");
          goto view_list_invalid;
      }
}
void edit(void)
{
   int choice,test=0;
   FILE *old,*newrec;
   old=fopen("record.dat","r");
   newrec=fopen("new.dat","w");
```

Notes

```
printf("\nEnter the account no. of the customer whose info you want to change:");

scanf("%d",&upd.acc_no);

 while(fscanf(old,"%d %s %d/%d/%d %d %s %s %lf %s %f %d/%d/%d",&add.acc_
no,add.name,&add.dob.month,&add.dob.day,&add.dob.year,&add.age,add.address,add.
citizenship,&add.phone,add.acc_type,&add.amt,&add.deposit.month,&add.deposit.
day,&add.deposit.year)!=EOF)

{

   if (add.acc_no==upd.acc_no)

   {

        test=1;

    printf("\nWhich information do you want to change?\n1.Address\n2.Phone\n\nEnter
your choice(1 for address and 2 for phone):");

      scanf("%d",&choice);

      system("cls");

      if(choice==1)

        {

printf("Enter the new address:");

 scanf("%s",upd.address);

fprintf(newrec,"%d %s %d/%d/%d %d %s %s %lf %s %f %d/%d/%d\n",add.acc_no,add.
name,add.dob.month,add.dob.day,add.dob.year,add.age,upd.address,add.citizenship,add.
phone,add.acc_type,add.amt,add.deposit.month,add.deposit.day,add.deposit.year);

        system("cls");

        printf("Changes saved!");

        }

      else if(choice==2)

        {

        printf("Enter the new phone number:");

        scanf("%lf",&upd.phone);

          fprintf(newrec,"%d %s %d/%d/%d %d %s %s %lf %s %f %d/%d/%d\n",add.
acc_no,add.name,add.dob.month,add.dob.day,add.dob.year,add.age,add.address,add.
citizenship,upd.phone,add.acc_type,add.amt,add.deposit.month,add.deposit.day,add.
deposit.year);

        system("cls");

        printf("Changes saved!");

        }
```

```
    }
    else
        fprintf(newrec,"%d %s %d/%d/%d %d %s %s %lf %s %f %d/%d/%d\n",add.
acc_no,add.name,add.dob.month,add.dob.day,add.dob.year,add.age,add.address,add.
citizenship,add.phone,add.acc_type,add.amt,add.deposit.month,add.deposit.day,add.
deposit.year);
  }
  fclose(old);
  fclose(newrec);
  remove("record.dat");
  rename("new.dat","record.dat");

if(test!=1)
    {  system("cls");
       printf("\nRecord not found!!\a\a\a");
       edit_invalid:
        printf("\nEnter 0 to try again,1 to return to main menu and 2 to exit:");
        scanf("%d",&main_exit);
        system("cls");
          if (main_exit==1)

            menu();
        else if (main_exit==2)
            close();
        else if(main_exit==0)
            edit();
        else
            {printf("\nInvalid!\a");
            goto edit_invalid;}
    }
  else
    {printf("\n\n\nEnter 1 to go to the main menu and 0 to exit:");
```

Notes

```
        scanf("%d",&main_exit);
        system("cls");
        if (main_exit==1)
            menu();
        else
            close();
        }
}


void transact(void)
{   int choice,test=0;
    FILE *old,*newrec;
    old=fopen("record.dat","r");
    newrec=fopen("new.dat","w");


        printf("Enter the account no. of the customer:");
    scanf("%d",&transaction.acc_no);
     while (fscanf(old,"%d %s %d/%d/%d %d %s %s %lf %s %f %d/%d/%d",&add.acc_
no,add.name,&add.dob.month,&add.dob.day,&add.dob.year,&add.age,add.address,add.
citizenship,&add.phone,add.acc_type,&add.amt,&add.deposit.month,&add.deposit.
day,&add.deposit.year)!=EOF)
  {


        if(add.acc_no==transaction.acc_no)
        {   test=1;
         if(strcmpi(add.acc_type,"fixed1")==0||strcmpi(add.acc_type,"fixed2")==0||strcmpi(add.
acc_type,"fixed3")==0)
              {
                printf("\a\a\a\n\nYOU CANNOT DEPOSIT OR WITHDRAW CASH IN FIXED
ACCOUNTS!!!!!");
                fordelay(1000000000);
                system("cls");
                menu();
```

Notes

```
        }
        printf("\n\nDo you want to\n1.Deposit\n2.Withdraw?\n\nEnter your choice(1 for
deposit and 2 for withdraw):");
        scanf("%d",&choice);
        if (choice==1)
        {
            printf("Enter the amount you want to deposit:$ ");
            scanf("%f",&transaction.amt);
            add.amt+=transaction.amt;
            fprintf(newrec,"%d %s %d/%d/%d %d %s %s %lf %s %f %d/%d/%d\n",add.
acc_no,add.name,add.dob.month,add.dob.day,add.dob.year,add.age,add.address,add.
citizenship,add.phone,add.acc_type,add.amt,add.deposit.month,add.deposit.day,add.
deposit.year);
            printf("\n\nDeposited successfully!");
        }
        else
        {
            printf("Enter the amount you want to withdraw:$ ");
            scanf("%f",&transaction.amt);
            add.amt-=transaction.amt;
            fprintf(newrec,"%d %s %d/%d/%d %d %s %s %lf %s %f %d/%d/%d\n",add.
acc_no,add.name,add.dob.month,add.dob.day,add.dob.year,add.age,add.address,add.
citizenship,add.phone,add.acc_type,add.amt,add.deposit.month,add.deposit.day,add.
deposit.year);
            printf("\n\nWithdrawn successfully!");
        }
    }
    else
    {
        fprintf(newrec,"%d %s %d/%d/%d %d %s %s %lf %s %f %d/%d/%d\n",add.
acc_no,add.name,add.dob.month,add.dob.day,add.dob.year,add.age,add.address,add.
citizenship,add.phone,add.acc_type,add.amt,add.deposit.month,add.deposit.day,add.
deposit.year);
    }
}
  fclose(old);
```

Notes

```
fclose(newrec);
remove(“record.dat”);
rename(“new.dat”,”record.dat”);
if(test!=1)
{
    printf(“\n\nRecord not found!!”);
    transact_invalid:
   printf(“\n\n\nEnter 0 to try again,1 to return to main menu and 2 to exit:”);
   scanf(“%d”,&main_exit);
   system(“cls”);
   if (main_exit==0)
     transact();
 else if (main_exit==1)
     menu();
 else if (main_exit==2)
     close();
 else
 {
     printf(“\nInvalid!”);
     goto transact_invalid;
 }

}
else
{
    printf(“\nEnter 1 to go to the main menu and 0 to exit:”);
     scanf(“%d”,&main_exit);
     system(“cls”);
     if (main_exit==1)
         menu();
     else
         close();
```

Notes

```
  }


}
void erase(void)
{
   FILE *old,*newrec;
   int test=0;
   old=fopen("record.dat","r");
   newrec=fopen("new.dat","w");
   printf("Enter the account no. of the customer you want to delete:");
   scanf("%d",&rem.acc_no);
    while (fscanf(old,"%d %s %d/%d/%d %d %s %s %lf %s %f %d/%d/%d",&add.acc_
no,add.name,&add.dob.month,&add.dob.day,&add.dob.year,&add.age,add.address,add.
citizenship,&add.phone,add.acc_type,&add.amt,&add.deposit.month,&add.deposit.
day,&add.deposit.year)!=EOF)
  {
      if(add.acc_no!=rem.acc_no)
            fprintf(newrec,"%d %s %d/%d/%d %d %s %s %lf %s %f %d/%d/%d\n",add.
acc_no,add.name,add.dob.month,add.dob.day,add.dob.year,add.age,add.address,add.
citizenship,add.phone,add.acc_type,add.amt,add.deposit.month,add.deposit.day,add.
deposit.year);


      else
         {test++;
         printf("\nRecord deleted successfully!\n");
         }
  }
  fclose(old);
  fclose(newrec);
  remove("record.dat");
  rename("new.dat","record.dat");
  if(test==0)
     {
        printf("\nRecord not found!!\a\a\a");
```

Notes

```
        erase_invalid:
          printf("\nEnter 0 to try again,1 to return to main menu and 2 to exit:");
          scanf("%d",&main_exit);

           if (main_exit==1)
              menu();
           else if (main_exit==2)
              close();
           else if(main_exit==0)
              erase();
           else
              {printf("\nInvalid!\a");
              goto erase_invalid;}
      }
   else
      {printf("\nEnter 1 to go to the main menu and 0 to exit:");
      scanf("%d",&main_exit);
      system("cls");
      if (main_exit==1)
         menu();
      else
         close();
      }

}

void see(void)
{
   FILE *ptr;
   int test=0,rate;
   int choice;
   float time;
```

```
float intrst;
ptr=fopen("record.dat","r");
printf("Do you want to check by\n1.Account no\n2.Name\nEnter your choice:");
scanf("%d",&choice);
if (choice==1)
{   printf("Enter the account number:");
    scanf("%d",&check.acc_no);

    while (fscanf(ptr,"%d %s %d/%d/%d %d %s %s %lf %s %f %d/%d/%d",&add.acc_
no,add.name,&add.dob.month,&add.dob.day,&add.dob.year,&add.age,add.address,add.
citizenship,&add.phone,add.acc_type,&add.amt,&add.deposit.month,&add.deposit.
day,&add.deposit.year)!=EOF)
    {
        if(add.acc_no==check.acc_no)
        {   system("cls");
            test=1;

        printf("\nAccount NO.:%d\nName:%s \nDOB:%d/%d/%d \nAge:%d \nAddress:%s
\nCitizenship No:%s \nPhone number:%.0lf \nType Of Account:%s \nAmount deposited:$
%.2f \nDate Of Deposit:%d/%d/%d\n\n",add.acc_no,add.name,add.dob.month,add.dob.
day,add.dob.year,add.age,add.address,add.citizenship,add.phone,
            add.acc_type,add.amt,add.deposit.month,add.deposit.day,add.deposit.year);
            if(strcmpi(add.acc_type,"fixed1")==0)
            {
                time=1.0;
                rate=9;
                intrst=interest(time,add.amt,rate);
                printf("\n\nYou will get $%.2f as interest on %d/%d/%d",intrst,add.deposit.
month,add.deposit.day,add.deposit.year+1);
            }
            else if(strcmpi(add.acc_type,"fixed2")==0)
            {
                time=2.0;
                rate=11;
                intrst=interest(time,add.amt,rate);
```

Notes

```
                printf("\n\nYou will get $.%.2f as interest on %d/%d/%d",intrst,add.deposit.
month,add.deposit.day,add.deposit.year+2);


            }
          else if(strcmpi(add.acc_type,"fixed3")==0)
            {
              time=3.0;
              rate=13;
              intrst=interest(time,add.amt,rate);
              printf("\n\nYou will get $.%.2f as interest on %d/%d/%d",intrst,add.deposit.
month,add.deposit.day,add.deposit.year+3);


            }
          else if(strcmpi(add.acc_type,"saving")==0)
            {
              time=(1.0/12.0);
              rate=8;
              intrst=interest(time,add.amt,rate);
              printf("\n\nYou will get $.%.2f as interest on %d of every month",intrst,add.
deposit.day);


            }
          else if(strcmpi(add.acc_type,"current")==0)
            {

              printf("\n\nYou will get no interest\a\a");


            }

      }
  }
```

Notes

```
}
else if (choice==2)
{   printf("Enter the name:");
    scanf("%s",&check.name);
     while (fscanf(ptr,"%d %s %d/%d/%d %d %s %s %lf %s %f %d/%d/%d",&add.acc_
no,add.name,&add.dob.month,&add.dob.day,&add.dob.year,&add.age,add.address,add.
citizenship,&add.phone,add.acc_type,&add.amt,&add.deposit.month,&add.deposit.
day,&add.deposit.year)!=EOF)
    {
       if(strcmpi(add.name,check.name)==0)
       {   system("cls");
          test=1;
       printf("\nAccount No.:%d\nName:%s \nDOB:%d/%d/%d \nAge:%d \nAddress:%s \
nCitizenship No:%s \nPhone number:%.0lf \nType Of Account:%s \nAmount deposited:$%.2f
\nDate Of Deposit:%d/%d/%d\n\n",add.acc_no,add.name,add.dob.month,add.dob.day,add.
dob.year,add.age,add.address,add.citizenship,add.phone,
          add.acc_type,add.amt,add.deposit.month,add.deposit.day,add.deposit.year);
          if(strcmpi(add.acc_type,"fixed1")==0)
            {
              time=1.0;
              rate=9;
              intrst=interest(time,add.amt,rate);
             printf("\n\nYou will get $.%.2f as interest on %d/%d/%d",intrst,add.deposit.
month,add.deposit.day,add.deposit.year+1);
            }
          else if(strcmpi(add.acc_type,"fixed2")==0)
            {
              time=2.0;
              rate=11;
              intrst=interest(time,add.amt,rate);
             printf("\n\nYou will get $.%.2f as interest on %d/%d/%d",intrst,add.deposit.
month,add.deposit.day,add.deposit.year+2);
            }
          else if(strcmpi(add.acc_type,"fixed3")==0)
```

Notes

```
                {
                    time=3.0;
                    rate=13;
                    intrst=interest(time,add.amt,rate);
                  printf("\n\nYou will get $.%.2f as interest on %d/%d/%d",intrst,add.deposit.
month,add.deposit.day,add.deposit.year+3);


                }
              else if(strcmpi(add.acc_type,"saving")==0)
                {
                    time=(1.0/12.0);
                    rate=8;
                    intrst=interest(time,add.amt,rate);
                  printf("\n\nYou will get $.%.2f as interest on %d of every month",intrst,add.
deposit.day);


                 }
              else if(strcmpi(add.acc_type,"current")==0)
                {

                    printf("\n\nYou will get no interest\a\a");

                }

            }
        }


   fclose(ptr);
    if(test!=1)
      {  system("cls");
         printf("\nRecord not found!!\a\a\a");
```

Notes

```
        see_invalid:
          printf("\nEnter 0 to try again,1 to return to main menu and 2 to exit:");
          scanf("%d",&main_exit);
          system("cls");
            if (main_exit==1)
               menu();
          else if (main_exit==2)
               close();
          else if(main_exit==0)
               see();
          else
               {
                  system("cls");
                  printf("\nInvalid!\a");
                  goto see_invalid;}
    }
else
    {printf("\nEnter 1 to go to the main menu and 0 to exit:");
    scanf("%d",&main_exit);}
    if (main_exit==1)
    {
       system("cls");
       menu();
    }
    else
      {


       system("cls");
       close();
       }
```

Notes

```
}



void close(void)

{

   printf("\n\n\n\nThis C Mini Project is developed by Code With C team!");

   }



void menu(void)

{   int choice;

   system("cls");

   system("color 9");

   printf("\n\n\t\t\tCUSTOMER ACCOUNT BANKING MANAGEMENT SYSTEM");

    printf("\n\n\n\t\t\t\xB2\xB2\xB2\xB2\xB2\xB2\xB2 WELCOME TO THE MAIN MENU \
xB2\xB2\xB2\xB2\xB2\xB2\xB2");

    printf("\n\n\t\t1.Create new account\n\t\t2.Update information of existing account\n\t\
t3.For transactions\n\t\t4.Check the details of existing account\n\t\t5.Removing existing
account\n\t\t6.View customer's list\n\t\t7.Exit\n\n\n\n\n\t\t Enter your choice:");

   scanf("%d",&choice);



   system("cls");

   switch(choice)

   {

      case 1:new_acc();

      break;

      case 2:edit();

      break;

      case 3:transact();

      break;

      case 4:see();

      break;

      case 5:erase();
```

Notes

```
        break;
        case 6:view_list();
        break;
        case 7:close();
        break;

    }




}
int main()
{
    char pass[10],password[10]="codewithc";
    int i=0;
    printf("\n\n\t\tEnter the password to login:");
    scanf("%s",pass);
    /*do
    {
    //if (pass[i]!=13&&pass[i]!=8)
        {
            printf("*");
            pass[i]=getch();
            i++;
        }
    }while (pass[i]!=13);
    pass[10]='\0';*/
    if (strcmp(pass,password)==0)
        {printf("\n\nPassword Match!\nLOADING");
        for(i=0;i<=6;i++)
        {
            fordelay(100000000);
```

Notes

```
        printf(“.”);
    }
        system(“cls”);
    menu();
    }
else
    {   printf(“\n\nWrong password!!\a\a\a”);
    login_try:
    printf(“\nEnter 1 to try again and 0 to exit:”);
    scanf(“%d”,&main_exit);
    if (main_exit==1)
        {

        system(“cls”);
        main();
        }

    else if (main_exit==0)
        {
        system(“cls”);
        close();}
    else
        {printf(“\nInvalid!”);
        fordelay(1000000000);
        system(“cls”);
        goto login_try;}

    }
    return 0;
}
```

Functions used in Bank Management System:

The source code for Customer Account Bank Management System is relatively short and easy to understand. I have divided this C mini project into many functions, most of which are related to different banking activities. Listed below are some of the more important functions which may help you understand the project better.

menu() – This function displays the menu or welcome screen to perform different banking activities mentioned below.

new_acc() – This function creates a new customer account. It asks for some personal and banking details of the customer such as name, date of birth, citizenship number, address and phone number. You can enter the amount to deposit and choose one type of deposit account – saving, current, fixed for 1 year, fixed for 2 years or fixed for 3 years.

view list() – With this function, you can view the customer's banking information such as account number, name, address and phone number provided while creating the account.

edit() – This function has been used for changing the address and phone number of a particular customer account.

transact() – With this function, you can deposit and withdraw money to and from a particular customer account.

erase() – This function is for deleting a customer account.

see() – This function shows account number, name, date of birth, citizenship number, age, address, phone number, type of account, amount deposited and date of deposit. It also displays the amount of interest corresponding to a particular account type.

Stack Operations Project

A stack is an abstract data type that serves as a collection of elements, with two principal operations: push, which adds an element to the collection, and pop, which removes the most recently added element that was not yet removed.

We start our code by including the header files "stdio.h" and "conio.h". The code also has a array to hold the values and the variable top for manipulation

```
struct stack /* Structure definition for stack */

{

int stk[MAXSIZE];

int top;

}
```

There are four options given to the user for executing the stack. They are Push, Pop, Display and Exit.

Notes

```
while (option)
{
printf (“----------------------------------------\n”);
printf (“ 1 --> PUSH \n”);
printf (“ 2 --> POP \n”);
printf (“ 3 --> DISPLAY \n”);
printf (“ 4 --> EXIT \n”);
printf (“----------------------------------------\n”);

printf (“Enter your choice\n”);
scanf (“%d”, &choice);
1
2
3
4
5
6
7
8
9
10
11

while (option)
{
printf (“----------------------------------------\n”);
printf (“ 1 --> PUSH \n”);
printf (“ 2 --> POP \n”);
printf (“ 3 --> DISPLAY \n”);
```

```
printf (“ 4 --> EXIT \n”);
printf (“----------------------------------------\n”);

printf (“Enter your choice\n”);
scanf (“%d”, &choice)
```

Option1 will initiate the push function

Option2 will initiate the pop function

Option 3 will initiate the Display function

```
int num;
if (s.top == (MAXSIZE - 1))
{
printf (“Stack is Full\n”);
return;
}
else
{
printf (“Enter the element to be pushed\n”);
scanf (“%d”, &num);
s.top = s.top + 1;
s.stk[s.top] = num;
}
```

Option 4 will return the program from the loop and the main() will exit

2

3

4

5

6

Notes

```
7
8
9
10


switch (choice)
{
case 1: push();
break;
case 2: pop();
break;
case 3: display();
break;
case 4: return;
}
```

Push function -> Its check is the stack is full or not. If not full it adds a number to the array. Else it will display the error message of “Stack is full”.

Pop function -> Its check is the stack is empty or not. If not empty it pops the last number from the array. Else it will display the error message of “Stack is Empty”.

Top variable is decreased by 1 each time a value is added to the stack

```
int num;
if (s.top == - 1)
{
printf (“Stack is Empty\n”);
return (s.top);
}
else
{
num = s.stk[s.top];
printf (“poped element is = %d\n”, s.stk[s.top]);
s.top = s.top - 1;
```

```
}
return(num);
```

display function -> Its check is the stack is empty or not. If not empty the using a for loop it printa all the values of the stack. Else it will display the error message of “Stack is Empty”.

```
int i;
if (s.top == -1)
{
printf (“Stack is empty\n”);
return;
}
else
{
printf (“\nThe status of the stack is\n”);
for (i = s.top; i >= 0; i--)
{
printf (“%d\n”, s.stk[i]);
}
}
printf (“\n”);
```

Mini Project in C Employee Record System

In this project, you can manage employee records – add, list, modify and delete records. Understanding this project will help you learn how to add, view, change and remove data using file handling.

The main features of this project include basic file handling operations; you will learn how to add, list, modify and delete data to/from file. The source code is relatively short, so thoroughly go through the mini project, and try to analyze how things such as functions, pointers, files, and arrays are implemented.

Currently, listed below are the only features that make up this project, but you can add new features as you like to make this project a better one!

- Add record
- List record
- Modify record
- Delete record

```
#include <stdio.h> ///for input output functions like printf, scanf
```

Notes

```
#include <stdlib.h>
#include <conio.h>
#include <windows.h> ///for windows related functions (not important)
#include <string.h>  ///string operations

/** List of Global Variable */
COORD coord = {0,0}; /// top-left corner of window

/**
   function : gotoxy
   @param input: x and y coordinates
   @param output: moves the cursor in specified position of console
*/
void gotoxy(int x,int y)
{
   coord.X = x;
   coord.Y = y;
   SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE),coord);
}

/** Main function started */

int main()
{
   FILE *fp, *ft; /// file pointers
   char another, choice;

   /** structure that represent a employee */
   struct emp
   {
      char name[40]; ///name of employee
      int age; /// age of employee
```

```
    float bs; /// basic salary of employee
};

struct emp e; /// structure variable creation

char empname[40]; /// string to store name of the employee

long int recsize; /// size of each record of employee

/** open the file in binary read and write mode
* if the file EMP.DAT already exists then it open that file in read write mode
* if the file doesn't exit it simply create a new copy
*/
fp = fopen("EMP.DAT","rb+");
if(fp == NULL)
{
    fp = fopen("EMP.DAT","wb+");
    if(fp == NULL)
    {
        printf("Connot open file");
        exit(1);
    }
}

/// sizeo of each record i.e. size of structure variable e
recsize = sizeof(e);

/// infinite loop continues untile the break statement encounter
while(1)
{
```

Notes

```
system("cls"); ///clear the console window
gotoxy(30,10); /// move the cursor to postion 30, 10 from top-left corner
printf("1. Add Record"); /// option for add record
gotoxy(30,12);
printf("2. List Records"); /// option for showing existing record
gotoxy(30,14);
printf("3. Modify Records"); /// option for editing record
gotoxy(30,16);
printf("4. Delete Records"); /// option for deleting record
gotoxy(30,18);
printf("5. Exit"); /// exit from the program
gotoxy(30,20);
printf("Your Choice: "); /// enter the choice 1, 2, 3, 4, 5
fflush(stdin); /// flush the input buffer
choice  = getche(); /// get the input from keyboard
switch(choice)
{
case '1':  /// if user press 1
    system("cls");
    fseek(fp,0,SEEK_END); /// search the file and move cursor to end of the file
    /// here 0 indicates moving 0 distance from the end of the file

    another = 'y';
    while(another == 'y')  /// if user want to add another record
    {
        printf("\nEnter name: ");
        scanf("%s",e.name);
        printf("\nEnter age: ");
        scanf("%d", &e.age);
        printf("\nEnter basic salary: ");
        scanf("%f", &e.bs);
```

```
            fwrite(&e,recsize,1,fp); /// write the record in the file


            printf("\nAdd another record(y/n) ");
            fflush(stdin);
            another = getche();
          }
          break;
        case '2':
          system("cls");
          rewind(fp); ///this moves file cursor to start of the file
          while(fread(&e,recsize,1,fp)==1)  /// read the file and fetch the record one record
per fetch
          {
          printf("\n%s %d %.2f",e.name,e.age,e.bs); /// print the name, age and basic salary
          }
          getch();
          break;


        case '3':  /// if user press 3 then do editing existing record
          system("cls");
          another = 'y';
          while(another == 'y')
          {
            printf("Enter the employee name to modify: ");
            scanf("%s", empname);
            rewind(fp);
            while(fread(&e,recsize,1,fp)==1)  /// fetch all record from file
            {
             if(strcmp(e.name,empname) == 0)  ///if entered name matches with that in file
              {
                printf("\nEnter new name,age and bs: ");
                scanf("%s%d%f",e.name,&e.age,&e.bs);
```

Notes

```
                fseek(fp,-recsize,SEEK_CUR); /// move the cursor 1 step back from current position
                fwrite(&e,recsize,1,fp); /// override the record
                break;
              }
            }
            printf("\nModify another record(y/n)");
            fflush(stdin);
            another = getche();
          }
          break;
        case '4':
          system("cls");
          another = 'y';
          while(another == 'y')
          {
            printf("\nEnter name of employee to delete: ");
            scanf("%s",empname);
            ft = fopen("Temp.dat","wb");  /// create a intermediate file for temporary storage
            rewind(fp); /// move record to starting of file
            while(fread(&e,recsize,1,fp) == 1)  /// read all records from file
            {
              if(strcmp(e.name,empname) != 0)  /// if the entered record match
              {
                fwrite(&e,recsize,1,ft); /// move all records except the one that is to be deleted to temp file
              }
            }
            fclose(fp);
            fclose(ft);
            remove("EMP.DAT"); /// remove the orginal file
            rename("Temp.dat","EMP.DAT"); /// rename the temp file to original file name
```

```
            fp = fopen("EMP.DAT", "rb+");
            printf("Delete another record(y/n)");
            fflush(stdin);
            another = getche();
          }
          break;
       case '5':
          fclose(fp);  /// close the file
          exit(0); /// exit from the program
       }
    }
    return 0;
}
```

Customer Billing System C Project

Customer Billing System Project is a simple console application designed to demonstrate the practical use of C programming language and its features as wells as to generate an application which can be used in any departmental store, shops, cafes etc. for billing to the customer.

User Defined Functions Used:

Although graphics has not been used in this project, the application of user defined functions and structures have been effectively used here. The major user defined functions used in this C project are:

- void input()
- void writefile()
- void search()
- void output()

The function void input() is used to add the new customer account i.e. with the help of this functions the parameters such as name, address, phone number, amount paid etc. are asked and entered. Another function void writefile() has been utilized to create a file on hard disc of computer for storing the information and data of a customer.

The function void search() has been used to look for previously stored accounts either by name or by number of the customer. The fourth and the last user defined function used in this Customer Billing System Project in C is void output() which has been defined to show the result as console output.

In Customer Billing System, structure has very beautifully used to group the data type in single unit. The date variables (day, month and year) have been grouped in the

Notes

structures named date where as other variables such as name, number, street, paid amount etc. are grouped under another structure named account.

Customer Billing System application is so simple to use. In order to use the application, click at the exe file and then, you will have three options to:

1. To add account
2. To search account
3. To exit

As per your need, enter 1, 2,or 3 and follow the instructions provided by the application itself.

Features:

1. It can hold any number of accounts and account can be added to the program at any time.
2. The programming of simple calculations such as calculation of due amount, balance etc. have been embed in the code of project.
3. The Customer Billing System project in C gives you the facility of searching the account by two ways, either by name of the customer or by the number of customer.
4. The due amount to be paid is shown as negative balance.
5. If you have nothing to do with the program, you can directly exit from the main menu.

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

 void input();
 void writefile();
 void search();
 void output();

 struct date{
            int month;
            int day;
            int year;
            };

 struct account {
          int number;
```

Notes

```
        char name[100];
        int acct_no;
        float mobile_no;
        char street[100];
        char city[100];
        char acct_type;
        float oldbalance;
        float newbalance;
        float payment;
        struct date lastpayment;
}customer;
int tl,sl,ts;
void main()
        {
          int i,n;
          char ch;
          clrscr();

          _setcursortype(_NOCURSOR);
          printf("   CUSTOMER BILLING SYSTEM:\n\n");
          printf("================================\n");
          printf("\n1:    to add account on list\n");
          printf("2:    to search customer account\n");
          printf("3:    exit\n");
          printf("\n================================\n");
          do{
              printf("\nselect what do you want to do?");
              ch=getche();
          }while(ch<='0' || ch>'3');
          switch(ch){
                                  case '1':
                                                  clrscr();
```

Notes

```
                                        printf("\nhow many customer ac-
counts?");
                                        scanf("%d",&n);
                                        for(i=0;i<n;i++){
                                                input();
                                                if(customer.payment>0)
                                                        customer.
acct_type=(customer.payment<0.1*customer.oldbalance)? 'C': 'D';
                                                else
                                                        customer.
acct_type=(customer.oldbalance>0)?'D' : 'C';
                                                customer.
newbalance=customer.oldbalance - customer.payment;
                                                writefile();
                                        }
                                        main();
                                case '2':
                                        clrscr();
                                        printf("search by what?\n");
                                        printf("\n1 --- search by customer
number\n");
                                        printf("2 --- search by customer
name\n");
                                        search();
                                        ch=getche();
                                        main();
                                case '3':
                                        clrscr();
                                        delay(700);
                                        textcolor(RED);
                                        gotoxy(25,25);
                                        cprintf("\nA PROJECT BY BIDUR &
SUJAN");
                                        delay(1500);
                                        exit(1);
```

Notes

```
        }
}


 void input()
        {
         FILE *fp=fopen("bidur.dat","rb");
         fseek (fp,0,SEEK_END);
         tl=ftell(fp);
         sl=sizeof(customer);
         ts=tl/sl;
         fseek(fp,(ts-1)*sl,SEEK_SET);
         fread(&customer,sizeof(customer),1,fp);
         printf("\ncustomer no:%d\n",++customer.number);
         fclose(fp);
         printf("       Account number:");
         scanf("%d",&customer.acct_no);
         printf("\n      Name:");
         scanf("%s",customer.name);
         printf("\n      mobile no:");
         scanf("%f",&customer.mobile_no);
         printf("       Street:");
         scanf("%s",customer.street);
         printf("       City:");
         scanf("%s",customer.city);
         printf("       Previous balance:");
         scanf("%f",&customer.oldbalance);
         printf("       Current payment:");
         scanf("%f",&customer.payment);
         printf("       Payment date(mm/dd/yyyy):");
```

Notes

```
            scanf("%d/%d/%d",&customer.lastpayment.month,&customer.lastpayment.
day,&customer.lastpayment.year);
            return;
}

void writefile()
{
            FILE *fp;
            fp=fopen("bidur.dat","ab");
            fwrite(&customer,sizeof(customer),1,fp);
            fclose(fp);
            return;
}

void search()
{
          char ch;
          char nam[100];
          int n,i,m=1;
          FILE *fp;
          fp=fopen("bidur.dat","rb");
          do{
                                printf("\nenter your choice:");
                                ch=getche();
          }while(ch!='1' && ch!='2');
          switch(ch){
              case '1':
                                fseek(fp,0,SEEK_END);
                                tl=ftell(fp);
                                sl=sizeof(customer);
                                ts=tl/sl;
                                do{
```

```
                    printf("\nchoose customer number:");
                    scanf("%d",&n);
                    if(n<=0 || n>ts)
                    printf("\nenter correct\n");
                    else{
                       fseek(fp,(n-1)*sl,SEEK_SET);
                       fread(&customer,sl,1,fp);
                       output();
                    }
                    printf("\n\nagain?(y/n)");
                    ch=getche();
                }while(ch=='y');
                fclose(fp);
                break;
        case '2':
                fseek(fp,0,SEEK_END);
                tl=ftell(fp);
                sl=sizeof(customer);
                ts=tl/sl;
                fseek(fp,(ts-1)*sl,SEEK_SET);
                fread(&customer,sizeof(customer),1,fp);
                n=customer.number;

                do{
                    printf("\nenter the name:");
                    scanf("%s",nam);
                    fseek(fp,0,SEEK_SET);
                    for(i=1;i<=n;i++)
                    {
                     fread(&customer,sizeof(customer)
,1,fp);
                    if(strcmp(customer.name,nam)==0)
                     {
```

Notes

```
                                                    output();
                                                    m=0;
                                                    break;
                                                }
                                            }
                                            if(m!=0)
                                            printf("\n\ndoesn't exist\n");
                                            printf("\nanother?(y/n)");
                                            ch=getche();
                                        }while(ch=='y');
                                        fclose(fp);
                    }
                    return;
                }



void output()
            {
              printf("\n\n   Customer no    :%d\n",customer.number);
              printf("    Name                :%s\n",customer.name);
              printf("    Mobile no      :%.f\n",customer.mobile_no);
              printf("    Account number :%d\n",customer.acct_no);
              printf("    Street         :%s\n",customer.street);
              printf("    City           :%s\n",customer.city);
              printf("    Old balance    :%.2f\n",customer.oldbalance);
              printf("    Current payment:%.2f\n",customer.payment);
              printf("    New balance    :%.2f\n",customer.newbalance);
             printf("   Payment date  :%d/%d/%d\n\n",customer.lastpayment.month,customer.
lastpayment.day,customer.lastpayment.year);
              printf("    Account status :");
```

```
textcolor(128+RED);
switch(customer.acct_type)
  {
  case 'C':
                        cprintf("CURRENT\n\n");
                        break;
  case 'O':
                        cprintf("OVERDUE\n\n");
                        break;
  case 'D':
                        cprintf("DELINQUENT\n\n");
                        break;
  default:
                        cprintf("ERROR\\n\n");
  }
  textcolor(WHITE);
  return;
}
```

Quiz Game Mini Project in C

This is a complete and error-free Quiz Game Mini Project in C designed as a simple console application. In this project, a number of questions are asked, and the user is awarded cash prize for each correct answer given. n quiz game, questions are chosen in such a a way that they cover all fields of a typical quiz contest. The user's general knowledge is tested with quiz questions regarding science, technology, movies, sports, general health, geography and many more.

- edit_score() – adds the current cash prize won to the previous one upon giving the right answer to a question
- help() – help menu with game summary and rules
- reset_score() – to reset the highest score/cash prize to default
- show_record() – shows the highest cash prize won by a particular user
- show_score() – to view the highest score

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

Notes

```
#include<stdlib.h>
#include<string.h>
void show_record();
void reset_score();
void help();
void edit_score(float , char []);
int main()
    {
    int countr,r,r1,count,i,n;
    float score;
    char choice;
    char playername[20];
    mainhome:
    system("cls");
    printf("\t\t\tC PROGRAM QUIZ GAME\n");
    printf("\n\t\t_____");

    printf("\n\t\t\t   WELCOME ");
    printf("\n\t\t\t      to ");
    printf("\n\t\t\t   THE GAME ");
    printf("\n\t\t_____");
    printf("\n\t\t_____");
    printf("\n\t\t   BECOME A MILLIONAIRE!!!!!!!!!!    ") ;
    printf("\n\t\t_____");
    printf("\n\t\t_____");
    printf("\n\t\t > Press S to start the game");
    printf("\n\t\t > Press V to view the highest score  ");
    printf("\n\t\t > Press R to reset score");
    printf("\n\t\t > press H for help            ");
    printf("\n\t\t > press Q to quit             ");
    printf("\n\t\t_____\n\n");
```

```
choice=toupper(getch());
if (choice=='V')
        {
        show_record();
        goto mainhome;
        }
else if (choice=='H')
        {
        help();getch();
        goto mainhome;
        }
        else if (choice=='R')
        {reset_score();
        getch();
        goto mainhome;}
        else if (choice=='Q')
        exit(1);
else if(choice=='S')
{
 system("cls");

printf("\n\n\n\n\n\n\n\n\n\n\n\t\t\tResister your name:");
 gets(playername);
system("cls");
printf("\n ---------- Welcome %s to C Program Quiz Game ----------",playername);
printf("\n\n Here are some tips you might wanna know before playing:");
printf("\n ----------");
printf("\n >> There are 2 rounds in this Quiz Game,WARMUP ROUND & CHALLANGE ROUND");
printf("\n >> In warmup round you will be asked a total of 3 questions to test your");
printf("\n    general knowledge. You are eligible to play the game if you give atleast 2");
printf("\n    right answers, otherwise you can't proceed further to the Challenge Round.");
```

Notes

```
printf("\n >> Your game starts with CHALLANGE ROUND. In this round you will be asked a");
printf("\n    total of 10 questions. Each right answer will be awarded $100,000!");
printf("\n    By this way you can win upto ONE MILLION cash prize!!!!!.....");
printf("\n >> You will be given 4 options and you have to press A, B ,C or D for the");
printf("\n    right option.");
printf("\n >> You will be asked questions continuously, till right answers are given");
printf("\n >> No negative marking for wrong answers!");
printf("\n\n\t!!!!!!!!!!!!! ALL THE BEST !!!!!!!!!!!!!");
printf("\n\n\n Press Y  to start the game!\n");
printf("\n Press any other key to return to the main menu!");
if (toupper(getch())=='Y')
                                        {
                                            goto home;
   }
        else
                                        {
    goto mainhome;
   system("cls");
   }

 home:
 system("cls");
 count=0;
 for(i=1;i<=3;i++)
 {
system("cls");
 r1=i;



 switch(r1)
                                        {
```

```
case 1:
printf("\n\nWhich of the following is a Palindrome number?");
printf("\n\nA.42042\t\tB.101010\n\nC.23232\t\tD.01234");
if (toupper(getch())=='C')
                {
                   printf("\n\nCorrect!!!");count++;
                   getch();
                   break;
}
else
     {
          printf("\n\nWrong!!! The correct answer is C.23232");
        getch();
     break;
     }

     case 2:
printf("\n\n\nThe country with the highest environmental performance index is...");
printf("\n\nA.France\t\tB.Denmark\n\nC.Switzerland\t\tD.Finland");
if (toupper(getch())=='C')
                {printf("\n\nCorrect!!!");count++;
                getch();
                break;}
else
          {printf("\n\nWrong!!! The correct answer is C.Switzerland");
     getch();
     break;}
```

Notes

```
        case 3:
                                printf("\n\n\nWhich animal laughs like human being?");
                                printf("\n\nA.Polar Bear\t\tB.Hyena\n\nC.Donkey\t\
tD.Chimpanzee");
                                if (toupper(getch())=='B')
                                        {printf("\n\nCorrect!!!");count++;
                                        getch();
                                        break;}
                                else
                                    {printf("\n\nWrong!!! The correct answer is
B.Hyena");
                                    getch();
                                    break;}

        case 4:
                                printf("\n\n\nWho was awarded the youngest player
award in Fifa World Cup 2006?");
                                printf("\n\nA.Wayne Rooney\t\tB.Lucas Podolski\n\
nC.Lionel Messi\t\tD.Christiano Ronaldo");
                                if (toupper(getch())=='B')
                                        {printf("\n\nCorrect!!!");count++;
                                        getch();
                                         break;}
                                else
                                   {printf("\n\nWrong!!! The correct answer is B.Lucas
Podolski");
                                    getch();
                                    break;}

        case 5:
        printf("\n\n\nWhich is the third highest mountain in the world?");
        printf("\n\nA.Mt. K2\t\tB.Mt. Kanchanjungha\n\nC.Mt. Makalu\t\tD.Mt. Kilimanjaro");
        if (toupper(getch())=='B')
```

```
        {printf("\n\nCorrect!!!");count++;
        getch();
         break;}
    else
                        {printf("\n\nWrong!!! The correct answer is B.Mt.
Kanchanjungha");
                        getch();
                        break;}

    case 6:
                    printf("\n\n\nWhat is the group of frogs known as?");
                    printf("\n\nA.A traffic\t\tB.A toddler\n\nC.A police\t\tD.An
Army");
                    if (toupper(getch())=='D' )
                            {printf("\n\nCorrect!!!");count++;
                            getch();
                            break;}
                    else
                        {printf("\n\nWrong!!! The correct answer is D.An
Army");
                        getch();
                        break;}}
                        }

        if(count>=2)
        {goto test;}
        else
        {
        system("cls");
        printf("\n\nSORRY YOU ARE NOT ELIGIBLE TO PLAY THIS GAME, BETTER
LUCK NEXT TIME");
        getch();
        goto mainhome;
        }
```

Notes

```
    test:
    system("cls");
    printf("\n\n\t*** CONGRATULATION %s you are eligible to play the Game ***",player-
name);
    printf("\n\n\n\n\t!Press any key to Start the Game!");
    if(toupper(getch())=='p')
                                        {goto game;}
game:
    countr=0;
    for(i=1;i<=10;i++)
    {system("cls");
    r=i;

    switch(r)
                                        {
                                        case 1:
                                        printf("\n\nWhat is the National Game of England?");
                                        printf("\n\nA.Football\t\tB.Basketball\n\nC.Cricket\t\
tD.Baseball");
                                        if (toupper(getch())=='C')
                                        {printf("\n\nCorrect!!!");countr++;getch();
                                                break;getch();}
                                        else
                                            {printf("\n\nWrong!!! The correct answer is
C.Cricket");getch();
                                        goto score;
                                        break;}

                                        case 2:
                                        printf("\n\n\nStudy of Earthquake is called.....,");
                                        printf("\n\nA.Seismology\t\tB.Cosmology\n\
nC.Orology\t\tD.Etimology");
                                        if (toupper(getch())=='A')
```

Notes

```
                {printf("\n\nCorrect!!!");countr++;getch();
                        break;}
                else
                    {printf("\n\nWrong!!! The correct answer is
A.Seismology");getch();
                goto score;
                 break;
                 }

    case 3:
                printf("\n\n\nAmong the top 10 highest peaks in the
world, how many lie in Nepal? ");
                printf("\n\nA.6\t\tB.7\n\nC.8\t\tD.9");
                if (toupper(getch())=='C')
                {printf("\n\nCorrect!!!");countr++;getch();
                        break;}
                else
                    {printf("\n\nWrong!!! The correct answer is
C.8");getch();
                 goto score;
                 break;}

    case 4:
                printf("\n\n\nThe Laws of Electromagnetic Induction
were given by?");
                printf("\n\nA.Faraday\t\tB.Tesla\n\nC.Maxwell\t\
tD.Coulomb");
                if (toupper(getch())=='A')
                {printf("\n\nCorrect!!!");countr++;getch();
                        break;}
                else
                    {
        printf("\n\nWrong!!! The correct answer is A.Faraday");getch();
                 goto score;
```

Notes

```
                    break;
                    }

        case 5:
                    printf("\n\n\nIn what unit is electric power measured?");
                    printf("\n\nA.Coulomb\t\tB.Watt\n\nC.Power\t\
tD.Units");
                    if (toupper(getch())=='B')
                                { printf("\n\
nCorrect!!!");countr++;getch(); break;}
                    else
                    {
                        printf("\n\nWrong!!! The correct answer is
B.Power");
                    getch();
                    goto score;
                    break;
                    }

                    case 6:
                    printf("\n\n\nWhich element is found in Vitamin B12?");
                    printf("\n\nA.Zinc\t\tB.Cobalt\n\nC.Calcium\t\tD.Iron");
                    if (toupper(getch())=='B' )
                    {printf("\n\nCorrect!!!");countr++;getch();
                                break;}
                    else
                        {printf("\n\nWrong!!! The correct answer is
B.Cobalt");goto score;
                    getch();
                    break;}

        case 7:
```

```
printf("\n\n\nWhat is the National Name of Japan?");
printf("\n\nA.Polska\t\tB.Hellas\n\nC.Drukyul\t\
tD.Nippon");
if (toupper(getch())=='D')
{printf("\n\nCorrect!!!");countr++;getch();
                break;}
else
    {printf("\n\nWrong!!! The correct answer is
D.Nippon");getch();
  goto score;
  break;}

  case 8:
printf("\n\n\nHow many times a piece of paper can be
folded at the most?");
printf("\n\nA.6\t\tB.7\n\nC.8\t\tD.Depends on the size
of paper");
if (toupper(getch())=='B')
{printf("\n\nCorrect!!!");countr++;getch(); break;}
else
    {printf("\n\nWrong!!! The correct answer is
B.7");getch();
  goto score;
  break;}

  case 9:
printf("\n\n\nWhat is the capital of Denmark?");
printf("\n\nA.Copenhagen\t\tB.Helsinki\n\nC.Ajax\t\
tD.Galatasaray");
if (toupper(getch())=='A')
            {printf("\n\nCorrect!!!");countr++;
getch();
```

Notes

```
                                        break;}
                                else
                                    {printf("\n\nWrong!!! The correct answer is
A.Copenhagen");getch();
                                goto score;
                                break;}

    case 10:
                                printf("\n\n\nWhich is the longest River in the world?");
                                printf("\n\nA.Nile\t\tB.Koshi\n\nC.Ganga\t\tD.Amazon");
                                if (toupper(getch())=='A')
                                {printf("\n\nCorrect!!!");countr++;getch(); break;}
                                else
                                    {printf("\n\nWrong!!! The correct answer is
A.Nile");getch();break;goto score;}

    case 11:
                                printf("\n\n\nWhat is the color of the Black Box in
aeroplanes?");
                                printf("\n\nA.White\t\tB.Black\n\nC.Orange\t\tD.Red");
                                if (toupper(getch())=='C')
                                {printf("\n\nCorrect!!!");countr++;getch();
                                        break;}
                                else
        {printf("\n\nWrong!!! The correct answer is C.Orange");getch();
        break;goto score;}

    case 12:
                                printf("\n\n\nWhich city is known at 'The City of Seven
Hills'?");
                                printf("\n\nA.Rome\t\tB.Vactican City\n\nC.Madrid\t\
tD.Berlin");
                                if (toupper(getch())=='A')
```

```
                                        {printf("\n\nCorrect!!!");countr++;getch();
                                          break;}
                                else
        {printf("\n\nWrong!!! The correct answer is A.Rome");getch();
        break;goto score;}

                                case 13:
                                printf("\n\n\nName the country where there no mos-
quitoes are found?");
                                printf("\n\nA.Japan\t\tB.Italy\n\nC.Argentina\t\
tD.France");
                                if (toupper(getch())=='D')
                                {printf("\n\nCorrect!!!");countr++;getch();
                                        break;}
                                else
                                        {printf("\n\nWrong!!! The correct answer is
D.France");getch();
                                    break;goto score;}

      case 14:
                                printf("\n\n\nWho is the author of 'Pulpasa Cafe'?");
                                printf("\n\nA.Narayan Wagle\t\tB.Lal Gopal Subedi\n\
nC.B.P. Koirala\t\tD.Khagendra Sangraula");
                                if (toupper(getch())=='A')
                                {printf("\n\nCorrect!!!");countr++;getch();
                                        break;}
                                else
                                        {printf("\n\nWrong!!! The correct answer is
A.Narayan Wagle");getch();
                                    break;goto score;}

                                case 15:
                                printf("\n\n\nWhich Blood Group is known as the Uni-
```

Notes

```
versal Recipient?");
                                        printf("\n\nA.A\t\tB.AB\n\nC.B\t\tD.O");
                                        if (toupper(getch())=='B')
                                        {printf("\n\nCorrect!!!");countr++;getch();
                                                        break;}
                                        else
                                                {printf("\n\nWrong!!! The correct answer is
B.AB");getch();
                                              goto score;
                                              break;}

                                        case 16:
                                        printf("\n\n\nWhat is the unit of measurement of dis-
tance between Stars?");
                                        printf("\n\nA.Light Year\t\tB.Coulomb\n\nC.Nautical
Mile\t\tD.Kilometer");
                                        if (toupper(getch())=='A')
                                        {printf("\n\nCorrect!!!");countr++; getch();
                                                        break;
                                                        }
                                        else
                                            {printf("\n\nWrong!!! The correct answer is A.Light
Year");getch();
                                              goto score;
                                              break;}

                                        case 17:
                                        printf("\n\n\nThe country famous for Samba Dance
is.....");
                                        printf("\n\nA.Brazil\t\tB.Venezuela\n\nC.Nigeria\t\
tD.Bolivia");
                                        if (toupper(getch())=='A')
                                                        {printf("\n\nCorrect!!!");countr++;
getch();
```

```
                    break;}
            else
                {printf("\n\nWrong!!! The correct answer is
A.Brazil");getch();goto score;
                break;}

            case 18:
            printf("\n\n\nWind speed is measure by_____?");
            printf("\n\nA.Lysimeter\t\tB.Air vane\n\nC.Hydrometer\t\
tD.Anemometer\n\n");
            if (toupper(getch())=='D')
                    {printf("\n\nCorrect!!!");countr++;
getch();
                    break;}
            else
                {printf("\n\nWrong!!! The correct answer is
D.Anemometer");getch();goto score;
                break;}

            case 19:
            printf("\n\n\nWhich city in the world is popularly known
as The City of Temple?");
            printf("\n\nA.Delhi\tB.Bhaktapur\n\nC.Kathmandu\
tD.Agra\n\n");
            if (toupper(getch())=='C')
            {printf("\n\nCorrect!!!");countr++; getch();
                    break;}
            else
                {printf("\n\nWrong!!! The correct answer is
C.Kathmandu");getch();goto score;
                break;}

            case 20:
            printf("\n\n\nWhich hardware was used in the First
Generation Computer?");
```

Notes

```
                                        printf("\n\nA.Transistor\t\tB.Valves\n\nC.I.C\t\tD.S.S.I");
                                        if (toupper(getch())=='B')
                                                {printf("\n\nCorrect!!!");countr++;
getch();
                                                break;}
                                        else
                                            {printf("\n\nWrong!!! The correct answer is
B.Valves");getch();goto score;
                                          break;}

                                        case 21:
                                        printf("\n\n\nOzone plate is being destroyed regularly
because of____ ?");
                                        printf("\n\nA.L.P.G\t\tB.Nitrogen\n\nC.Methane\t\tD.
C.F.C");
                                        if (toupper(getch())=='D')
                                                {printf("\n\nCorrect!!!");countr++;
getch();
                                                break;}
                                        else
                                            {printf("\n\nWrong!!! The correct answer is D.
C.F.C");getch();goto score;
                                          break;}

                                        case 22:
                                        printf("\n\n\nWho won the Women's Australian Open
Tennis in 2007?");
                                        printf("\n\nA.Martina Hingis\t\tB.Maria Sarapova\n\
nC.Kim Clijster\t\tD.Serena Williams");
                                        if (toupper(getch())=='D')
                                                {printf("\n\nCorrect!!!");countr++;
getch();
                                                break;}
                                        else
                                            {printf("\n\nWrong!!! The correct answer is D.Serena
Williams");getch();goto score;
```

```
                                    break;}


                                case 23:
                                printf("\n\n\nWhich film was awarded the Best Motion
Picture at Oscar in 2010?");
                                printf("\n\nA.The Secret in their Eyes\t\tB.Shutter
Island\n\nC.The King's Speech\t\tD.The Reader");
                                if (toupper(getch())=='C')
                                        {printf("\n\nCorrect!!!");countr++;
getch();
                                        break;}
                                else
                                    {printf("\n\nWrong!!! The correct answer is C.The
King's Speech");getch();goto score;
                                    break;}}}
        score:
   system("cls");
        score=(float)countr*100000;
        if(score>0.00 && score<1000000)
        {
          printf("\n\n\t\t********** CONGRATULATION **********");
           printf("\n\t You won $%.2f",score);goto go;}


         else if(score==1000000.00)
        {
          printf("\n\n\n \t\t********** CONGRATULATION **********");
          printf("\n\t\t\t\t YOU ARE A MILLIONAIRE!!!!!!!!!");
          printf("\n\t\t You won $%.2f",score);
          printf("\t\t Thank You!!");
        }
         else
{
        printf("\n\n\t******** SORRY YOU DIDN'T WIN ANY CASH ********");
```

Notes

```
            printf("\n\t\t Thanks for your participation");
            printf("\n\t\t TRY AGAIN");goto go;}

        go:
        puts("\n\n Press Y if you want to play next game");
        puts(" Press any key if you want to go main menu");
        if (toupper(getch())=='Y')
                                goto home;
        else
                                {
                                edit_score(score,playername);
                                goto mainhome;}}}

void show_record()
   {system("cls");
        char name[20];
        float scr;
        FILE *f;
        f=fopen("score.txt","r");
        fscanf(f,"%s%f",&name,&scr);
        printf("\n\n\t\t**************************************************************");
        printf("\n\n\t\t %s has secured the Highest Score %0.2f",name,scr);
        printf("\n\n\t\t**************************************************************");
        fclose(f);
        getch();}

void reset_score()
   {system("cls");
   float sc;
        char nm[20];
        FILE *f;
        f=fopen("score.txt","r+");
```

```
        fscanf(f,"%s%f",&nm,&sc);
        sc=0;
        fprintf(f,"%s,%.2f",nm,sc);
   fclose(f);}

void help()
        {system("cls");
   printf("\n\n                    HELP");
   printf("\n --------------------------------------------------------------------------");
   printf("\n ..... C program Quiz Game.....");
    printf("\n >> There are two rounds in the game, WARMUP ROUND & CHALLANGE ROUND");
  printf("\n >> In warmup round you will be asked a total of 3 questions to test your general");
   printf("\n    knowledge. You will be eligible to play the game if you can give atleast 2");
   printf("\n    right answers otherwise you can't play the Game.....");
   printf("\n >> Your game starts with the CHALLANGE ROUND. In this round you will be asked");
   printf("\n    total 10 questions each right answer will be awarded $100,000.");
   printf("\n    By this way you can win upto ONE MILLION cash prize in USD.....");
   printf("\n >> You will be given 4 options and you have to press A, B ,C or D for the");
   printf("\n    right option");
    printf("\n >> You will be asked questions continuously if you keep giving the right answers.");
   printf("\n >> No negative marking for wrong answers");
        printf("\n\n\t**********BEST OF LUCK**********");
        printf("\n\n\t*****C PROGRAM QUIZ GAME is developed by CODE WITH C TEAM********");}

void edit_score(float score, char plnm[20])
        {system("cls");
        float sc;
        char nm[20];
        FILE *f;
        f=fopen("score.txt","r");
```

Notes

```
fscanf(f,"%s%f",&nm,&sc);
if (score>=sc)
  { sc=score;
    fclose(f);
    f=fopen("score.txt","w");
    fprintf(f,"%s\n%.2f",plnm,sc);
    fclose(f);}}
```