

This book was particularly designed for new students taking their first curriculum lesson, but it is also welcome for everyone interested in studying C to review. It is intended to complement lectures in the classroom with a focus on C programming. The subjects are arranged on the basis of the order of lecture. In addition to modifications in the language and presentation of programming, there were other sections. At the end of the chapter are listed some common questions and scheduling errors. Here you will only tackle a subset of the ANSI C language. In particular, the student is assumed to be working with ANSI C (which supports ANSI C) or the Borland Turbo C / C++ compiler in Windows under the Linux environment and programming. There are also standards for coding to be followed and a few pointers for programming using the Linux environment. May you learn a great deal from this manual study and may the knowledge you've gained be used for everyone's common good.



Dr. Yogesh Kumar Sharma, Presently working as a Associate Professor (HOD / Research Coordinator) Department of Computer Science Engineering and IT at "Shri Jagdish prasad Jhabarmal Tibrewala University", Chudela, Jhunjhunu (Rajasthan). He completed his B.Sc with Computer Application in the year of 2002 awarded from S.M.L. (P.G.) College, Jhunjhunu.

FOR AUTHOR USE

Sharma, Varghese, Nagaprasad

Yogesh Kumar Sharma  
Mili Mary Varghese  
Sriramula Nagaprasad

# C Programming Language for Beginners

Fundamentals of C Programming Language



**Yogesh Kumar Sharma  
Mili Mary Varghese  
Sriramula Nagaprasad**

**C Programming Language for Beginners**

FOR AUTHOR USE ONLY

FOR AUTHOR USE ONLY

**Yogesh Kumar Sharma  
Mili Mary Varghese  
Sriramula Nagaprasad**

# **C Programming Language for Beginners**

**Fundamentals of C Programming Language**

FOR AUTHOR USE ONLY

**LAP LAMBERT Academic Publishing**

**Imprint**

Any brand names and product names mentioned in this book are subject to trademark, brand or patent protection and are trademarks or registered trademarks of their respective holders. The use of brand names, product names, common names, trade names, product descriptions etc. even without a particular marking in this work is in no way to be construed to mean that such names may be regarded as unrestricted in respect of trademark and brand protection legislation and could thus be used by anyone.

Cover image: [www.ingimage.com](http://www.ingimage.com)

Publisher:

LAP LAMBERT Academic Publishing

is a trademark of

International Book Market Service Ltd., member of OmniScriptum Publishing Group

17 Meldrum Street, Beau Bassin 71504, Mauritius

Printed at: see last page

**ISBN: 978-620-2-81561-1**

Copyright © Yogesh Kumar Sharma, Mili Mary Varghese, Sriramula Nagaprasad

Copyright © 2020 International Book Market Service Ltd., member of OmniScriptum Publishing Group

FOR AUTHOR USE ONLY

# **C Programming Language for beginners**

FOR AUTHORIZED USE ONLY

## About Authors



### **Dr. Yogesh Kumar Sharma**

Dr. Yogesh Kumar Sharma, Presently working as a Associate Professor (HOD / Research Coordinator) Department of Computer Science Engineering and IT at "Shri Jagdish prasad Jhabarmal Tibrewala University", Chudela, Jhunjhunu (Rajasthan). He completed his B.Sc with Computer Application in the year of 2002 awarded from S.M.L. (P.G.) College, Jhunjhunu, University of Rajasthan, M.C.A from Modi Institute of Management and Technology, Kota, University of Kota in the year 2005, Ph.D. in Faculty of Computer Science, from "Shri Jagdish prasad Jhabarmal Tibrewala University", Jhunjhunu (Rajasthan) in the year 2014. His research areas Data Communication & Networking, Operating System, Computer Organization and Architecture, Data Mining, Image processing, Cloud Computing, Software Engineering etc. In his research life he Published 75 Papers in National and International journals, 25 National and International conferences, 03 workshops. Under his guidance 05 research scholars awarded Ph.D. Presently 08 research scholars working under his guidance. He invited as a Guest Lecturer for Students in M.Sc. I.T. (Master Program in Information Technology) on the behalf of National University of Science and Technology, Muscat, Oman, Nov. 2019. He is Paper Setter, Answer Sheet Evaluator (Copy Checker) and Practical Examiner in University of Rajasthan, Jaipur, Pandit Deendayal Upadhyay University (Shekhawati University), Sikar, Maharaja Ganga Singh University, Bikaner, University of Kota, Kota, Board of Secondary Education Rajasthan, Ajmer. Ph.D. He is Evaluator and Ph.D. Final Viva-Voce Examiner in OPJS University, NIMS University, Mewar University. Published 2 patents on Title: Parallel Processing System to Reduce Complexity in Data-Mining of Industrial & Social Big-Data. And Title: Computer Implemented Method for Detecting Downlink Control Channel in Long Term Evolution Wireless Communication. He is a member of IAENG, IACSIT, CSTA and UACEE.



**Ms. Mili Mary Varghese**

**Ms. Mili Mary Varghese**, faculty of the Department of Computing, Muscat College, Oman. Awarded Bachelor Degree in Mathematics by Kerala University and Master of Computer Application by Mahatma Gandhi University. Oracle Certified Professional (OCP) from Oracle Corporation. She has achieved Associate Fellow status of Higher Education Academy in 2019. She has more than 10 years of experience in programming and facilitating learning in different programming languages like Java, C, C++, Php, Python etc. . Her research area includes Big Data and Cloud Computing. She has published research papers in reputed Journals.

FOR AUTHOR USE ONLY





**Dr. S. Nagaprasad**

**Dr. S. Nagaprasad** working as a Faculty in Computer science and Applications, Dept. of Computer Science and Applications, Tara Government College, Sangareddy, Telangana state. He completed his B.C.A. in the year of 1998-2001 awarded from Osmania university, M.Sc (I.T.) in the year of 2001-2003 awarded from Sikkim Manipal University, Sikkim, his research work completed from Ph.D in Computer Science and Engineering, from Acharya Nagarjuna University, Sep-2015. His research areas data mining, networking, image processing, machine learning etc. In his research life he present completed 30 international journals in his research area, 15 National and International conferences, for his research interest he attended 10 workshops. He worked as a faculty in Computer Science at S.K.N.R. Government Arts and Science College, Jagtial Telangana state for 10 years, and he worked as faculty in Computer Science at S.R.R. Government Arts and Science College, Karimnagar Telangana state for 05 years. Presently he is Guiding 5 Ph.D. scholars in Computer Science and Engineering “Shri. Jagdishprasad Jhabarmal Tibrewala University”, Jhunjhunu (Rajasthan).

**Index**

S.No.	Name of the Topic	Page No
<b>Unit I</b>		18-30
<b>Fundamentals of Computers</b>	Introduction of the Computer	18
	Computer Components	18
	Input	18
	Central Processing Unit (CPU)	19
	Arithmetic Logical Unit (ALU)	19
	Control Unit	20
	Memory Unit	20
	Hardware	21
	Software	22
	System Software	22
	Application Software	22
	Operating System	22
	History of Operating System	22
	Features of Operating System	23
	Functions of Operating System	23
	Process Management	23
	Memory Management	23
	File Management	23
	Device Management	23
	I/O System Management	23
	Secondary Storage Management	23
	Security	24
	Command Interpretation	24
	Networking	24
	Communication Management	24
	Translators	24
Compiler	24	
Interpreter	24	

	Assembler	25
	Algorithm and Flow Chart	25
	Algorithm	26
	Pseudo Code	26
	Flow Chart	27
	Terminal	27
	Input / Output	27
	Processing	27
	Decision	27
	Connectors	27
	Flow Lines	28
	Multiple Choice Questions	28
	Fill in the Blanks	29
	Short Questions	29
	Review Questions	29
	Programming Exercises	29
<b>Unit-II C Programming Installation</b>		31-35
<b>Unit-III – Overview of C Language</b>		36-49
<b>Overview of C language</b>	Introduction	36
	History of C Language	36
	C Language Importance	38
	Features of C Programming Language	39
	The Procedural Language	39
	The Efficient and Fast	39
	The Modularity	40
	Type Statically	40
	General Purpose Language	40
	Built-in Operators Rich Set	40
	Rich Functions with Libraries	40
	Middle Level Language	40
	Portability	40
	Easy to Extend	40

	C Language Programming Structure	40
	Program Structure of the C Language	41
	Basic Structure of C Programs	43
	What errors we are getting in the compilation time	44
	Program Comments	45
	Unix System	47
	MS-DOS System	47
	Windows System	47
	Multiple Choice Questions	48
	Fill in the Blanks	48
	Short Questions	49
	Review Questions	49
	Programming Exercises	49
	<b>UNIT –IV</b>	50-68
<b>Constants, Variables and Data Types</b>		
	Introduction	50
	Character Set	50
	Source Character Set	50
	Alphabets	51
	Digits	51
	Special Symbols	51
	Execution Character Set	51
	Tokens in C Language	54
	Keywords	54
	Keyword Properties	54
	Identifiers	56
	Identifiers Rules	56
	Improved language Classification for Identifiers	56
	Operators	57
	Constants	57
	Integer Constants	58
Floating Point Constants	58	

Character Constants	58
String Constants	58
Variables	58
Declaration of a Variable	59
Declaration of a Syntax	59
Types of Variable	59
Local Variable	59
Global Variable	59
Static Variable	59
Automatic Variable	59
External Variable	59
Data Types in C	60
Primary Data Type	61
Integer Data Type	61
Floating Point Data Type	62
Character Data Type	62
Void Data Type	63
Enumerated Data Type	63
Derived Data Types	63
Variable declaration	63
Primary Type Declaration	63
Storage Class in C	64
Auto	65
Extern	65
Static	65
Register	65
Storage Class Declaration	66
Symbolic Constants Definition	66
Declaring a Variable as Constant	66
Declaring a Variable as Volatile	66
Multiple Choice Questions	67
Fill in the Blanks	67

	Short Questions	68
	Review Questions	68
	Programming Exercises	68
<b>Unit V</b>		69-85
<b>Operators and Expressions</b>	Introduction	69
	Arithmetic Operators	69
	Relational Operators	70
	Logical Operators	72
	Increment and Decrement Operator	72
	Pre Increment and Decrement Operator	73
	Post Increment and Decrement Operator	73
	Assignment Operators	74
	Bit-Wise Operator	75
	Conditional Operator	76
	Special Operator	77
	Sizeof Operator	77
	Pointer Operator	78
	Comma Operator	79
	Dot Operator	79
	Arithmetic Expression	79
	Evaluation of Expression	79
	Precedence of Arithmetic Operators	80
	Some Computational Problems	80
	Type Conversions in Expressions	80
	Implicit Type Conversion	80
	Explicit Type Conversion	81
	Operator Precedence and Associativity	82
	Mathematical Functions	83
	Multiple Choice Questions	84
	Fill in the Blanks	85
Short Questions	85	
Review Questions	85	

	Programming Exercises	85
<b>Unit VI</b>		86-95
<b>Managing Input and Output Operations</b>	Introduction	86
	C Language Input Functions	87
	Scanf () Function	87
	getchar () Function	87
	Getch (): function	88
	Gets() function	88
	Fscanf() function	89
	C Language Output Functions	89
	Printf () Function	89
	Formatted Printf () Function	92
	Putchar () Function	93
	Puts () Function	93
	FPrintf () Function	94
	Multiple Choice Questions	94
	Fill in the Blanks	94
	Short Questions	94
Review Questions	94	
Programming Exercises	95	
<b>Unit VII</b>		96-107
<b>Decision Making and Branching</b>	Introduction	96
	Decision Making	96
	If Statement in C Language	97
	Simple If Statement	97
	If Else Statement	98
	Nested If Statement	99
	If Else If Statement or If Else Ladder	101
	Switch Statement	102
	The Conditional (? :) Operator	104
	Break, Continue and GOTO Statements	105
	Goto Statement	106

	Multiple Choice Questions	106
	Fill in the Blanks	107
	Short Questions	107
	Review Questions	107
	Programming Exercises	107
<b>Unit VIII</b>		108-116
<b>Decision Making and Looping</b>	Introduction	108
	While Statement	108
	While Statement Properties	108
	do while Statement	110
	For Loop Statement	111
	Jumps in Loops	112
	Jumping Out of a Loop	113
	Break Statement	113
	Continue Statement	114
	Concise Test Expressions	115
	Multiple Choice Questions	115
	Fill in the Blanks	115
	Short Questions	116
	Review Questions	116
Programming Exercises	116	
<b>Unit IX</b>		117-131
<b>Arrays</b>	Introduction	117
	Declaration of an Array	118
	Accessing Individual Elements of an Array	119
	One-Dimensional Arrays	121
	Declaration of a Single or One Dimensional Array	121
	Initialization of Single or One Dimensional Array	121
	Accessing Elements of Single or One Dimensional Array	121
	Two Dimensional Array	122
	Declaration of Two Dimensional Array	122
	Data Type	123



	Name of the Array	123
	Row Size	123
	Column Size	123
	Two Dimensional Array in C Initialization	123
	First Approach	123
	Second Approach	124
	Access of Two Dimensional Array Elements	124
	Multi - Dimensional Array	124
	Multi-Dimensional Array Initialization	126
	Dynamic Arrays	127
	Malloc()	127
	Calloc()	128
	Realloc ()	128
	Free ()	129
	Multiple Choice Questions	130
	Fill in the Blanks	130
	Short Questions	130
	Review Questions	130
	Programming Exercises	131
	<b>Unit X</b>	
<b>Character Arrays and Strings</b>	Introduction	132
	Creating String in C Programming Language	132
	Declaring and Initializing String Variables	132
	Terminating Null Character	134
	Reading Strings from Terminal	134
	Reading a Line of Text	136
	Using getchar and gets	136
	Writing Strings to Screen	137
	Arithmetic Operations on Characters	137
	Putting Strings Together	138
	Comparison of Two Strings	138
	String handling functions in C	139

	Table of Strings	140
	Multiple Choice Questions	140
	Fill in the Blanks	141
	Short Questions	141
	Review Questions	141
	Programming Exercises	141
<b>Unit – XI</b>		142-160
<b>User Defined Functions</b>	Introduction	142
	Need for User Defined Functions	142
	A Multi-Function Program	143
	Modular Programming	145
	Elements of User Defined Functions	145
	Definition of Functions	146
	Function Header	146
	Name and Type	146
	Formal Parameter List	146
	Function Body	147
	Return Values and their Types	147
	Function Calls	148
	Function Call	149
	Function Declaration	149
	Category of C Functions	149
	Function with No arguments and No return values	150
	Function with arguments but No return values	150
	Function with No arguments but return values	151
	Function with arguments and return values	152
	Nesting of Functions in C	153
	Recursion in C	153
	Passing Arrays to Functions	155
	Searching and Sorting	156
	Sorting the Data	156
Passing Strings to Functions	156	

	Basic Rules	157
	The Scope, Visibility and lifetime of variables	157
	Automatic Variables	158
	External Variables	158
	Static Variables	158
	Register Variables	158
	Multi File Programs	159
	Multiple Choice Questions	159
	Fill in the Blanks	159
	Short Questions	159
	Review Questions	160
	Programming Exercises	160
<b>Unit – XII</b>		161-176
<b>Structure and Unions</b>	Introduction	161
	Defining a Structure	161
	Memory Allocation of Structure	161
	Arrays vs Structures	162
	Creating Structure Variables	162
	Structure variables are created and used	162
	Type Defined Structures	163
	Accessing Structure Members	164
	Structure Initialization	165
	Operations on Individual Members	167
	Arrays of Structures	167
	Declaring C Array of Structures	168
	Arrays within Structures	168
	Purpose of an array within Structure	168
	Structure within Structures	169
	Structures and Functions	169
	Call by Value	170
	Call by Reference	171
Unions	171	

	Accessing Union Members	173
	Size of Structures	173
	Bit Fields	174
	Declaring Bit Fields	175
	Data Type	175
	Multiple Choice Questions	175
	Fill in the Blanks	176
	Short Questions	176
	Review Questions	176
	Programming Exercises	176
	<b>Unit – XIII</b>	177-185
<b>Pointers in C</b>	Introduction	177
	Understanding Pointers	177
	Underlying Concepts of Pointers	178
	Pointers in C	179
	Accessing the Address of Variables	179
	Declaring Pointers	179
	Assigning Address to Pointer	180
	Accessing Variable Value Using Pointer	180
	Memory Allocation of Pointer Variables	180
	Chain of Pointers	180
	Pointer Expression	180
	Pointer Arithmetic Operations in C	181
	Pointer Increments and Scale Factor	181
	Rules of pointer Operations	182
	Pointer and Arrays	182
	Pointers and Character Strings	183
	Arrays of Pointers	183
	Pointers as Function Arguments	183
	Functions Returning Pointers	183
	Pointers to Functions	184
Pointers and Structures	184	

	Multiple Choice Questions	184
	Fill in the Blanks	185
	Short Questions	185
	Review Questions	185
	Programming Exercises	185
<b>Unit – XIV</b>		186-193
<b>File Management in C Language</b>	Introduction	186
	Defining and Opening a File	186
	File Name	187
	Data Structure	187
	Purpose	187
	Closing a File	187
	Input / Output Operations on Files	188
	Character Input / Output Operation	188
	Fputc()	188
	Fgetc()	188
	Getc () and putc ()	188
	Integer Input / Output Operations	188
	Putw ()	188
	Getw ()	189
	String I/O	189
	fputs()	189
	fgets()	189
	Formatted I/O	189
	fprintf()	189
	fscanf()	189
	Block read / Write	189
	fwrite()	190
	fread()	190
	Error Handling During I/O Operations	190
Random Access to Files	191	
Fseek()	191	

File Pointer	191
Displacement	191
Pointer Position	191
Ftell ()	191
Rewind ()	191
Command Line Arguments	191
Multiple Choice Questions	192
Fill in the Blanks	192
Short Questions	192
Review Questions	192
Programming Exercises	193

FOR AUTHOR USE ONLY

## Unit I

### Fundamentals of Computers

#### Main Objectives of the Unit:

It should be complete this unit:

- ❖ Know the machine system and know how it functions
- ❖ The apprenticeship in designing reasoning using algorithms and maps

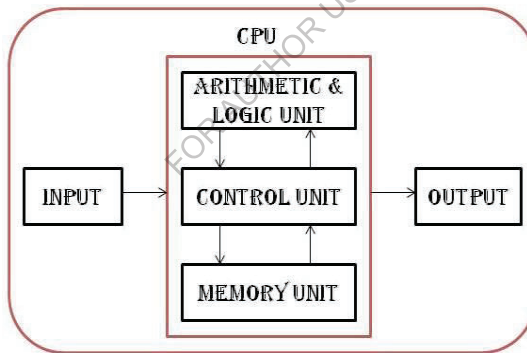
#### Introduction of the Computer:

The machine is an electronic system that deals with memory specified instructions. A machine may collect data from the user through input devices; process the provided user's data (Processing; Output); and save data (Information) for future usage. You may describe a machine as follows:

The machine is an electronic system that runs under the influence of the instructions placed in his memory which collects data from the user which saves data for potential usage.

#### Computer Components:

A computer offers a statistical and rational measurement electronic unit. Figure displays the main components of a computer.



**Figure 1.1.: Computer Components.**

The central processor, input and output devices and storage facilities are the core device components as seen in figure 1.1. These components communicate through buses, as seen in Figure 1.1.

#### Input:

Input devices and other commands can be used by users to enter data on the computer. The most common alternative for input devices is the mouse and the keyboard. Examples include

scanner, reader, etc. Most input devices are electro-mechanical instruments that are manually operated by the user input by pressing keys, for example, to insert details or to transfer the mouse to the computer. The captured data is digitised by a CPU or memory using mechanical methods.

### Central Processing Unit (CPU):

The Processor is the main part of a machine for guidance. It functions on the OS and software, receives user comments or working data. It collects the data and generates the output that can be processed or accessed on your computer. It is a central workstation.

The CPU is the real chip with at least one core measured within the device. Most CPUs have just one processor for many years. However, a single CPU often has 2 or 3 processors or "core processors," defined as a double core CPU and a four-core CPU. That is more popular.

There is a Processor for certain users.

Six (Hexacore) or eight (Octocore) may be mounted on the high-end CPUs. More than a CPU is feasible on a machine with many key components. For example, on a two Hexacore CPU server there are 12 processors.

ALU, FPU, registry and L1 cache typically varies from each Kernel core edition. In some cases the processor core will have its own L2 cache, but the same L2 cache can be shared. Just one frontal bus route is usable in the Kernel and system memory.

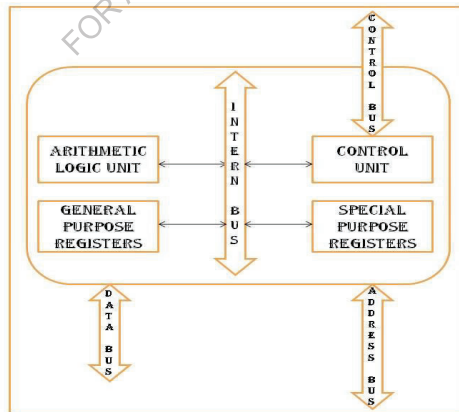


Figure 1.2.: Layout Model of the CPU

### Arithmetic Logic Unit (ALU):

In the machine, in addition to arithmetical operations (e.g. addition, subtraction, etc.) an Arithmetic Logic Unit (ALU) is usable, and is capable of carrying out logical operations (for



example AND, OR, Ex-OR, Invert, etc.). The control unit provides the data that the ALU requires from memories or input devices and guides the ALU, on the basis of the memory commands, to execute a special procedure. ALU is the computer's "calculator" component.

The key part of a computer system's core function unit is an arithmetic logic device (ALU). It handles both arithmetic and logic operations to be completed with instruction phrases. The ALU is divided in arithmetic units (AU) and logic units (LU) in some microprocessor architectures.

An ALU may be equipped for several different operations by engineers. As the operations are more complicated, the ALU would also raise its expense and take up more room in the Processor and dissipate more heat. That is why engineers render ALU as efficient as possible to ensure that the CPU is strong and quick, but is not so complicated that its costs and other drawbacks are prohibitive.

ALU is considered an Integer Unit (IU) as well. The arithmetic logic device for all calculations required by the CPU is this section of the Hardware. Logical in fact several of these activities. It can control the Processor, but also absorbs more energy and emits more heat, depending on how the ALU is built. Therefore it is important to manage the strength and efficiency of the ALU and the expense of the whole machine. This is why faster CPUs are costly, power-consuming and heat dissipating.

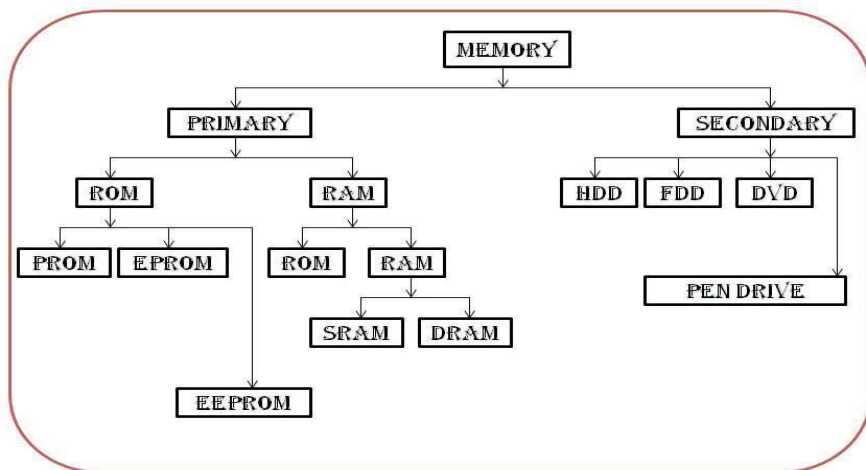
#### **Control Unit (CU):**

A Control Device or CU is a computer processing module. Computer commands may be tailored to machine logic, data, input and output machines. Examples of devices utilising control structures involve CPUs and GPUs.

A device control captures and translates input data and transfers them to the central processor for feedback signals. You will then ask the attached hardware what to do on the computer. The roles of the control device vary in configuration between the numerous vendors, based on the type of a CPU. The explanation below demonstrates how to operate a computer.

#### **Memory Unit (MU)**

The data retrieval unit contains the memory or recording system. The unit is variable in pace and variance. Entry speed indicates time for memory data to be stored or recovered. The period(s) during which information is processed in the brain corresponds to uncertainty. The machine memory may be separated into various groups dependent on these characteristics. Figure 1.3 indicates the machine's power gradation.



**Figure 1.3: Computer Memory-Classification**

As seen in Figure 1.3, the memories are usually described as primary and secondary memories.

The main storage is often called the primary storage as the Kernel will communicate directly from the address bus from its database ad. The RAM and ROM are the main components. There are two major forms for all versions (read only memory). RAM is a complicated storage system with facts and guidance for operating apps. The RAM is unstable because it kills all accumulated data while power is off the computer. ROM is a non-volatile read-only site, in contrast, which generates all read advertisements. And when a device is disconnected the ROM is non-volatile, preserving the accumulated data. The ROM and the other necessary routines are included in the boot strap loader. Boot Strap Loading is the device used to charge a portion of the operating system's secondary memory ram up until it has been powered and control is carried out in self-test.

In this respect, a big, non-volatile storage capacity, which can hold knowledge, often known as an auxiliary memory, as many times as we like, allows secondary memory lighter than access time of the primary memory. The flash drives are a second example safety.

#### **Hardware:**

Device hardware requires an electric, hydraulic, electrical and user bus system. The circuitry of the device is, in other terms, a bare unit. By contrast, the programme offers a range of instructions that enable bare equipment to function and achieve our targets, orders,

commands, codes, etc. The programme is the heart of the computer while the machinery is the skeleton. Hardware and software are necessary for computers to function.

**Software:**

Code is a series of programmes programmed to operate in certain respects. Computer series Structured guidelines for problem solving.

Two kinds of applications occur

**System Software:**

Computer technology comprises of a variety of information analysis, control and extension feature applications. The programme is typically produced by producers of computers. The software consists of programmes of low levels, which communicate with the hardware on a very basic basis. The communication of the device with the end user is used in machine software.

**Application Software:**

Computer programmes satisfy the basic criteria of a given area. Computer programmes. In the division of software apps, all software applications produced in the lab would be listed. A single simple text editing interface may contain the framework tools. Microsoft notepad, for example. A number of applications may be used, often known as a software package, to make a laptop operate effectively.

**Operating System (OS):**

An OS is a device that serves as an intermediary between the end-use and the hardware of the machine. To run other programmes, each machine must have at least one OS. An app such as Chrome, MS Phrase, Sports, etc requires to run and work in a certain area. The App allows you to interact with the machine without being acquainted with the language of the device. Without an operating system, the consumer is unable to access either machine or mobile device.

**History of Operating System:**

- ❖ The first programmes to handle tape storage in the late 1950's were
- ❖ In the early 1950's, the General Motors Development Centre introduced the first IBM 701 OS.
- ❖ Betriebssystemen began utilising drives in the mid-1960s.
- ❖ The first Unix OS version was developed in the late 1960s.
- ❖ Microsoft's first App was DOS.
- ❖ The 86 DOS software was built by a Seattle company in 1981.

- ❖ In 1985 when an Interface was developed and combined with MS-DOS, the latest common App Windows first appeared.

### **Features of an Operating System:**

A list of essential features typically contained in an operating system is as follows:

Mode of security and control

Allows disc and file systems connect Computer drivers Protection Networking

Rolling of the Software

Digital Computer Multitasking computer control

I / O project Management

File Exploitation

Detection and correction of errors

Allotment of land

Security of knowledge and services

### **Functions of an Operating System:**

Each role executes in operating system software:

#### **Process Management:**

Service management assists Software in task development and elimination. It also offers coordination and collaboration frameworks between systems.

#### **Memory Management:**

The memory management module performs the task of assigning and unassigning memory space to programmes requiring those facilities.

#### **File Management:**

It manages all activities related to files, like organisational storage, recovery, naming, sharing and file protection.

#### **Device Management:**

Device management maintains all device tracks. The I / O controller is also known as this module responsible for this task. The assignment and de-allocation of devices is also carried out.

#### **I/O System Management:**

One of the primary goals in every framework is to conceal the user's peculiarities.

#### **Secondary Storage Management:**

Systems include multiple layers of storage which includes primary storage, secondary storage, and cache storage. Instructions and details must be placed in primary storage or cache such that a running programme may access it.

**Security:**

The protection module safeguards operating machine data and knowledge against the danger of ransom ware and allowed entry.

**Command Interpretation:**

This module interprets commands provided to process these commands by the and operating machine tools.

**Networking:**

A distributed machine is a collection of processors who do not share a clock, memory or hardware. Via the network, processors interact within themselves.

**Communication Management:**

Coordination and assignment by the different computer system users of compilers, interpreters and other software resources.

**Translators:**

This is a concern because computers tend to use a combination of high and low programming languages. Computers only understand machine code (binary).

In order to get around the issue, a translator must transfer the high-level and low-level software code (source code).

The source code (object code) is translated into computer code by a translator.

There are several forms of translator systems, each of which may perform multiple functions.

**Compiler:**

Compilers are used to convert into computer code (object code) a programme written in a high language. The translated software disc, until assembled (all in one go), can then be used directly on the server, and can then be run separately. It can take some time to compile, but without recompilation the translated software is constantly used.

After complete software translation, an error report is sometimes made. Errors may trigger a machine to crash in the code of the software. Only by updating the initial source code and recompilation of the software will these errors be corrected.

**Interpreter:**

Interpreter systems provide one sentence from the high-level language software to interpret, interpret and implement. When a line of code representing an error is crossed, the interpreter ceases.

During the creation of a programme, interpreters are also used. They improve the debugging phase by reviewing and testing single line of code before starting. Interpreted applications are automatically started, although the software will start a completed file later.

There is no executable programme. Any time you start it, the software would be translated from scratch.

**Assembler:**

Assemblers are used to translate a programme into a machine code (object code) file written in a low-level assembly language to use and execute it on the computer.

The programme file can be used repeatedly without re-assembling once assembled.

**Algorithm and Flowchart:**

In the last part, we see that the programme is transparent, but important to the machine, and helps users to connect with the device and function to accomplish a practical goal depending on their information.

Software is a series of programmes to carry out a project together. Software is a set of commands that carry out a process in a sequence. See a software illustration that permits the user to join, delegate, and display the outcomes during the execution of two items. The software includes three sets of directions: firstly, it allows users to enter the data; secondly it allows the numbers to be inserted and thirdly, the results to be shown.

These three sets will have been programmed by the programmer as the software was drawn up and definitions used to execute them. The explanation is exceedingly clear.

However, more frequently than not, programming bugs can be overcome. For these complex problems, difficult strategies remain important. Related methods of dynamic. Responsive observations and methodological methods. The need for the hour is a way of demonstrating the necessary programme rationale and of converting it into a specific series of instructions for solving this problem with the computer.

Two such tools are the algorithms and flowcharts. These techniques are distinct from both the programming language and the goal computer. The argument in each of these instruments will then be used to compose programmes in each programmer's language of choice.

In reality, programming languages can often be logically autonomous. If the programming comprises three primary logic types, independent of the process and language used in writing).

Imperative Statements

Conditional Statements

Loop Statements

The instruments we work on as a programme also have statements in one of those three categories; each of these three types of structures should be defined. First, tools like algorithms and flowcharts should be used to formulate and write a programme.

Half was finished; it was expected, after all.

**Algorithm:**

An algorithm is a series of instructions to execute a specified operation. For e.g. it can be an easy way to merge the two numbers or to play a compressed video file. Search engines are used to view the most relevant search index returns using patented algorithms for specific queries.

Algorithms are also developed in computer engineering. These traits may be applied to in a wide network, limited systems. For instance, a picture viewing programme can be used to build a function library with a custom algorithm in order to produce different image file formats. The software for image-editing will provide algorithms for image processing.

E.g. chopping, re-sizing, sharpening, blurring, growing the red eyes and adding light.

A software machine can often execute a certain function in a variety of ways. But programmers normally try to construct algorithms for the most performance. Developers may guarantee that their applications are running by consuming limited machine resources as soon as possible with extremely effective algorithms. Not all algorithms is, of course, preferably established first.

Thus, in upcoming app upgrades developers also boost existing algorithms. The most recent upgrade concentrates on better algorithms sometimes when you see a new iteration of "optimised" or "faster moving" tech.

Definition of an algorithm to connect two user-input numbers

```
Step 1: Start
Step 2: Declare Variables num1, num2 and sum.
Step 3: Read values num1 and num2
Step 4: Add num1 and num2 and assign the result to sum sum=num1+num2
Step 5: Display sum
Step 6: Stop
```

**Pseudo Code:**

Pseudo code is a widely employed term in algorithms and fields of computing. The compiler outlines the implementation of an algorithm. This is a process. This is a system. This is literally to suggest that the cooked algorithm is reflected. With pseudo codes, programmers sometimes reflect algorithms irrespective of their programming knowledge, or background. Like the name means, the pseudo code is a deceptive language or machine model that a layman with some programming skills may only comprehend. Just a single English

descriptive and annotative algorithm can be used. The machine cannot compile or view the programming language like any other code.

**Flowchart:**

Flowchart is a schematic of the progression of a programme's sequential steps. Flowcharts utilise basic graphical structures to display associations and process-data flow processes and arrows.

**Terminal:**

The oval symbol represents the start, stop and end in the logic flow of a system. In certain cases a pause / stop is usually used in device logic. The first and last characters of the flowchart are the portal.

**Input/Output:**

For an input / output element, a parallelogram is used. In the parallel graph in the flow map are shown the computer instructions for input and output on the output unit.

**Processing:**

In an integer box there are guidelines. All activities such as addition , subtraction and division are labelled with the symbol or the phase symbol.

**Decision:**

The symbol of diamonds is the point of judgement. Diamonds are shown in the flowchart for decisions like yes / no question or true / fake.

**Connectors:**

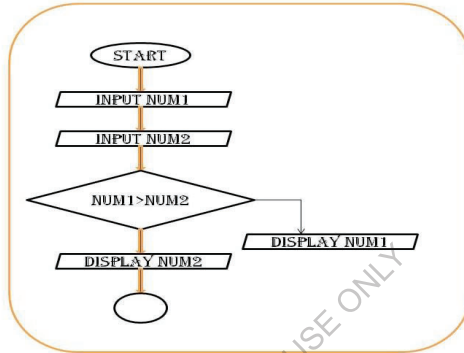
Connectors are helpful when a flux map is complex or distributed over more than one tab to prevent misunderstanding. There's a loop visible.





**Flow Lines:**

Flow lines indicate the approximate execution sequence of orders. Arrows are the control flow path and the interaction between the various flowchart symbols.

**Example of flowchart for two numbers below:**

**Figure 1.4: Flowchart Example**

**Multiple Choice Questions (MCQ)**

- The screen name for which the touch entry is recognised is:  
A) Touch Screen B) Screen C) Android Screen D) None

Ans: A

- Computer Screen is known as  
A) CU B) VDU C) CDU D) DVD

Ans: B

- Which is the product of a programme operation?  
A) Information B) Data C) Software D) None

Ans: A

- The printed output is called from a computer  
A) Paper B) Softcopy C) Hardcopy D) None

Ans: C

- The first machines have been programmed  
A) Machine Language B) Source Language C) Binary Language D) None

Ans: A

**Fill in the Blanks:**

- 1) ALU Stands for \_\_\_\_\_
- 2) BCD Stands for \_\_\_\_\_
- 3) RAM Stands for \_\_\_\_\_
- 4) ASCII Stands for \_\_\_\_\_
- 5) CPU Stands for \_\_\_\_\_

**Short Question:**

1. Explain about Computers
2. What is RAM
3. Describe Software
4. Explain about Flowchart

**Review Questions:**

1. What is an Operating System?
2. Explain components of a Computer?

**Programming Exercises:**

1. Write an algorithm and print the 1 to 100 figures on a flowchart
2. Write and draw a flow chart that transforms Fahrenheit to Celsius temperature.
3. Write an algorithm and create a flowchart Decide whether a pupil has finished the test or not.
4. Write down an algorithm to locate the total of 3 numbers
5. Write an algorithm and create a flowchart that displays the square region and perimeter
6. Write and create an algorithm to find simple interest
7. Enter an algorithm and draw a flowchart to swap 2 numbers with a time variable
8. Write an algorithm and create an outline to locate the maximum of two digits
9. Write an algorithm to determine the maximum of the three numbers
10. Write an algorithm and create a flowchart to locate from 1 to 100 numbers.
11. Write an algorithm and create an outline to identify a disability number between 1 and 100
12. Write an algorithm and create an outline for the description of series 1 to N
13. Write an algorithm to print a number multiplier table and create a floor charts
14. Write an equation and create a flowchart for the complete and average of such amounts.

15. Write an algorithm and create a flow chart to decide whether or not a number is first.
16. Write an algorithm and create a flowchart to find two numbers GCD and LCM
17. Write an algorithm and create a flow diagram to define the entire number divider.

FOR AUTHOR USE ONLY

## Unit II

### C Programming Language Installation

#### Main Objectives of the Unit:

It should be complete this unit:

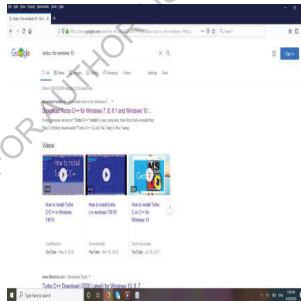
- ❖ Know Installation of C software
- ❖ View of C Programming

How to configure the framework for C language:

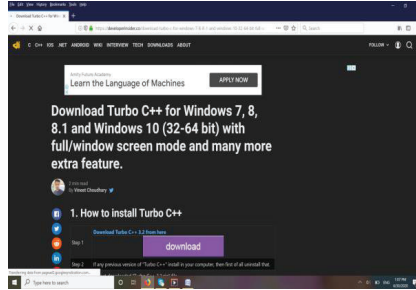
The Turbo C++ programming software has to be downloaded first. It's a 32-bit or 64-bit system, we have to check it. If you press Turbo C++ downloads, you have to go to the update folder and to select turbo C++3.1, then press on the Turbo C++ preview page, for Windows 7,8,8.1, and Windows 10 (32-64 Bits). An android, what we need to install is an executable programme.

First, the browser and type google you need to open

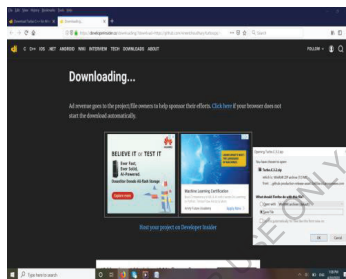
This screen will then be seen in the browser in Google Style turbo c for Windows 10, then press the Open Turbo C++ tab.



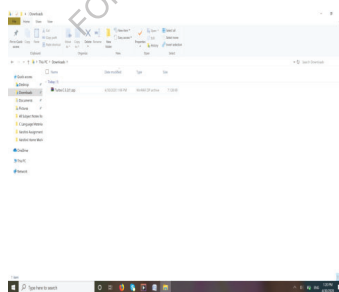
The lower screen would have appeared after clicking on this link. And the update button is pressed.



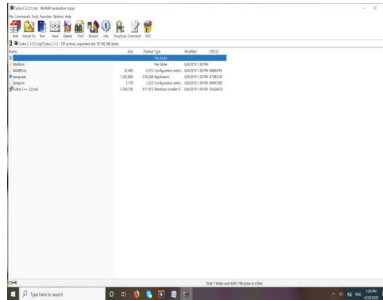
The computer would be like this if you tapped on the download button. It is saved in the file folder when you press on the all right icon.



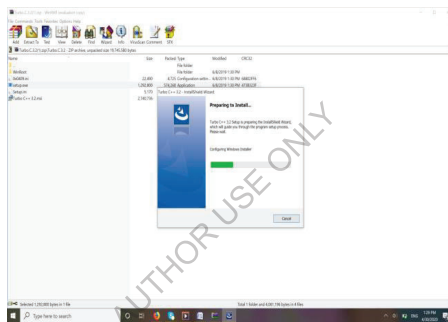
We now go to the folder that we downloaded and click the file. The display is as follows:



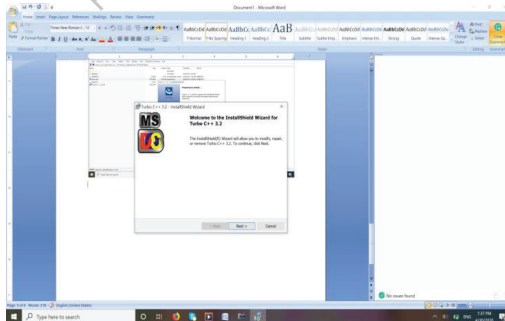
When we press Turbo C 3.2 the screen is this direction.



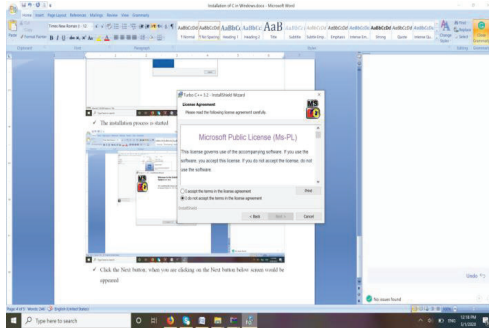
Then we click on the configuration. The processing of the installation has started here; we see the display of the window preparation.



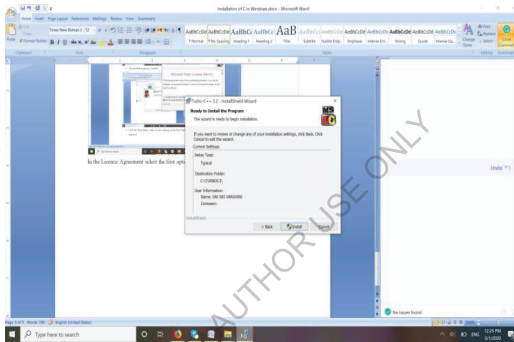
The phase of implementation began



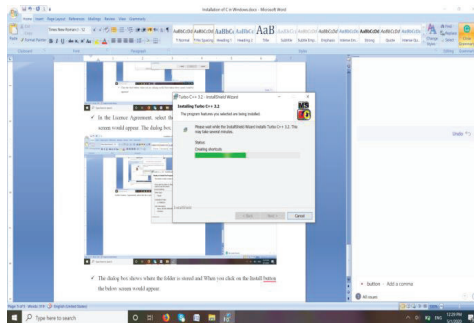
Click on the Next tab, and it will activate as you press the Next tab below the computer. We have two choices in the licence agreement: one recognises the terms and the other does not approve the terms;



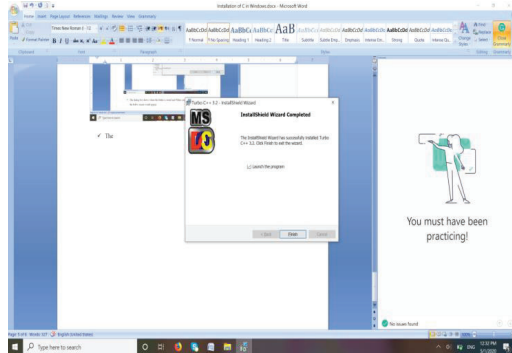
In the Licence Agreement, select the first option and click the next button, the below screen would appear. The dialogue box is ready to install the program.



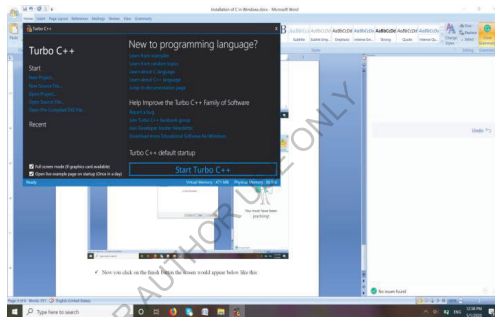
The dialogue box indicates the location of the folder and the installation process is started when you press the install button.



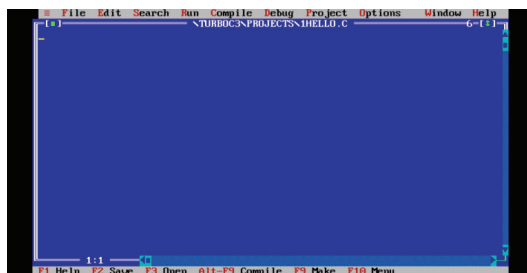
Underneath the screen, the installation process is successful.



The computer appears in the following manner, now you press on the finish icon.



Clicking the Start Turbo C++ will send you the following picture to compose programmes in the browser.





## UNIT III

### Overview of C language

#### Main Objective of the Unit:

It should be complete this unit:

- ❖ Produce a description of the vocabulary of C
- ❖ Example the essential C principles by sample programs
- ❖ Example programs demonstrate the usage of user-defined functions and math functions
- ❖ Describe the underlying C-program framework
- ❖ Recognize the C language programming type
- ❖ Describe how to compile and operate a C program

#### Introduction:

The programming method, which allows it easy for learners and experts to easily grasp C language tutorials. Our C tutorial describes for every subject. Dennis Ritchie develops the C language to create machine applications which interact directly with hardware devices such as drivers, kernels and so on. C programming shall be regarded as the foundation of other programming languages and hence its mother tongue.

#### History of C Language:

For a programming language, C seems a strange term. Nonetheless, this weird sounding language is one of today's most common computer languages, since it is an individual standardized, machine-level language. This helps app engineers to build applications where they are applied, without caring about the hardware platforms.

The ALGOL, adopted in the early 1960s, is the origin of all current languages. ALGOL utilized the block form of the first programming language. While it was never common in the United States, in Europe it was commonly used. ALGOL presented the computer science world with the idea of formal programming. It term popularized in the 1960s by computer scientists such as Corrado Bohm, Guiseppe Jacopini and Edsger Dijkstra. A variety of languages have been revealed subsequently.

In 1967, Martin Richards primarily created a language named BCPL for the writing of machine applications. In 1970, Ken Thompson developed and referred to a language with various features of the BCPL. B was used to build early variants of the Bell Laboratories UNIX operating system. The machine programming language was "typeless," both BCPL and B.

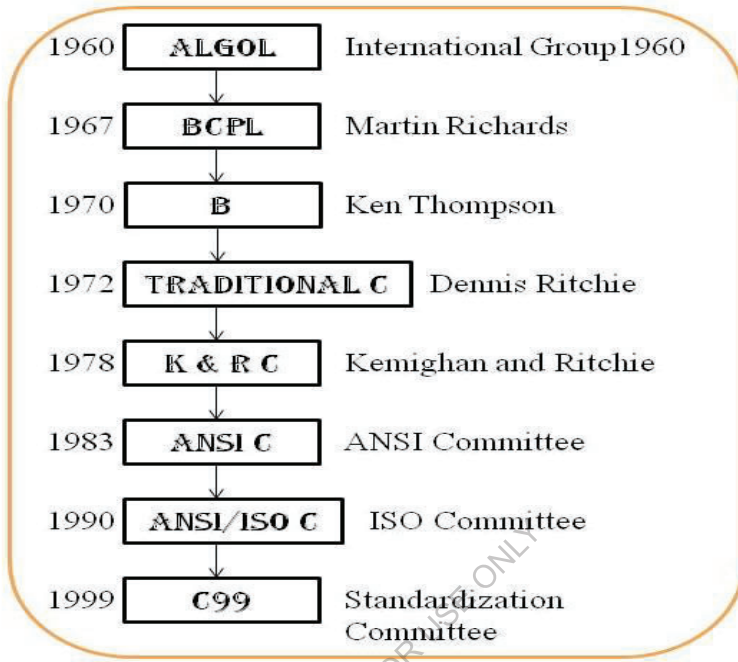
Throughout 1972, Dennis Ritchie at the Bell Labs developed from ALGOL, BCPL, and B. The definition of data forms and other important functionality were implemented utilizing other terms from certain languages. It's closely aligned with UNIX because it was built along with the UNIX operating system. In Bell Laboratories, the operating system has been encoded almost entirely in C. UNIX is one of today's most common network operating systems and the core of the superhighway of Internet applications.

For several years C was used mainly in research contexts, but eventually it became commonly accepted by programming professionals with the release of several C compilers for commercial purposes and the increasing success of UNIX. Today, C operates on a broad variety of hardware and operating systems.

In the 1970s, C was turned into the conventional C that is now established. The language became popular with Brian Kerningham and Dennis Ritchie in 1978 after the release of the book "The C programming language." The book became so famous that the programming world named the K&RC vocabulary. The rapid growth of C contributed to the creation of many related yet mostly incompatible language variants. For device developers, that posed a serious problem.

In 1983 ANSI formed a formal committee to develop a C format and guarantee that the C language remains a norm. In December 1989, the Committee approved a variant C now known as ANSI C. The International Standard Organization (ISO) accepted it in 1990. The C edition is sometimes known as the C89 variant.

In the 1990's, C++, an originally C language, was updated and modified and in November of 1977 it became a language that was accepted by ANSI / ISO. C++ also introduced several new features to C to make it more flexible, and not just a pure object-oriented language. Around this time, the modern JAVA language C and C++ was developed by Sun Micro Systems from USA.



**Figure. 3.1. C Language History**

The dynamic nature of all popular computer languages. By adding new functionality, you can continue to boost the capacity and reach and C is no exception. Although C++ and Java were developed from C, the C standardization committee considered that the usefulness of the language would be improved by a number of C++/Java features if added to C. The 1999 standard for C was the result. Currently this update is labeled C99. Figure 4 demonstrates the past and growth of C. Although C99 is an updated edition, several compilers still don't embrace all the latest features found in C99. We therefore discuss all the new features added in an annex by C99 separately so that interested readers can refer to and use new material as quickly as possible.

#### **C Language Importance:**

It is probably due to its many attractive qualities that C is increasingly common. It is a versatile language that can compose any complicated program with a rich set-down built into functions and operators. The compiler C combines the ability of the ab assembly language with the functions of the highest standard language and is thus ideal for the writing of software for systems and business packages.

C-written programs are effective and quick. This is because of its diversity of data types and strong operators. It's much quicker than BASIC. For eg, it takes around one second for a program to increase a variable from 0 to 15000 in C and more than fifty seconds for a Simple interpreter.

ANSI C comprises just 32 keywords, and its value lies in its incorporated functionality. There are a number of standard functions which can be used for program development.

C's extremely mobile. So, C programs written for one machine will operate with little to no modifications on another. If we plan to use a new computer that has another operating system, portability is important.

C language is ideally suited for structured programming, so that the user has to consider a problem as regards function modules or blocks. A complete program would take place in a proper collection of these modules. This modular structure makes it easier to debug, test and maintain the program.

C's ability to expand itself is another important feature. Basically, a C program is a collection of functions that the C library supports. We should connect our own functions continuously to the C collection. The programming task becomes easy with the provision of a large number of functions.

We will examine a few of the C programs and analyze and understand how they work before discussing certain features of C.

### **Features of C Programming Language:**

C is a term of action. Dennis Ritchie first created it in 1972. This has mainly been designed to construct an operating system as a programming language. Such features render C language ideal for machine programming like an OS or compiler creation, and involve low-level memory, clear keywords and clean design.

#### **1. The Procedural Language:**

Predefined instructions are executed step by step in a procedural language such as C. C can contain more than one role to perform a particular task. This is the only way that new people think of programming works in a specific programming language. In the field of computing there are other computer paradigms too. The most popular paradigm is an object-oriented language of programming.

#### **2. The Efficient and Fast:**

Newer languages such as java, python provide more functionality than c, but their output rate drop effectively due to the extra processing of these languages. The mid-range programming language allows programmers exposure to easy control of the computing equipment, but

languages at a higher degree do not. This is one of the reasons why C is the first choice for beginning language programming. It's fast, because static languages are faster than languages that are dynamically typed.

### **3. The Modularity:**

Modularity is called the concept of storing C programming language code as libraries for further use. The bulk of its power is provided by its libraries by this programming language truck. C language has its own library for common problems, such as using a header file stored in its library to help us use a certain function.

### **4. Type Statically:**

The vocabulary for programming C is static. The variable sort is tested when compiled, but not when operating. Means that the author will note the form of variables used every time a program is typed.

### **5. General Purpose Language:**

C programming language is used in different applications, from system programming to photo editing software. The following are some of the common applications used:

### **6. Built-in Operators Rich Set:**

This is a dynamic language that contains a wide number of embedded operators that use complicated or condensed C systems in prose.

### **7. Rich Functions with Libraries:**

Robust libraries and C apps also allow a novice to quickly write.

### **8. Middle Level Language:**

Since it is a medium-size language, it has both the assembly language and the high-level software functionality.

### **9. Portability:**

C language is extensively portable as programs written in C can run and compile with none or little changes on every system.

### **10. Easy to Extend:**

C-language programs can be extended to include more functionalities and operations if a program already has been written into it.

### **C Language Programming Structure:**

When we type a C program used by the compiler, it compiles the code into the machine language code. A machine language code can be understood by the computer. The C program code cannot be directly understood. However, it is easier for us to write C programming. That is why we will join C systems in which high-level languages are unique to the English and

machine language learning protocols, which are usually used only instead of typing code directly in the computer language. This makes it easier

For us to write C program and then use the compiler, a system program that requires C compiles for machine language learning for us again. A quick guide to the programming phase where a program meets the so-called programming cycle which involves three parts: one is to create an application; the second is to compile an application and the third is running the application. In the first part, when we write a scheme, we first save it and then modify, modify or edit the scheme. In the second part, when the program is compiled, when it gets results. The compilation part is complete. The application will revert to the editing step if errors are received. If the program results in the third stage of running the program, the output is not correct; the editing step will return. The output is not correct.

### **Program Structure of the C Language:**

We know the concept of C language programming until we study the fundamentals of C language programming first. Explore the teaching in programmers in the C language.

#### **First Program:**

```
#include <stdio.h>
#include <conio.h>
main()
{
printf("Hai");
}
```

Programming explanation:

#### **#include<stdio.h>**

This is a preprocessor command which tells the C programming compiler to include the standard input and output. Header files <stdio.h> before going to actual compilation. The first line to start the # include program is a pre-processed command.

#### **main()**

The main function is supposed to be. Both C programs begin with the main function, which ends with the first declaration of the main function. It is also part of the Syntax of the parenthesis).

#### **Printf(“Hai”);**

Now that there is only one line in the primary function that says "printf," printf is a feature named "C Program output." You include it in double quotes to print a particular message.

Everything in the repeated quotations is then printed on the computer. And here we mention one more point, each of which ends with a semicolon.

**Note:** These are the additional symbols of punctuation that tell you the C declaration symbols are valid. (#, {, 'and }).

### **Printf () and Scanf () in C language:**

The inbuilt application functions in the C programming language are the Printf() and scanf() functions accessible as usual from the C library. Such functions are defined and macros in "stdio.h," a header file in the language of C, are stated.

For use these printf ()and scanf() library functions in C language, use the file "stdio.h," as seen in the following C Programme.

#### **Printf():**

The printing of production is one of these operations in every method. We use the output to request a user's data, and then show status / result, calculations etc. There are several functions to print structured output in C programming. The printf() function which writes output to the display of the computer is discussed here. The stdio library must be included in the source code to use the printf() function.

A "word, set, float, integer, octal and hexadecimal number" feature in C-Standard is used to print onto the display device. We are using printf() to show the value of an integer variable with a percent d type specifier.

- ❖ Add to the character list, percent f is used for variable float, percent s for variable line, percent lf for multiple, and percent f for hexadecimal.
- ❖ We use "\n" in C printf() statements to create a newline. The case-sensitive language is C.

Printf() and scanf(), for example, differ from Printf() and Scanf(). In the lower case it is needed

to be all characters in printf) (and scanf).

#### **Scanf():**

- ❖ Scanf () is used for reading text, string and numerical data from keyboards in the C programming language. Select the following example program in which the user enters a character. The meaning is allocated to and shown in the vector "Ch."
- ❖ The value of the variable "str" is assigned to the Then, and then displayed, the user enters a string.
- ❖ RESERVE The scanf () statement using the format set %d. So, the entered amount is obtained for string as an integer and percentage s.

- ❖ The scanf() sentence as & ch uses Ampersand before the field name "ch."

It's like a pointer used to signify the element.

**Basic Structure of C Programs:**

- ❖ The descriptions mentioned so far demonstrate that a series of building blocks may be used as a system called functions. A feature is a sub-routine containing one or more statements to achieve a given purpose. We build functions for writing and assembling a C program. A C device can be used for one or more pieces as seen in Figure 5.

C PROGRAM STRUCTURE	EXAMPLE
DOCUMENTATION SECTION	(THIS IS USED FOR COMMENTS)
LINK SECTION #INCLUDE	(<STDIO.H>, #INCLUDE<CONIO.H>)
DEFINITION SECTION	((VOID FUNC:))
GLOBAL DECLARATION SECTION	(VARIABLE USED IN MORE THAN ONE FUNCTION) (EG: INT A= 10:)
MAIN FUNCTION SECTION	(MAIN())
DECLARATION PART	(PRINTF("A VALUE INSIDE MAIN():%D",A):)
EXECUTABLE PART	(FUNO):)
SUB PROGRAM SECTION	(VOID FUNCO) { (PRINTF("\n A VALUE INSIDE FUNO:%D",A): }
FUNCTION-1 FUNCTION-2 FUNCTION-3 (USER DEFINED FUNCTIONS)	

Figure 3.2. C Language Program Overview

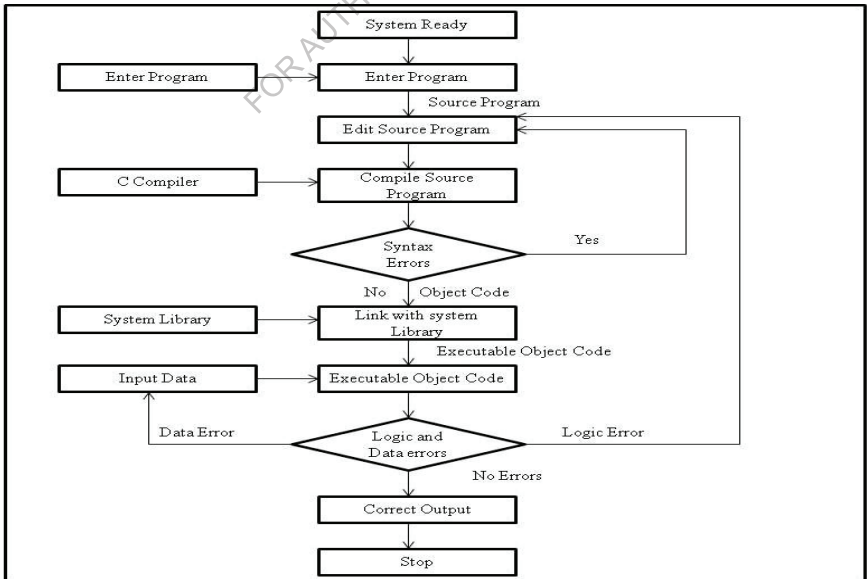


Figure 3.3. Development Life Cycle of a Program



### What errors we are getting in the compilation time:

- ❖ <stdio.h> No reference has been made to the header file, i.e.
- ❖ (No key) role is included
- ❖ Ignore the double limit of the statements
- ❖ The half-colon absents in the statements
- ❖ There No additional signs like an open brace and a closed brace are mentioned.
- ❖ There are a few errors you might make, as we have learned, also in pure javascript.
- ❖ It is recommended that we try to solve mistakes, try to compile them and study error messages.
- ❖ If you immediately correct the errors in the next applications, the errors are unchanged.
- ❖ We are now developing a C Language review software
- ❖ First, go to the File menu and click on the special opportunity to choose the new alternative.
- ❖ Write the software and register it as the file name and.c (C Executable Extension)

```

File Edit Search Run Compile Debug Project Options Window Help
[ ] TURBOC\PROJECTS\HELLO.C 6-1
#include <stdio.h>
#include <conio.h>
void main()
{
printf( 'Hello' );
getch();
}
7:57
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu

```

- ❖ Towards Next, assembled the software. Several compilers such as Sun compiler and Borland compiler are open. We press Alt+F9 for compilation in the TurboC setting.
- ❖ At this time the compiler will detect and display errors like typing errors, keyword errors, syntax use errors. The author will repair the glitches. Yet a programmer will remember that logical errors cannot be identified by the compiler.
- ❖ The compiled code will be saved to an object file after this process.
- ❖ Press Ctrl+F9 to link and execute an executable file for object code with the system library.
- ❖ Click Enter key to give you the appropriate feedback on the console pad. Press Alt+F5 to show the software results.

- ❖ If errors don't appear below the screen after compiling the program, they will appear. It's the display.



```
C:\TURBOC\BIN\TC
hello
```

### Second Program:



```
File Edit Search Run Compile Debug Project Options Window Help
N66.C
#include<stdio.h>
#include<conio.h>
main()
{
printf("Good morning");
printf(" Good afternoon");
printf(" Good evening");
printf(" Good night");
getch();
}

C:\TURBOC\BIN\TC
good morninggood afternoon good eveninggood night
```

Here we have a somewhat more complicated programme: one is Good Morning, the other Good Afternoon, the third Good Evening, the fourth Good Night, which is a little more difficult than the code that we saw. To recap # in the first line `<stdio.h>` the compiler will be given a file library for the standard input and output file. Then we've got the principal function. The open and close brackets immediately after the `main()` shows that `main()` is always the function running on the first row of the `main()` function, then the function body, and the function's logic, which is contained in two curly brackets, with the first curly bracket signal beginning of the function. This `main()` function contains four declarations. Each statement, as we said before, ends with a semi-colon, we have one statement only. Rough brackets with a set of statements and each statement performed at the exact location of the write code sequence.

### Program Comments:

Program Comments is also known if you code in addition to the statement you have executed and you can make a few further comments.

- ❖ Comments to the system are not included in the plan. The text between successive /\*and \*/ in the program we write is a program comment which the compiler would ignore.
- ❖ Comments from the program are written in a reasonable manner, to understand and to explain the programme.
- ❖ A variety of suggestions can be contained in the software notes. The statements can be short or lengthy.

Such software statements are useful to more detailed projects, as users can better grasp as maintain the devices. The comments on the program are a good idea for our schedule feedback.

More than one newline character definition program:

```

1  #include <iostream>
2  #include <conio.h>
3
4  using namespace std;
5
6  int main()
7  {
8      printf("good morning\n");
9      printf("good afternoon\n");
10     printf("good evening\n");
11     printf("good night\n");
12     getch();
13 }

```

Output:

```

good morning
good afternoon
good evening
good night

```

**Note: getch(); and clrscr():** These are the widespread functions:

**clrscr():**

The display panel is used for washing. Regardless of the output, the prior ones will be removed and the output will be void. Once your screen is clear, we only show the statements you write in the print declaration.

**getch ();**

Used to verify the program's final output. Therefore, no keyboard and programmer characters can linger until the period doesn't leave, so that the software won't stop. The software is terminated until you type some character from the keyboard.

**Program on Adding Two Numbers:**

```

1: #include <stdio.h>
2: #include <stdlib.h>
3: int main()
4: {
5:     int a,b,c;
6:     clrscr();
7:     printf("enter a value\n");
8:     scanf("%d",&a);
9:     printf("enter b value\n");
10:    scanf("%d",&b);
11:    c=a+b;
12:    printf("c is %d\n",c);
13:    getch();
14: }

```

```

enter a value
4
enter b value
5
c is 9

```

### UNIX System:

After installing the UNIX operating system into the memory the machine is able to accept the software. In a log you will join the software. The names of the file may contain a dot and a letter (c) and special letters and letters.

It is possible to create a file by editing text, ed or vi. Call the writer and build the script.

#### Ed filename

If the file exists previously, it is loaded. If this does not yet occur, the file must be generated to accept the new program. The publisher updates the program.

The script is placed on disk after the edit has stopped. You will test it at any point later using your user name. The application in the repository is regarded as the source script, because it contains the initial version of the system.

### MS-DOS System:

For any word processing tools, the application can be generated in non-document mode. The first.c, new.c and order are the end of the file term.

#### NAYAN.C

Under MS-DOS, the NAYAN.C file must load the software and generate object. Code. The file is called NAYAN.obj, in another file. The technology is preserved. The house is not done. If errors are identified in any language. The software has to be updated and re-compiled.

### Windows System:

In contrast to Linux the Windows Framework has no built-in C compiler. There are several IDEs.

An IDE is an important programming environment that allows programmers in a way that enables programmers to write, save, compile, edit, connect, and run programs. The IDEs must be installed

individually on a windows system. The best-known IDE is Borland C, Microsoft Visual Lab, Turbo C or C++, etc. These are the most popular examples. Consequently, a mouse will do much of the job. Some of these IDEs also work in the DOS setting. The DOS box or command prompt should be named by opening the IDE. The provided tools will help us to carry out the necessary compilation, connection and execution activities once in IDE.

**Main Thoughts:****Multiple Choice Questions:**

1. C Language Programming was Created by \_\_\_\_\_

- a) Dennis Ritchie      b) Ken Thomson      C) Bill Gates      d) None

Ans: A

2. C Language was developed in the year

- a) 1850      b) 1900      C) 1972      d) All

Ans: C

1) Dennis Ritchie Date of Birth

- a) 9-9-1951      b) 9-9-1958      C) 9-9-1947      d) 9-9-1941

Ans: D

2) Dennis Ritchie Full Name?

- a) Dennis Bill Ritchie      b) Dennis MacAlistair Ritchie      c) Dennis Nag Ritchie      d) None

Ans: B

3) C programming Language was created at

- a) London      b) Australia      c) AT&T Bell Lab      d) None

Ans: C

4) C Programming Language Committee Standardized Which Place

- a) ISO      b) ASCII      c) B      d) ANSI

Ans: D

5) C Programming Language is evolved from

- a) B      b) BCPL      c) C++      d) A and B

Ans: D

6) Printf is

- a) Output      b) Input      c) both A and B      d) None

Ans: A

7) Scanf is

- a) Output      b) Input      c) both A and B      d) None

Ans: B

- 8) JVM Full form  
c) Java Virtual Machine b) Java Virtual Mike c) Java Vest Machine d) None

Ans: A

**Short Question:**

- 1) Describe Memory Management
- 2) Define Printf() and Scanf()
- 3) Explain Program Comments
- 4) Describe about getch()?
- 5) clrscr()?

**Review Questions:**

- 1) Describe C Language History?
- 2) C Language Importance?
- 3) C language Features?
- 4) C language Programming structure?

**Exercises**

Write a name, date of birth and gender printing program

Write a number 1 to 100 printing program

Compose a Fahrenheit to Celsius conversion system

Write down the curriculum and determine whether a pupil has completed the test or not.

Write the 3 numbers summary program

Write a square area and perimeter search programme.

Write an interest-finding program

Write a timely vector exchange scheme Two amounts.

Write the two-number software larger

Specify a three-number program to find

Write down a program for finding even 1 to 100 numbers

Create a system between 1 and 100 to identify an affected amount

Create a program to find sequence total 1 to N 13

Write a multiplication table printing program of a number

Write the schedule in order to find the total and average of the numbers given.

Write a program to determine whether or not a number is prime

Write a two-number GCD and LCM software

Write a program to find the entire number divider.

## Unit-IV

### Constants, Variables and Data Types

#### Main Objective of the Unit:

It should be complete this unit:

- ❖ Learn the C character collection and keywords
- ❖ ADD Variables and constants Definition
- ❖ Define different C data forms
- ❖ Explain how vector is used by a system
- ❖ Contents Illustrate how a device hires constants

#### Introduction:

A programming language is intended for the encoding and knowledge delivery of some forms of data consisting of numbers, characters and sequences. A precision list named a system executes the role of data processing. Such guidelines are found in the laws of other symbols and terms, called syntax codes. The language syntax laws must be verified accurately for all system commands.

As every other word, C has its own vocabulary and grammar. Within this portion the definitions and forms of constants and variables related to the C programming language will be discussed.

#### Character Set:

While a series of characters used to render terms, sentences, etc. is found in any language, C often includes a collection of characters including alphabets, digits and special symbols. A minimum of 256 characters are supported in C language.

Every C system includes statements. These sentences are made with vocabulary and these phrases are formed with characters in C. The following set of characters is included in the C language collection.

In 'C' Programming Language set of characters is described into two ways:

- 1) Source Character Set
- 2) Execution Character Set

#### 1. Source Character Set:

- ❖ Alphabets
- ❖ Digits
- ❖ Special Characters
- ❖ White Spaces (Blank spaces)

A compiler also prohibits the usage of character but is commonly used for formatting results. The programming function 'C' has the following character:

**Alphabets:**

All English alphabets are supported by the language C. Together the bottom and top letters endorse 52 alphabets.

Lower case letters: a to z

Upper case letters: A to Z

**Digits:**

The 10 numbers that are used to build numerical values in C language are supported by C language.

**Digits 0 to 9****Special Symbols:**

C Language embraces a broad variety of special symbols including abstract symbols, white spaces, reverse spaces and other special symbols. C language embraces

Special Symbols - ~ @ # \$ % ^ & \* ( ) \_ - + = { } [ ] ; : ' " / ? . > , < \ | tab newline space NULL bell backspace vertical tab etc.,

**2. Execution Character Set:**

Some ASCII characters (American Standard Code for Information Interchange) show that the computer or printer is not shown. These characters do other than display the text. For example, removing, switching to a newline or ringing a glove.

They are part of the tests. An escape series is usually a backstroke and a letter or mixture of numbers. An escape sequence is usually named a single character but an real type continuity.

This is used throughout the service of the device. The moving characters sequence display even a backslash (\) with a thread. Note that a constant character represents one personality, while it consists of two characters. These character combinations are referred to as the escape chain.

Characters usually associated for C values in ASCII



Character	ASCII VALUE	Meaning
NULL	0	Null
SOH	1	Header Start
STX	2	Text Start
ETX	3	Text End
EOT	4	Transaction End
ENQ	5	Enquiry
ACK	6	Acknowledgement
BEL	7	Bell
BS	8	Backspace
HT	9	Horizontal Tab
LF	10	Line feed
VT	11	Vertical Tab
FF	12	Form Feed
CR	13	Carriage Return
SO	14	Shift Out
SI	15	Shift In
DLE	16	Data Link Escape
DC1	17	Device Control 1
DC2	18	Device Control 2
DC3	19	Device Control 3
DC4	20	Device Control 4
NAK	21	Negative Acknowledgement
SYN	22	Synchronous Idle
ETB	23	End of Trans Block
CAN	24	Cancel
EM	25	End of Medium
SUB	26	Substitute
ESC	27	Escape
FS	28	File Separator
GS	29	Group S Separator
RS	30	Record Separator
US	31	Unit Separator

Table 4.1. C Programming Language ASCII Value Character set

Printable Characters			
Character	ASCII VALUE	Character	ASCII VALUE
Space	32	!	33
!"	34	#\$	35
#\$%	36	%&	37
%&'	38	'(	40
'(	41	)*	42
)*+,-	43	-.0	44
-.0	45	12	46
12	47	34	48
34	49	56	50
56	51	7	52
7	53	8	54
8	55	9	56
9	57	:	58
:	59	<	60
<	61	=	62
=	63	?@	64
?@	65	A	66
A	67	B	68
B	69	C	70
C	71	D	72
D	73	E	74
E	75	F	76
F	77	G	78
G	79	H	80
H	81	I	82
I	83	J	84
J	85	K	86
K	87	L	88
L	89	M	90
M	91	N	92
N	93	O	94
O	95	DEL	127

Table 4.2. Printable Character set

Example Program on Character Set:

```

104=>#
105=>|
106=>|
107=>#
108=>|
109=>#
110=>#
111=>#
112=>#
113=>#
114=>#
115=>#
116=>#
117=>#
118=>#
119=>#
120=>#
121=>#
122=>#
123=>#
124=>#
125=>#
126=>#
127=>#

```

### Tokens in C Language:

Every C system is a series of instructions and each instruction consists of a set of other units. The token is named increasing smallest single unit of a c program. Every c-program instruction is a tokens array. Tokens are used to construct c programs and the basic building blocks of a c program are listed.

#### Keywords:

Although any language has words to build sentences, C programming has also words to create c system instructions with a particular purpose. Keywords are unique words with a particular definition in the vocabulary of programming C. Keywords in C programming language are also regarded as reserved words.

There are 32 keywords in the programming language C. These 32 keywords have a definition that the programmer already understands.

Keywords are the term that the programmer already recognizes with a predefined definition Whenever a compiler discovers a term, it knows its meaning immediately.\

#### Keyword Properties:

- ❖ All keywords are specified as small letters in C programming language, so they can only be used in small letters
- ❖ Every keyword has a particular definition, users can not modify it.
- ❖ Keywords cannot be used as attribute, function, list, pointing, etc ... as user-defined names.
- ❖ That keyword in C programming language is anything that the programmer may do or define some kind of behavior.

- ❖ The list below displays all 32 keywords with their meaning.

32 Keywords with their meaning		
S.No.	Name of the Keyword	Meaning of the Keyword
1	Auto	Used as an automatic class of storage
2	Break	Control declaration used to terminate statements of switch and loop
3	Case	Used for a case in the switch declaration
4	Char	Used to reflect the form of data
5	Const	Used to describe a permanent
6	Continue	Unconditional control statement used to test the looping statements at the top.
7	Default	Used in Switch statement to represent a default case
8	Do	Used to define do block in do-while statement
9	Double	Used to present double datatype
10	Else	Used to define false block of if statement
11	Enum	Used to define enumerated datatypes
12	Extern	Used to represent external storage class
13	Float	Used to represent floating point datatype
14	For	Used to define a looping statement
15	Goto	Used to represent unconditional control statement
16	If	Used to define a conditional control statement
17	Int	Used to represent integer datatype
18	Long	It is a type modifier that alters the basic datatype
19	Register	Used to represent register storage class
20	Return	Used to terminate a function execution
21	Short	It is a type modifier that alters the basic datatype
22	Signed	It is a type modifier that alters the basic datatype
23	sizeof	It is an operator that gives size of the memory of a variable
24	Static	Used to create static variables – constants
25	Struct	Used to create structures userdefined datatypes
26	Switch	Used to define switch case statements
27	Typedef	used to specify temporary name for the datatypes
28	Union	Used to create union for grouping different types under a name
29	Unsigned	It is a type modifier that alters the basic data type
30	Void	Used to indicate nothing return value, parameter of a function
31	Volatile	Used to creating volatile objects
32	While	Used to define a looping statements

Table 4.3. 32 keywords

**Keywords Example program:**

```

1 int main()
2 {
3     float x=10.5;
4     myfunc();
5 }
6 float myfunc()
7 {
8     printf("value of x is: %f\n", x);
9     return 0;
10 }

```

In the above program, Float and Return are keywords. The float is used to assign variables and the return is used to retrieve an integer style value in this method.

### Identifiers:

Programmers may define a vector, sequence, reference, method etc in the programming language C. Character recognition is a set that serves as a variable name, feature, sequence, reference, structure. In other terms, you may specify a user name such that a person may recognize the name of a vector name, feature name, array name, point name, form name or mark in the c language only. In other terms, it is the user specified name.

The identifier is an entity's user-defined name to describe during software distribution

### Identifiers Rules:

- ❖ An ID may comprise just a number and a symbol (UPPERCASE and lowercase).
- ❖ A numerical value should not begin with an identifier. The letter or an underscore may be used to continue.
- ❖ In between the marker and whitespace, we cannot use any different symbols. The only underscore sign is therefore allowed. When signatures, keywords are not to be included.
- ❖ The duration of an identifier is not constrained. The first 31 characters are only regarded by the programmer. In its sense, an identity must be special.

### Improved language Classification for Identifiers:

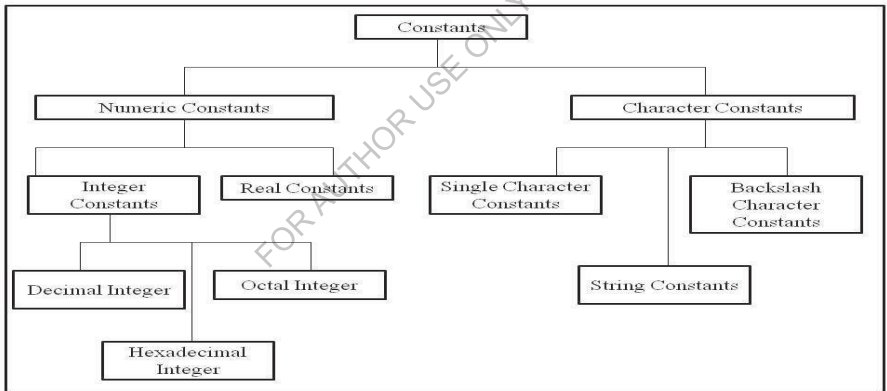
The foregoing are the standard guidelines for finding effective programming.

- ❖ To identify the object, the description must be significant.
- ❖ Because an underscore will cause conflicts with device, we do not launch an underline.
- ❖ With a lowercase letter, we continue any identification. In case a code includes more than one phrase, a small letter ends with the first phrase and the first letter ends with second term. We may also use an aster in an code to distinguish different terms.

**Operators:**

An operator is a symbol used in program operations for arithmetic and logic. That implies that an operator is a mark that asks the programmer to execute arithmetic or logic. C's programming language supports a large number of operators classified as the following.

1. Arithmetic Operator
2. Relational Operator
3. Logical Operator
4. Increment and Decrement Operator
5. Assignment Operator
6. Bitwise Operators
7. Conditional Operators
8. Special Operators

**Constants:**

**Figure 4.1.: C Constants**

Through C programming, the variable is identical to a constant, except during system run, the constant retains just one value. This implies that, if a constant value is given, it cannot be modified when running the program. The value is fixed throughout the program when it is assigned to the constant. This is how a constant can be described.

A constant is a named memory location that has only one value during running the program. In C, any data type such as integer, floating point, character, string and double can be a constant, etc.,

The constants are divided into two basic groups with subtypes each. The criteria are:

**Integer Constants:**

An integer constant may be a decimal integer or an octal integer. A value of the decimal integer is calculated to a straight integer number whereas the number for the octal integer is calculated to 'o' and the value of the hexadecimal is prefixed to 'OX.'

An integer constant can also be a non-signed type of integer constant. The unsigned integer value is suffixed with 'u' and 'l' is suffixed with the shorter integer constant value, whereas 'ul' is suffixed with the unsigned short integer constant value.

**Floating Point Constants:**

An integer and decimal component must be present in a floating point-constant. Often the exponent component can also be used. If exponents are defined as a floating-point constant, 'e' or 'E' will suffix the meaning.

**Character Constants:**

A word in a single quote is a character constant. A persistent character has a fixed line count. There are several pre-defined constants called escape sequences in the programming language C.

Each escape sequence has its own unique functionality and is prefixed with the \ symbol in each escape sequence. Such escape sequences are included in the 'printf()' output function.

**String Constants:**

A string constant is a set of letters, numbers, symbols and escape sequences found in repeat quotations.

**Variables:**

C programming language variables are the designated memory locations where a user during execution of the program can store different values of the same data type. This implies that a variable is a name assigned to a memory position where different values of the same data form may be kept. This ensures that during program execution a storage container may be described as a variable that contains the same datatype values. The structured data category description is the following:

Variable is a name given at a location where during program execution we can store different data values of the same data type.

Growing element shall be defined in the definition segment in c programming language before using it. The datatype that specifies the number and form of values and the memory capacity that will be assigned will be used for each element.

A name of a variable may contain a symbol, letters and digits. The guidelines for defining a vector name are as follows:

- ❖ Not a digit will continue variable name. Variable name.
- ❖ Any attribute names will include the keywords.
- ❖ No special symbols are needed except underscore ( ) for a variable name.
- ❖ A variable name may be any type, but only the first 31 characters of the variable name are considered by the compiler.

**Declaration of a Variable:**

The variable declaration informs the compiler to assign the memory needed by the variable name and requires data type values to be assigned only to the specified memory position. In the language of c programming, the declaration may be rendered either as global variables before the method or inside some block or feature. But at the start of the block or task it has to be.

**Declaration of Syntax:**

datatype variable Name;

**Example:**

Int digit;

**int** is a datatype here and Variable Name is **digit**

The above statement tells the compiler that assigns the name number to 2 bytes of memory and allows for integer values in that memory location only.

**Types of Variable:****1. Local Variable:**

A vector that has been defined to be a local function vector or element.

**2. Global Variable:**

A global variable is considered an external element or block variable. The global variable value may be modified by any function. Each functionality is available.

**3. Static Variable:**

The static variable is a persistent keyword attribute.

**4. Automatic Variable:**

The default is to immediately announce all the C variables in the row. Auto may specifically announce an automatic attribute.

**5. External Variable:**

An external variable may be used to exchange it in a variety of C sources Records, Files. To announce an external attribute, you will use specific keyword.

**Example program of variable using add two numbers**

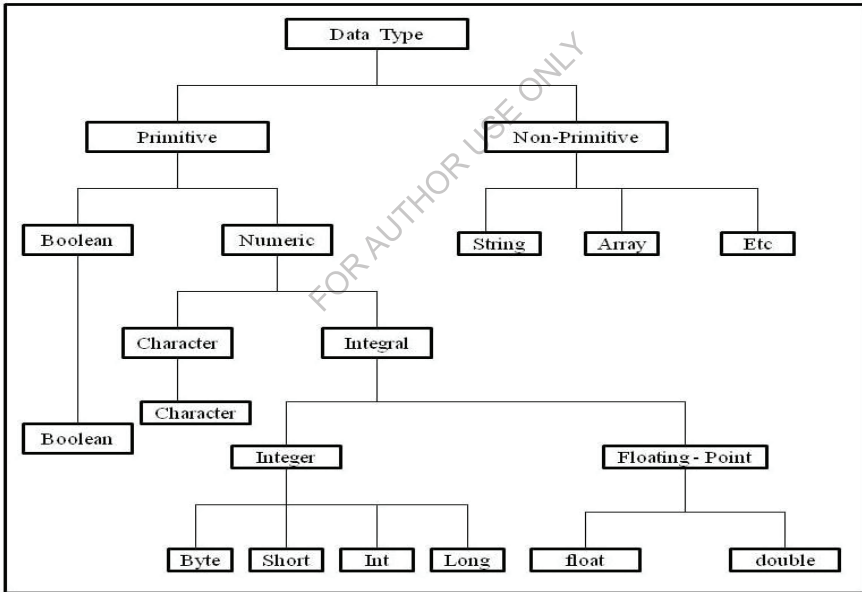


```

File Edit Search Run Compile Debug Project Options Window Help
[1] 19:20:19 19:20:19 sum of c is 30
#include<stdio.h>
#include<conio.h>
main()
{
    int a,b,c;
    clrscr();
    a=10;
    b=20;
    c=a+b;
    printf("sum of c is %d");
    getch();
    return 0;
}

```

**Data Types in C:**



**Figure 4.2. Data Types**

The data used in the c system is graded according to its properties in different forms. A data type can be specified as a set of values with similar properties in the C programming language. Both attributes have the same characteristics in a data form.

C Programming language data types are used to determine which type of value in a variable to be processed. The variable data form specifies the memory space and amount of value of an item. Throughout a c system, a variable or permanent or array will have a data type and that data type determines the sum of memory and the number of values to be retained in the variable or constant or list, which is to be allocated. The structured data category description is the following:

The data type is a value set with predetermined features. Component forms, variables, lists, points and functions are declared using data types.

Data types are classified as follows in the c programming language ..

- A) Basic Data Type or Primary Data Type
- B) User Defined Data Type or Derived Data Type
- C) Enumeration Data Types
- D) Void Data Types**

**a) Primary Data Type:**

The simple data forms are the primary data categories in the programming language C. The program also describes all key data forms. Called Built-in data forms, main data categories are often named. The key data forms of c programming language are as follows:

- 1) Integer Data Type
- 2) Floating Point Data Type
- 3) Double Data Type
- 4) Character Data Type

**1) Integer Data Type:**

The integer data form is a number package. No decimal representation is given for every numerical number. We use the "int" keyword to show the c type of integer data. We use the int keyword to define the variables and to determine the function return sort. The integer data form is used for numerous adjustment forms such as short, medium, signed and unsigned. The table below contains information on the integer data form.

Type	Size (Bytes)	Range	Specifier
Int (Signed Short Int)	2	-32768 to 32767	%d
Short Int (Signed Short Int)	2	-32768 to 32767	%d
Long Int (Signed Long Int)	4	-2,147,483,648 to 2,147,483,647	%d
Unsigned Int (Unsigned Short Int)	2	0 to 65535	%u
Unsigned Long Int	4	0 to 4,294,967,295	%u

**Table 4.4. Range of Integer Data Type**

## 2) Floating Point Data Type:

Data forms of floating-point are a collection of decimal value numbers. The decimal sum will be found in each floating-point number. There are two variations of the floating-point data form:

**float**

**double**

We use the "float" keyword to describe floating point data sort, and "double" in c. The float and twice the number of decimal places is identical, but distinct. The float meaning is 6 decimal places and the double number 15 to 19 decimal positions. The table below provides full clarity on the forms of floating points.

Type	Size (Bytes)	Range	Specifier
Float	4	1.2E-38 to 3.4E+38	%f
Double	8	2.3E-308 to 1.7E+308	%lf
Long Double	10	3.4E-4932 to 1.1E+4932	%ld

**Table 4.5. Range of Floating-Point Data Type**

## Character Data Type:

The data definition of character is a collection of characters found in individual quotes. The table below offers data on the type of character material. The following table provides full detail on all forms of data in c language of programming.

Type	Size (Bytes)	Range	Specifier
Char (Signed Char)	1	-128 to 127	%c
Unsigned Char	1	0 to 255	%c

**Table 4.6. Range of Character Data Type**

### Void Data Type:

None or no meaning is the void data form. The void is usually used to denote a function that provides no meaning. The void data form is often used for describing null feature parameters.

### Enumerated Data Type:

A data type mentioned is a user-defined data form composed of integer constants and a name is assigned to each integer constant. To define the data type listed, the keyword "Enum" is used.

### Derived Data Types:

Data types extracted are data categories specified by the consumer. The data forms obtained are often referred to as user-defined or secondary data types. The derived data types are provided by the following principles in the c programming language

- ❖ Arrays
- ❖ Structures
- ❖ Unions
- ❖ Enumeration

### Variable declaration

After the developer has built it we will declare the right variable names. There are two things in the argument:

Termination The variable name is told to the programmer.

- ❖ The data form generated by the vector
- ❖ At the start of the specification block, variables must be declared before they are included in a system.

### Primary Type Declaration

Any form of data can be saved using a variable. The name has little to do with its existence, in other words. The definition of vector clauses is the following:

**Datatype num1,num2.....numn;**

Num1,num2 .... numn are descriptions of variables. The variables are grouped into commas.

A sentence begins with a half-column. The correct declarations are for example:

**Int Count;**

**Int number, total;**

**Double ratio;**

The keywords reflect the integer form and current data meaning are Int and Double.

Data Type	Keyword relation
Character	Char
Unsigned Character	Unsigned Char
Signed Character	Int (or) Signed int
Signed Short Integer	Short or Short Int or Signed Short Int
Signed Long Integer	Long or Long Int or Signed Long Int
Unsigned Integer	Unsigned or Unsigned Int
Unsigned Short Integer	Unsigned Short or Unsigned Short Int
Unsigned Long Integer	Unsigned Long or Unsigned Long Int
Floating Point	Float
Double precision floating point	Double
Extended double precision floating point	Long Double

Table 4.7. Declaration of Variable

### Variable Declaration Program

```

1 | #include<string>
2 | #include<conio.h>
3 | //variable declaration//
4 | extern int a,b;
5 | extern int c;
6 | extern float f;
7 | int main() {
8 |     //variable definitions//
9 |     int a,b;
10 |     int c;
11 |     float f;
12 |     clrscr();
13 |     //actual initializations//
14 |     a=10;
15 |     b=20;
16 |     c=a+b;
17 |     printf("sum of a+b=%d",c);
18 |     getch();
19 |     printf("sum of a+b=%d",f);
20 |     getch();
21 |     return 0; }

```

Output window:  
value of c:30  
value of f:0.000000

### Storage Class in C:

Data groups are used to define the variable / function characteristics. By theory, such attributes provide the distance, exposure and duration of a particular attribute that allows us to monitor the involvement during system execution.

- ❖ Auto
- ❖ Extern
- ❖ Register
- ❖ Storage Class

**Auto:**

It is the default class of all specified variables inside a feature or a row. Therefore, the auto keyword is also used in C language while writing programs. Only in the block / function declared and not outside (defining its scope) can auto-variables be accessed. Of course, these can be accessed through the blocks in which the auto variable was declared within the parent block / function. However, the definitions presented here suggest the very precise memory location in which the variables exist. They may be reached beyond their reach as well. If declared, a waste attribute is allocated by design.

**Extern:**

Rather than the same block where it is used, the externals storage class simply informs us that the variable is elsewhere specified. Essentially, the meaning of a particular block is allocated to it, and may then be transcribed / changed into another row. So an external variable is none more than a regional variable with a valid meaning modified to be used somewhere. It is available in any function / block. A normal global variable can also be created externally by placing the "outer" keyword in any function / block before it is declared / defined. This simply implies that we don't launch a new variable but instead use / access the global variable. The primary benefit of utilizing foreign variables is that two separate files that are part of a wide system may be reached.

**Static:**

This storage class is used to define static variables widely found in c-language programs. Even after they are out of their scope, the static variables preserve their value. Static variables therefore retain in their sense the meaning of their last application. We would then assume that they are only configured once and remain before the completion of the project. A fresh recollection is then allocated as it is not retrieved. Their reach is local to their given feature. In every section of the system, global static variables may be manipulated. The value 0 is assigned to the compiler by default.

**Register:**

This storage class defines register variables with the same function as the auto variables. The main distinction is that if a free register is given, the programmer wants to store these variables in the microprocessor register. It renders register variables even easier to use than those placed in the memory during the program's runtime. If there is no free register, they are stored only in the memory. In the register keyword, typically just a few variables that have to be checked very frequently in a system, increasing the program's runtime.

An important and curious observation is that the address of a registry variable cannot be obtained via pointers.

**Storage Class Declaration:**

Variables in C do not only have the types of data, but also the class of storage which indicates where they are stored and the device component in which variables are recognized is defined by the storage class.

**Symbolic Constants Definition:**

If for any reason in a system the value of a constant is changed in many points, every assumption where it occurs in the software would modify, rendering it unalterable.

The abstract constant is the following:

```
# define symbolic name value
```

**Example:****Define PI 3.1415**

They then consider it impossible to use such systems. This is it.

1. Problem with changes in software
2. Computer comprehension issue.

**Declaring a Variable as Constant:**

During system execution.c, the values of these variables should be kept unchanged. The vector with the const parameter may be defined when initialized. For examples,

```
Const int a=10
```

Const is a modern data type descriptor for the ANSI format. The programmer will inform the software that the variable int a doesn't alter its meaning. However, it can be used on the right side of a statement like any other variable.

**Declaring a Variable as Volatile:**

The volatile parameter is used when we designate a value. The programmer should be told that the value will shift often. There are other dynamic features. The unreliable keyword cannot erase the memory assignment. There is no index for the variables in the file. The interest cannot shift in order of assignment. See if you can use the erratic acronym.

```
Volatile int x;
```

```
Int volatile x;
```

All of such arguments are here correct. Unlike most data types, we will use vector variables, structures, correlations, etc. The dynamic institutions and labor unions themselves will become unpredictable and their leader's dynamic.

The volatile is used in various places. Any global variables that can be accessed through other functions or interrupting services or the unpredictable may be used in a few multi-threaded applications for peripheral memory maps.

**Main thoughts:****Multiple Choice Questions:**

1. Define Token

- a) The Biggest unit      b) The smallest Unit      C) The Medium Unit      d) None

Ans: B

2. C Variable Cannot begin with

- a) Letter      b) Digit      C) Special Symbol      d) B & C

Ans: D

3. All modules of a system may control variables named Variables

- a) Internal      b) External      C) Local      d) Global

Ans: D

4. Enum types are processed by

- a) Processor      b) Preprocessor      c) Compiler      d) None

Ans: C

5. How many types of C Constants?

- a) Two      b) Three      c) Four      d) None

Ans: A

6. Enum Constants not a

- a) Primary Constant      b) Secondary      c) Dynamic      d) None

Ans: A

7. Character Constants not a

- a) Primary Constant      b) Secondary      c) Dynamic      d) None

Ans: B

8. In C Language how many keywords are present

- a) 10      b) 12      c) 100      d) 32

Ans: D

9. Types of Integers are

- a) Int      b) Long      c) Short      d) All of the Above

Ans: D

10. Which one is C keyword?

- a) Breaker      b) Default      C) Shorter      d) go to



Ans: B

**Short Questions:**

- 1) Describe Token
- 2) Explanation about Variable
- 3) Define Constant
- 4) Describe Operator

**Review Questions:**

- 9) Explain about Character Set?
- 10) Describe Data Types?

**Programing Exercises:**

1. Write a program on Check whether a Character is a Alphabet or Not?
2. Write a program to Print ASCII Value
3. Write a program on Local variables in C
4. Write a program on Global variables in C
5. Write a program of variables using add Two Numbers
6. Write a program of Variable Declaration

FOR AUTHOR USE ONLY

## Unit - V

### Operators and Expressions

#### Main goals of the Unit:

It should be complete this Unit:

- ❖ Know the different operators built in C programming language.
- ❖ Classify operators bitwise and unique
- ❖ Decide how to test mathematical expressions
- ❖ Describe forms of conversions in terms
- ❖ Explore how to apply operator precedence and laws of associativity

#### Introduction:

C embraces a large range of remote operators. We already used many of them, for instance =, +, -, & and <. An operator is a symbol that allows you to use your machine to make some mathematical or logical manipulations. In programmes operators are used for manipulating data and variables. They're usually part of the logical or math expressions.

Classification of C operators can be rendered in a variety of groups. Includes:

An operator is a function used in a programme to execute both mathematical and logical operations. That means an operator is a special symbol which tells the compiler to do math or logical operations. C programming language offers resources for a rich collection of operators defined as follows.

1. Arithmetic Operators
2. Relational Operators
3. Logical Operators
4. Increment and Decrement Operators
5. Assignment Operators
6. Bitwise Operators
7. Conditional Operators
8. Special Operators

#### 1. Arithmetic Operators:

The arithmetic operators are the symbols used to perform basic mathematical operations such as modulo addition, subtraction, multiplication, division, and percentage. The table below provides information on the arithmetic operators.

With numerical data types and character data type the addition operator can be used. It performs mathematical addition when used with numerical values and when used with values of the type of character data, it performs concatenation (appending).

The rest of the division operator is only used with data type integer.

No.	Operator Symbol	Description	Sample
1	+	Unary Plus or Addition	4+6=10
2	-	Unary Minus or Subtraction	4-1=3
3	*	Multiplication	6*6=36
4	/	Division	24/6=4
5	%	Remainder of the Division	9%4=1

Any element is truncated by the Integer division. The modulo division operations produce the remainder of an integer. Examples of Arithmetic Operator operation are:

**c-d    c+d**

**c\*d    c/d**

**c%d    -c\*d**

Here the c and d variables are classified as operands. The percentage of the modulo division operator cannot be used to produce floating point results. Please note C does not have an exponentiation feature. Older C models aren't unary plus but ANSI C allows them.

#### Arithmetic Operator Program Example:

```

#include<stdio.h>
#include<conio.h>
main()
{
    int a=10, b=20, c;
    clrscr();
    c=a+b;
    printf("value of a+b=%d\n",c);
    c=a-b;
    printf("value of a-b=%d\n",c);
    c=a*b;
    printf("value of a*b=%d\n",c);
    c=a/b;
    printf("value of a/b=%d\n",c);
    c=a%b;
    printf("remainder when a divided by b = %d\n",c);
    getch();
    return 0;
}

```

value of a+b=39  
value of a-b=18  
value of a\*b=390  
value of a/b=9  
remainder when a divided by b = 18

## 2. Relational Operators:

The relational operators are the terms used for the contrast of two quantities. Which implies that the relationship between two values is verified using the relational operators. Either

Relational Operator has two Truth or FALSE outcomes. The relational operators are used, in plain words, to describe requirements within a method. The table below contains details on link operators.

No.	Operator Symbol	Example	Sample	Description
1	< Less than	30<20	False	Returns Valid if the first value is below second, so returns FALSE
2	> Greater than	30>20	True	Returns TRUE where FALSE or otherwise returned if the first value is greater than the second
3	<= Less than or equal to	30<=20	False	Returns Valid because either the first value is smaller than the second value, or equivalent to it.
4	>= Greater than or equal to	30>=20	True	Returns Real where the first value is equal to or greater than the otherwise meaning Incorrect returns
5	== Equal to	30==20	False	If all values are the same returns TRUE if FALSE is returned otherwise.
6	!= Not equal to	30!=20	True	If the two meanings are not the same than TRUE FALSE returns.

#### Relational Operator Program Example:

```

1  file Edit Search Run Compile Debug Project Options Window Help
2  (-) RELATION.C (-)
3  #include<stdio.h>
4  #include<conio.h>
5  main()
6  {
7  int a=0;
8  int b=10;
9  clrscr();
10 printf("a: %d\n",a);
11 printf("b: %d\n",b);
12 printf("a<b: %d\n",a<b);
13 printf("a>b: %d\n",a>b);
14 printf("a<=b: %d\n",a<=b);
15 printf("a>=b: %d\n",a>=b);
16 printf("a!=b: %d\n",a!=b);
17 getch();
18 return 0;
19 }
20
21 a<b: 0
22 a>b: 1
23 a<=b: 0
24 a>=b: 1
25 a!=b: 0
26 a!=b: 1
27
28 13:26
29 F1 Help F11-F12 Next Msg F11-F12 Prev Msg F11-F12 Compile F9 Make F10 Menu

```

### 3. Logical Operators:

The symbols are the logical operators used to merge various conditions into one state. The table below contains details on logical operators.

No.	Operator Symbol	Example	Sample	Description
1	&& Logical AND	30<20 && 42>30	False	If all conditions are true, else the Correct returns Incorrect
2	 Logical OR	30<20    42>30	True	If all requirements are fulfilled True, or returns Fake Positive
3	! Logical NOT	!(30<20 && 42>30)	True	Returns Valid for a FLASE condition, and Inaccurate for an Inaccurate one.

#### Logical Operator Program Example:

```

File Edit Search Run Compile Debug Project Options Window Help
LOGICAL.C 1-1
#include<stdio.h>
#include<conio.h>
main()
{
int a=10, b=10, c=20, d;
clrscr();
!(a==b)&&(c==d));
printf("!(a==b)&&(c==d) is %d",d);
!(a==b)&&(c==d));
printf("!(a==b)&&(c==d) is %d",d);
!(a==b)||!(c==d));
printf("!(a==b)||!(c==d) is %d",d);
!(a==b)||!(c==d));
printf("!(a==b)||!(c==d) is %d",d);
!(a==b);
printf("!(a==b) is %d",d);
getch();
return 0;
}
1-1
FILE Help All-F9 Next Pg All-F7 Prev Pg All-F9 Compile F9 Make F10 Menu

(a==b)&&(c==d) is 1
(a==b)&&(c==d) is 0
(a==b)||!(c==d) is 1
(a==b)||!(c==d) is 0
!(a==b) is 1
!(a==b) is 0
-

```

### 4. Increment and Decrement Operator:

The operators of increment and decrement are called unary operators, because they both need one operand only. The increment operators apply one to the operand's actual value, and the decrement operator subtracts one from the operand's existing value. The table below offers details on operators with increments and decrements.

No.	Operator Symbol	Example	Sample	Description
1	++ Increment	Int x=6 x++;	X=7	Adds one to the Current value

2	-- Decrement	Int x=6 x--;	X=5	Removes one from existing value
---	-----------------	-----------------	-----	---------------------------------

The operators for increment and decrement are used in front of the operand ( $++x$ ) or after the operand ( $x++$ ). If used in front of the operand, we call it pre-increment or pre-decrement and if used after the operand, we call it post-increment or post decrement.

### Pre Increment and Decrement Operator:

In the case of pre-increment, before evaluation of expression, the value of the variable is increased by one. In the case of pre-decrement, before evaluation of expression, the value of the variable is decreased by one. That means that when we use pre-increment or pre-decrement, the value of the variable is first increased or decremented by one, then the modified value is used in the evaluation of the expression.

### Pre Increment and Decrement Operator Program Example:

```

1 i=11
2 j=11
3
4 #include <iostream>
5 #include <conio.h>
6 using namespace std;
7 int main()
8 {
9     int i=10, j;
10    clrscr();
11    cout<<endl<<endl;
12    cout<< " i=" << i << ", j=" << j << endl;
13    i++;
14    j--;
15    cout<<endl<<endl;
16    cout<< " i=" << i << ", j=" << j << endl;
17    getch();
18 }

```

### Post Increment and Decrement Operator:

In the case of post-increment, after measurement of expression the value of the element is raised by one. In the case of post-decrement, after evaluation of the expression, the value of the element is reduced by one. Which implies which when we use post-increment or post-decrement, the term is first calculated with the current meaning, and then the variable value is increased or decremented by one.

### Post Increment and Decrement Operator Program Example:

```

1: #include <stdio.h>
2: #include <conio.h>
3: void main()
4: {
5:     int i=12,j;
6:     i=i+5;
7:     j=i++;
8:     printf(" %d\n",i,j);
9:     getch();
10:    return 0;
11: }

```

Output: i=17, j=12

### 5. Assignment Operators:

The assignment operators are used to assign right side value (R-value) to the left side (L-value) vector. The task operator is used along with the arithmetic operators in numerous forms. The assignment operators in the C programming language are defined in the following table.

No.	Operator symbol	Example	Sample	Description
1	= Simple Assignment Operator	X=30 Y=20 Z=X+Y	30	Allocates operand to the correct Were for left-hand operand
2	+= Addition Assignment	X+=5 X=X+5	35	It connects the correct handle to the wrong handle, and assigns the left handle to the production.
3	-= Subtraction Assignment	X-=Y X=X-Y	10	This divides the correct handle from the left handle, and the production is transferred to the wrong handle.
4	*= Multiplication Assignment	X*=Y X=X*Y	600	The left operand multiplies the right operand and the left operand is provided with the result.
5	/= Division Assignment	X/=Y X=X/Y	1	The left handle is divided by the right handle, and the left handle is supplied with the commodity.
6	%= Remainder Assignment	X%=Y X%Y	1	It requires and assigns two modules Left operand in.

### Assignment Operators Program Example:

```

1  #include <stdio.h>
2  #include <conio.h>
3  main() {
4      int a; a=20; a+=5;
5      clrscr();
6      printf("Answer of 2+3 is:");
7      a=20;
8      a*=5;
9      printf("Answer of 2*5 is:");
10     a=20;
11     a/=5;
12     printf("Answer of 2/5 is:");
13     a=20;
14     a%=4;
15     printf("Answer of 2%5 is:");
16     clrscr();
17     return 0;
18 }

```

Output:

```

answer of 2+3
answer of 2+5
answer of 2*5
answer of 2/5
answer of 2%5
-

```

### 6. Bit-Wise Operator:

Bit operators are used in the C programming language in order to carry out bit level operations. The operations are conducted on the basis of binary meaning while we use the bit operators. The following table defines all C-language bitwise operators.

Two of the factors A and B can be regarded as  $X = 25$ , and  $Y = 20$ , as  $X = 25$ .

No.	Operator Symbol	Example	Sample	Description
1	& Bitwise AND	$(X \& Y) = 12$	00001100	AND performance is 1 when the bits of two operands equal 1. When an operand bit is 0, the corresponding bit values are evaluated at 0.
2	 Bitwise OR	$(X   Y) = 61$	00111101	The OR bit performance is 1 if one of the two operands at least is 1. In programming C, the OR operator is called.
3	^ Bitwise XOR	$(X \wedge Y) = 49$	00110001	If two operands have the necessary bits opposite, the bitwise XOR values would be 1. It has ^ marks.
4	~ Bitwise Complement	$(\sim X) = \sim(60)$	11000011	The replacement operator is a single operator (works on only one machine).



				It is shifted between 1 and 0 and between 0 and 1. It's the ~ label.
5	<< Shift Left	X<<2=240	11110000	A number of specified bits are moving to the left of the left shift operator. The fingerprint is <<.
6	>> Shift Right	X>>2=15	00001111	The correct shift operator moves all bits to the right by a certain amount of bits. It has >> marked.

### Bitwise Operator program example:

```

#include<stdio.h>
#include<conio.h>
main()
{
int a=6, b=14;
clrscr();
printf("The output of the bitwise AND operator a&b is %d\n", a&b);
printf("The output of the bitwise OR operator a|b is %d\n", a|b);
printf("The output of the bitwise EXCLUSIVE OR operator a^b is %d\n", a^b);
printf("The output of the bitwise COMPLEMENT operator ~a is %d\n", ~a);
printf("The value of a(2) is: %d\n", a(2));
printf("The value of a(2) is: %d\n", a(2));
getch();
return 0;
}

```

the output of the bitwise AND operator a&b is 6  
The output of the bitwise OR operator a|b is 14  
The output of the bitwise EXCLUSIVE OR operator a^b is 8  
The output of the bitwise COMPLEMENT operator ~a is -7  
The value of a(2) is: 24  
The value of a(2) is 1

## 7. Conditional Operator:

Also known as a ternary operator, the conditional operator needs three operators. It's a decision-making operator. We check a condition in this operator first, and then we carry out an operation based on the condition result out of both operations. If TRUE is the first function, the second option will be performed if the requirements are FALSE. With the following syntax, the conditional operator is used.

Condition? True Part: False Part;

Example

A = (10<15)? 100:200; ⇒ A value is 100

Special Operators (size of, pointer, comma, dot, etc.)

### Conditional Operator program example:

```

#include <stdio.h>
#include <conio.h>
main()
{
    int age;
    clrscr();
    printf("enter your age here\n");
    scanf("%d",&age);
    (age)>=18 ? printf("you are eligible to get driving licence\n"):
    printf("you are not eligible to get driving licence\n");
    getch();
    return 0;
}

```

enter your age here:  
18  
you are eligible to get driving licence\_

### 8. Special Operator:

The following are the c programming language special operators.

- ❖ Sizeof Operator
- ❖ Pointer Operator
- ❖ Comma Operator
- ❖ Dot Operator

#### Sizeof Operator:

The sizeof () operator is the regular C function. It is a single compiler operator and its scale is determined. Returns the vector scale. It is suitable for all forms of data, floats and variable points.

The sum of memory reserved for this form of data is easily returned if the sizeof() method is used for data forms. The output can vary on various processors, including 32-bit systems, although different data types can be seen on a 64-bit machine. In C, this is a case in point.

#### Sizeof() Operator Program Example:

```

File Edit Search Run Compile Debug Project Options Window Help
SIZEOF.C 4-11
main()
{
  int a=88;
  clrscr();
  printf("The size of variable 'a' is %d", sizeof(a));
  getch();
  return 0;
}
8:1
F1 Help Alt-F0 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu
The size of variable a:2

```

### Pointer Operator:

It is used in c programming language to define pointer variables.

No.	Operator Symbol	Description
1	& Address Operator	Gives address of a variable
2	* Value at Operator	Stored single address interest

To create a pointer, we use the variable '\*' operator and to locate the variable address using the '&' user.

### Pointer Operator Program Example:

```

File Edit Search Run Compile Debug Project Options Window Help
POINTER.C 1-11
#include<stdio.h>
#include<conio.h>
main()
{
  int a=88;
  clrscr();
  printf("The address of a is %p", &a);
  printf("The value of a is %d", a);
  printf("The value of a is %d", *( &a ));
  getch();
  return 0;
}
9:37
F1 Help F2 Save F3 Open Alt-F9 Compile F9 Make F10 Menu
The address of a is 65524
The value of a is 88
The value of a is 88

```

### Comma Operator:

This operator uses variables to be isolated during declaration, function call expressions separate, etc.

#### Comma Operator Program Example:

```

1:1
#include <stdio.h>
#include <conio.h>
int main()
{
    int a=1,b=4;
    int c;
    clrscr();
    c=a,b;
    printf("value of c is %d",c);
    getch();
    return 0;
}
value of c is 4

```

### Dot Operator:

A dot operator is used in a struct if we want to reach that information. For e.g., the above code implements a number of entities. Generally, the point operator is mistaken for the arrow operator. And when the attribute is a reference will the arrow operator be utilised.

### Arithmetic Expression:

An arithmetic expression is a mixture of variables, constants and operators organised according to the language syntax. We have taken a selection of generic words in the instances we have described to date. C will accommodate most complicated mathematical expressions. Below are some C expressions instances. Please notice C does not have a user reveal.

No.	Algebraic Expression	C Expression
1	$axb-c$ $(p+q)(r+s)$	$a*b-c$ $(p+q)*(r+s)$
2	$(\frac{ab}{c})$	$a*b/c$
3	$6a^2+2a+4$	$6*a*a+2*a+4$
4	$(\frac{p}{q})+r$	$p/q+r$

### Evaluation of Expression:

In programming language C, a concept focused on the operator's precedent and relations are evaluated. When a multiple operator is active, the operator's priority and relation is evaluated. The operator at the highest rate is first assessed and the operator at the lowest rank is last assessed.

The calculation of an expression is based on the context and the organisation of operators. To understand expression evaluation in c, let us find the following basic sample phrase.

$$6+4*8/3$$

This expression includes three +, \* and / operators. The two operators have the same higher incidence of multiplication and division and lower precedence. Therefore, all multiplication and division are first evaluated and then the addition is calculated according to the order of the operator. As multiplication and division have the same precedence, they are analysed on the basis of associativity. In this case, the associativeness of multiplication and division is gone. But the separation is completed first and then the extension. Therefore \* / and + for the preceding expression are evaluated.

#### **Precedence of Arithmetic Operators:**

The mathematical expression from left to right is calculated using the operator's precedence law.

#### **Some Computational Problems:**

If expressions contain actual values, safeguards are needed to safeguard them against such programming errors. We know that the computer offers approximate values in real numbers and that this error will lead to serious problems.

#### **Type Conversions in Expressions:**

When multiple types of variables and constants are mixed in a sentence, the same form of data is interpreted. The conversion method is called a predefined shape of conversion. The conversion of type C may be separated into two distinct types:

#### **Implicit Type Conversion:**

Form conversion is the process by which the programmer converts the data type directly from one data form to the next. The conversion form is often called a transfer form. The programmer would render the tacit conversion automatically.

For starters, if we allocate an integer value to a float variable, the integer value is transformed by adding a decimal value, 0. When the float value is assigned to an integer, it immediately transforms the float value into an integer by removing the decimal value. For further information on conversion, see the following example

**Implicit Expression Example program:**

```

1 | #include <stdio.h>
2 | #include <conio.h>
3 | int main()
4 | {
5 |     int x=10;
6 |     float y=12.99;
7 |     char ch='A';
8 |     clrscr();
9 |     printf("%d",y);
10 |    printf("The value of x is %d",x);
11 |    printf("%f",y);
12 |    printf("The value of y is %f",y);
13 |    printf("The value of x is %d",x);
14 |    getch();
15 |    return 0;
16 | }

```

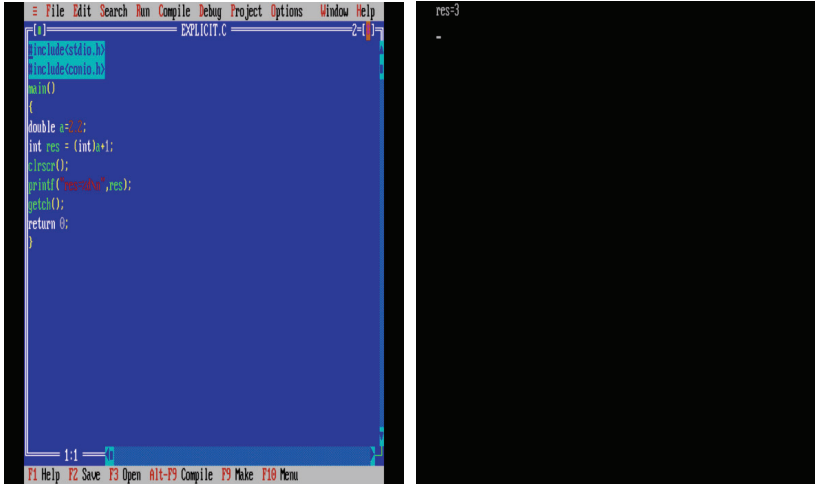
the value of x is 12  
the value of y is 12.000000  
the value of x is 98

In the above method, we add a vector value of  $x = y$  to the integer element, which is here the compiler can convert the float value (12.99) to integer (10), then the iterator is assigned to variable x, by extracting the decimal portion of the float value (12.99). This (10) value is represented by  $x = y$  as the decimal part of a float number (12,000000). This number is modified in the same manner.

**Explicit Type Conversion:**

Often known as Typecasting is the specific conversion process. Implicitly, compilers convert data from one application type into another. Data loss will occur if the programmer translates indirectly. In these situations, data is transformed from one data form into another through clear translation. To do this, we use the peculiar cast device. For data transfer from one type to another, we identify in parenthesis the target data type as a prefix to the converting data attribute.

**Explicit Expression Example program:**

The image shows two side-by-side screenshots from a C programming environment. The left screenshot displays a code editor with the following C code:

```
#include <stdio.h>
#include <conio.h>
main()
{
    double a=5.0;
    int res = (int)a*4;
    clrscr();
    printf("res=%d",res);
    getch();
    return 0;
}
```

The code is highlighted in blue. The right screenshot shows the output of the program, which is a black terminal window with the text "res=3" displayed in white.

### Operator Precedence and Associativity:

The priority of the operator is used to define the sequence of operators in an expression analysed. Every operator has precedence (priority) in c programming language. If there are more than one operator in an expression, the higher-precedence operator is first evaluated and the least-prompted operator is last evaluated.

The associativity of the operator is used to evaluate the same prioritisation order of operators in a word. In c programming we use association in order to decide the order of assessment of such operators, where an expression includes several operators of equivalent precedence. The operator precedence and associativity in c programming language are as seen in the table below.

S.No.	Operator and Description	Associativity
1	() Parenthesis [] Brackets . Member selection via Object Name > Member selection via pointer ++ -- Postfix Increment/Decrement	Left-to-Right
2	++ -- Prefix Increment/Decrement + - Unary plus/minus ! ~ Logical Negation/Bitwise Complement (type) Cast (Change Type) * Dereference & Address sizeof() Determine size in bytes	Right-to-Left
3	* Multiplication / Division % Modulus	Left-to-Right
4	+ Addition - Subtraction	Left-to-Right
5	<< Bitwise shift Left >> Bitwise shift Right	Left-to-Right
6	< Less than <= Less than or equal to > Greater than >= Greater than or equal to	Left-to-Right
7	== Equal to != Not Equal to	Left-to-Right
8	& Bitwise AND	Left-to-Right
9	^ Bitwise Exclusive OR	Left-to-Right
10	Bitwise Inclusive OR	Left-to-Right
11	&& Logical AND	Left-to-Right
12	Logical OR	Left-to-Right
13	?: Ternary Conditional	Right-to-Left
14	= Assignment	Right-to-Left

### Mathematical Functions:

Mathematical functions such as cos, sqrt, log, etc. are often used in the study of real-life problems. Most C compilers have these simple math functions. But programmes with a



greater math library are accessible and the reference manual can be reviewed to figure out which functions are accessible.

S.No.	Function	Description
1	floor ()	The nearest integer is given by this function, which is less than or equal to the given argument.
2	round ()	The next integer value of the float/ double-argument passed to this feature is returned with this feature. If the decimal value is ".1 to 0.5," the integer value returns less than the argument. When the decimal is between ".6 and .9," the integer value is larger than the argument.
3	ceil ()	The nearest integer value returns that is greater than or equivalent to the argument that is transferred to this function.
4	sin ()	The function is used for sine value calculation.
5	cos ()	This feature is used for cosine calculation.
6	cosh ()	This function is used for hyperbolic cosine calculation.
7	exp ()	The exponential 'E' to the xth power is determined using this function.
8	tan ()	The tangent is calculated using this function.
9	tanh ()	The hyperbolic tangent is calculated using this function.
10	sinh ()	This function is used for hyperbolic sinis calculations.
11	log ()	This function is used for natural logarithm calculation.
12	log10 ()	The base 10 logarithm is calculated with this feature.
13	sqrt ()	This feature is used to find the root of the argument that has been passed on to this feature.
14	pow ()	The power of the given number is used to find this.
15	tunc.()	This option truncates from floating point value the decimal value and returns integer value.

### Multiple Choice Questions (MCQ)

- That operator only takes integer operations of the following  
B) %    B) +    C) \*    D) None

Ans: A

- The user with the least responsibility  
B) ||    B) %    C) +    D) ++

Ans: A

- The C language is called operator percentage  
B) Division    B) Modulus    C) Percentage    D) None

Ans: B

4. What is accomplished fast?  
B) ++a    B) a++ C) both    D) None

Ans: C

5. What is an incomplete assignment operator of the following?  
B) X%=20    B) x/=20    C) x|=20    D) None

Ans: D

**Fill in the Blanks:**

1. The size of signed integer is \_\_\_\_\_ bytes.
2. There are \_\_\_\_\_ keywords in 'C' language.
3. In \_\_\_\_\_ operation, second expression is evaluated if first expression is false.
4. The operator  $i \ll 2$  is equivalent to \_\_\_\_\_  $i$  by 4.
5. The \_\_\_\_\_ operator is sometimes also called ternary operators.

**Short Questions:**

1. What is an Operator
2. Define Implicit Expression
3. Describe Math Function
4. Explain about Operator Precedence

**Review Questions:**

1. Explain different Types of Operators?
2. Explain Explicit Expression?

**Programming Exercises:**

1. Write a program on Arithmetic Operators
2. Write a program on Logical Operators
3. Write a program on Relational Operators
4. Write a program on Increment and Decrement Operator
5. Write a program on special Operator

## Unit VI

### Managing Input and Output Operations

#### Main goals of the Unit:

It should be complete this unit:

- ❖ Describe the character reading
- ❖ Demonstrate how to write a character
- ❖ Describe data formatted
- ❖ Describe the production formatted

#### Introduction:

Reading, processing and writing of data are three basic functions of a computer programme. Most programmes take data as an input and display data processed on a relevant medium, often called information or results. So far we have seen two methods for providing data for system variables. Assignment statements like `a=7; b=0` are used in assigning variables values. Another form is the input scan function, readable on a keyboard. We used both approaches in most of our previous sample programmes. For results, the `printf` function has been used extensively, which transfers results to a terminal.

In contrast to other high-level languages, C has no input / output clauses in its grammar. The `printf` and the `scanf` function call are both the input / output operations. In C input and output operations with several functions are more or less standardised. The standard library input / output is commonly known.

This unit discusses some I / O features that can be used on many machines without modification. However, the device reference manual must be checked for exact function descriptions and to see what other functions are usable.

It is recalled that in chapter 3 we included in the sample programme a statement that uses the `cos(x)` math library feature. The function `cos(x)` is retrieved from the math library by the compiler and not a C language. Every programme that uses a standard input / output function must have the same effect.

**# include <stdio.h >**

However, there can be exceptions. For the print and scan functions which are specified in the C language for example, this is not suitable. The name of the `stdio.h` file is a shortcut to the standard input / output header file. The command `# include <stdio.h >`, tells the compiler to scan for a file called `stdio.h`, and to put its content in that programme. The contents of the header file become a part of the source code when compiled.

### C Language Input Functions:

The programming language C provides structured feedback features. Inputs are used to translate the device values of the keyboard (input). In the c programming language, the following integrated input functions are given.

#### Scanf () Function:

The scanf () method reads several data values of various forms of keyboard files. In a file header "<stdio.h>," the scanf () functionality is mounted. We have to # include declaration in the respective file header if we want scanf () in our programme <stdio.h>. The scanf () method has the following syntax:

**Scanf("formatting strings",VariableName);**

#### Scanf () Example program:

The image shows a code editor window on the left and a terminal window on the right. The code editor displays the following C program:

```

1 | #include <stdio.h>
2 | #include <scanf.h>
3 | int main()
4 | {
5 |     int a;
6 |     clrscr();
7 |     printf("Enter any value of:");
8 |     scanf("%d",&a);
9 |     printf("you have entered of number",a);
10 |    getch();
11 |    return 0;
12 | }

```

The terminal window on the right shows the program's execution:

```

enter any value:
20
you have entered 20 number

```

In the illustration above, we have used scanf () to read and save the integer value from the keyboard in vector 'a.'

#### getchar () Function:

The getchar () method is the keyboard character reading and returning method. This method is used to interpret a single word. We may compose several characters or read many characters with a looping phrase. Taking the following example of applications.

#### getchar()Example program:

```

File Edit Search Run Compile Debug Project Options Window Help
GETCHAR.C
#include <stdio.h>
#include <conio.h>
main()
{
char ch;
clrscr();
printf("Enter any character:\n");
ch=getchar();
printf("you have entered %c\n",ch);
getch();
return 0;
}
8:13
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu
Enter a value:sadguna
you entered:s

```

If you create and execute the above file, any text can be inserted. The software can only interpret and show the same character when a record is inserted and clicked.

### Getch (): function:

The getch() function is the same as the getchar function. It uses the getch () function to read and return a keyboard character to the programme. This function is used to read a single character. We need to write multiple characters or read multiple characters with a loop statement. Take the following example of software;

### getch()Example program:

```

File Edit Search Run Compile Debug Project Options Window Help
GETCH.C
#include <stdio.h>
#include <conio.h>
main()
{
char ch;
clrscr();
printf("Enter any character:\n");
ch=getch();
printf("you have entered\n",ch);
getch();
return 0;
}
1:1
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu
enter any character:
you have entered\n

```

### Gets() function:

The gets() (feature is used in a text line to interpret and modify a set of characters. The function gets() (reads the line or series of a string before a newline mark is inserted. Taking the following example of applications:

```

File Edit Search Run Compile Debug Project Options Window Help
6233.C
#include<stdio.h>
#include<conio.h>
main()
{
char studentname[40];
clrscr();
printf("enter your name ");
gets(studentname);
printf("%s",studentname);
getch();
return 0;
}

enter your name:sai sri varshini
sai sri varshini_
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu

```

### **Fscanf() function:**

The fscanf () method is used for the file specification. The fscanf () function is used to interpret the file info. If you choose the feature fscanf () to be used, the file should be opened in reading mode.

### **C Language Output Functions:**

The programming language C gives streamlined features. Output Process requires user screen data or printer data, or any computer. The programming language c provides the following regular application functionality

### **Printf () Function:**

Printf () may be used in a written type or in conjunction of string and data values on a device string or data values show. In our script, we must have # as a valid header file <stdio.h> in # if we want to use our printf () function. This printf () must be inserted in a header file. The terminology below applies to the feature printf ().

**Syntax: printf("message");**

### **Printf () Function Example program:**

The screenshot shows a code editor window titled 'PRINTF.C' with the following code:

```

(1)
#include<stdio.h>
#include<conio.h>
main()
{
clrscr();
printf("this is shri nayan");
getch();
return 0;
}

```

The output window on the right displays the text: "this is shri nayan\_".

In the illustration above, we used the `printf()` method to print a string.

The `printf()` feature is often used for viewing data values. We use the data view format string if you want the data values to be shown.

**Syntax:** `printf("formatting string", VariableName);`

### Display Data Values Example program

The screenshot shows a code editor window titled 'DATABVAL.C' with the following code:

```

(1)
#include<stdio.h>
#include<conio.h>
main()
{
int a=12;
float b=1.5;
clrscr();
printf("Data of a,b");
getch();
return 0;
}

```

The output window on the right displays the text: "Data of a,b", "12", and "3.500000".

The `printf()` method is used to print variables `a` and `b` data values to the output monitor in the sample programme above. `a` is an integer vector here so we used string percent `d` and `b` is a floating vector. The `printf()` method may also be used to display string of data values.

**Syntax: printf("String formatting string",VariableName");**

### Display String Formatting Values Example program:



```

#include <stdio.h>
#include <conio.h>
main()
{
    int a=12;
    float b=3.500000;
    clrscr();
    printf("Integer value is %d and float value is %f",a,b);
    getch();
    return 0;
}

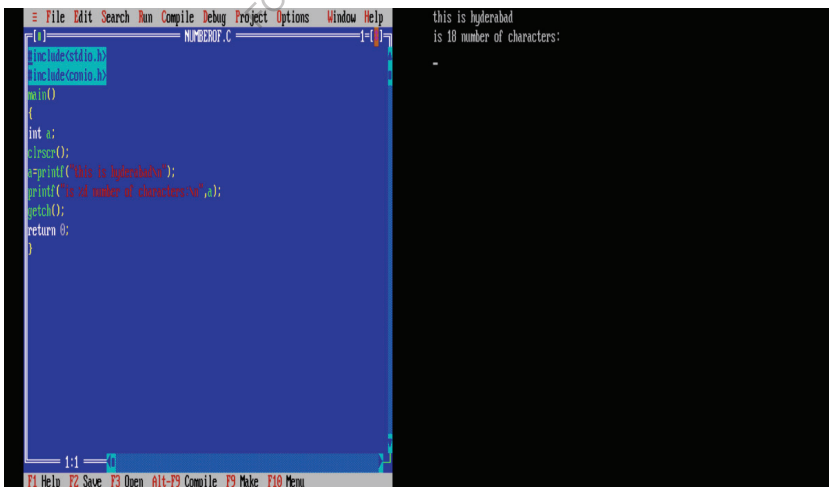
```

integer value=12  
float value=3.500000

In the above method, we look at strings along with data values.

In programming language C, each function must have a return value. It is also possible to return the printf () function by an integer. The printf () method is provided by an integer value equivalent to the total number of typed characters.

### Display Number of Characters Example program:



```

#include <stdio.h>
#include <conio.h>
main()
{
    int a;
    clrscr();
    a=printf("this is hyderabad");
    printf("no of number of characters=%d",a);
    getch();
    return 0;
}

```

this is hyderabad  
is 18 number of characters:



The first feature of `printf ()` in the above-mentioned software is to print "Hyderabad" 18 characters. The integer 18 is then allocated to the "a" variable. The meaning of "a" is printed in the second `printf ()` method.

### Formatted Printf () Function:

The outcome usually occurs in a single line, while `printf ()` displays the output on a single line. The outcomes are seen as we compose many `printf ()` statements. Refer to the following prototype programme

```

#include<stdio.h>
#include<conio.h>
main()
{
 clrscr();
 printf("Hyderabad");
 printf(" 18");
 printf(" this is c programming");
 printf(" language");
 getch();
 return 0;
}

```

Output: this is c programming language


There are 3 `printf ()` statements in the above programme in separate lines, but only one line display the output.

In order to show the output in different lines or however we like us use some special characters named escape sequences. Escape series is the special feature used in `printf ()` method in formatting the display as per user requirement. In the programming language C, we have these escape sequences

S.No.	Name of the Escape Sequence	Meaning of the Escape Sequence
1	<code>\n</code>	Move the NewLine cursor
2	<code>\t</code>	Horizontal tab inserts (5 space characters)
3	<code>\w</code>	Horizontal tab inserts (5 space characters)
4	<code>\a</code>	Beep sound
5	<code>\b</code>	Backspace (removes from its present location the previous character)
6	<code>\\</code>	Backward slash mark incorporates
7	<code>\?</code>	Inserts Object Question mark
8	<code>\'</code>	Single quote mark sign incorporates
9	<code>\"</code>	Inserts Symbol for double quote mark

**Putchar () Function:**

Just one character appears in the putchar () section on the output panel. The putchar () function prints and returns a return value of the same name, and the value is passed as a parameter. With this function, it simply prints a single character. We have a looping phrase to compose or use several times to print many characters. Please take the following sample apps.

**Putchar () Function Example Program:**

```
File Edit Search Run Compile Debug Project Options Window Help
PUTCHAR.C
#include<stdio.h>
#include<conio.h>
main()
{
char ch='c';
clrscr();
putchar(ch);
getch();
return 0;
}
```

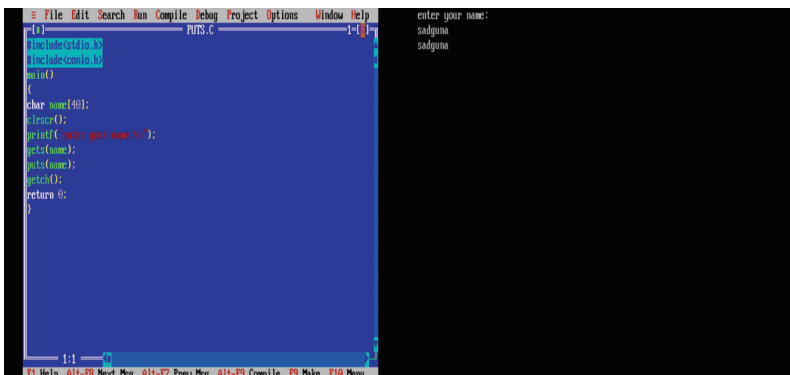
FOR AUTHOR USE ONLY

5:11

F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu

**Puts () Function:**

The puts () method facilitates the output display of a document. The puts () method prints a newline string or series of characters. Taking the software illustration below:

**Puts () Function Example Program:**

```
File Edit Search Run Compile Debug Project Options Window Help
PUTS.C
#include<stdio.h>
#include<conio.h>
main()
{
char name[40];
clrscr();
printf("Enter your name:");
gets(name);
puts(name);
getch();
return 0;
}
```

enter your name:  
sadguna  
sadguna

1:1

F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu

**FPrintf () Function:**

The fprintf () function is used to identify the details. For printing a character, the fprintf () method is used. By using fprintf (), the file must be opened in writing mode.

**Key Concepts:****Multiple Choice Questions (MCQ)**

1. What is the unusual one of the following?  
A) Printf B) scanf C) putchar D) None

Ans: B

2. What is getchar () default return type?  
A) Int B) Char C) char\* D) None

Ans: A

3. How high is the EOF value?  
A) 0 B) 10 C) -1 D) None

Ans: C

4. What is putchar () return value?  
A) The character written B) EOF if an error occurs C) both D) None

Ans: C

5. For a typical program, the input is taken using.  
C) Files B) scanf C) command line D) None

Ans: D

**Fill in the Blanks:**

6. For a typical program, the input is taken using \_\_\_\_\_  
7. Which among the following is odd one out \_\_\_\_\_  
8. The fprintf () function is used to \_\_\_\_\_ the details  
9. The puts () method facilitates the output display of a \_\_\_\_\_  
10. The putchar () function prints and returns a \_\_\_\_\_ of the same name

**Short Questions:**

1. Define Printf () Function?
2. Define Puts () Function?
3. Describe putchar () Function
4. Explain about fscanf () Functions?

**Review Questions:**

1. Explain formatted input and output operations?
2. Explain C language output function?

**Programming Exercises:**

1. Write a program on fscanf () function
2. Write a program on Puchar () function
3. Write a program on Getchar () function
4. Write a program on Gets () function
5. Write a program on Puts () function

FOR AUTHOR USE ONLY

## Unit VII

### Decision Making and Branching

#### Main goals of the Unit:

It should be complete this unit:

- ❖ Briefly explain about Decision Making with If Statement
- ❖ Explain If...Else Statement
- ❖ Describe Switch Statement
- ❖ Explain Conditional Operator
- ❖ Explains how goto statement is used for unconditional branching.

#### Introduction:

We have seen that the C programme represents a number of statements usually made in the order in which they appear. This is when there are no repetitive calculations or options. In reality, however, we face a number of cases when the order of execution of the statements may have to change, depending on certain requirements, or a number of statements repeated until certain conditions are fulfilled. That means determining whether certain conditions occurred or not and directing the computer to carry out certain statements accordingly.

The language has such decision-making capacities by supporting the following statements.

**If statement**

**Switch statement**

**Conditional Operator statement**

**Goto statement**

This are generally considered decision-making statements. Since these statements govern the execution movement, control statements are also named.

We have already included some of these claims in the previous cases. We can explore their capabilities and implementations in more depth here.

#### Decision Making:

The software flow is implemented line by line in the programming language C. Line by line. The software C would then be operated line by line from the key process. However, all programmes can not use this form of running streaming. Often we make different choices, so one or two code lines may be skipped. Imagine a case where we compose a curriculum to assess if a pupil in a specific field has succeeded or failed. We have to verify here if the marks match or do not surpass the pass marks. We assume that the pupil did not fall

otherwise, because the grades are better. To fix these problems in C, we use statements called decision-making statements.

Statements are claims that are used to verify a specified situation and to assess if a sequence of claims may be produced on the basis of the findings of a test. In the programming language C, there are two policy principles that are as follows.

**If Statement**

**Switch Statement**

### **If Statement in C Language:**

In C, if you use a statement to determine a condition. The declaration shall verify the conditions specified and determine whether the declarations shall be made on the basis of the result. C, if the statement is classified into four types as follows

**Simple if Statement**

**If else Statement**

**Nested if Statement**

**If else ladder or if else if Statement**

### **Simple If Statement:**

Easy to use the statement to verify the given statement and to execute the statement block on the basis of the condition result. The basic statement whether the defined status is determined. If it is valid, the next statement or set of statements is made. If this is FALSE, the next declaration or declaration block will be carried out. This is the general syntax and implementation of the fundamental declaration.

### **Simple if Statement Syntax:**

**If (Condition)**

{

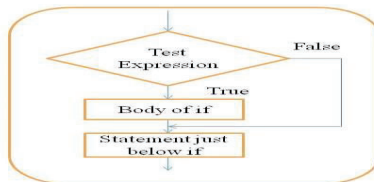
----

**Block of Statements**

----

}

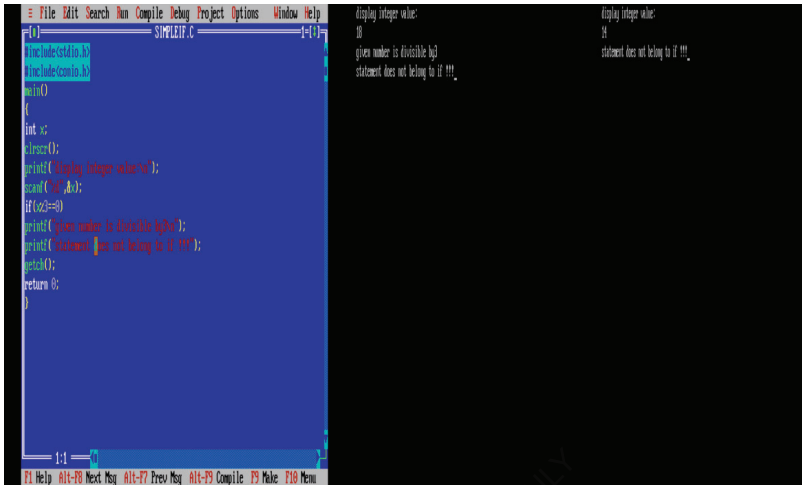
### **Simple if Statement Flow Chart:**



**Figure 7.1. Flow chart of simple if statement**

If only a condition-based method is performed or skipped, easy whether a sentence is used.

**Simple If Statement Example Program:**



**If Else Statement:**

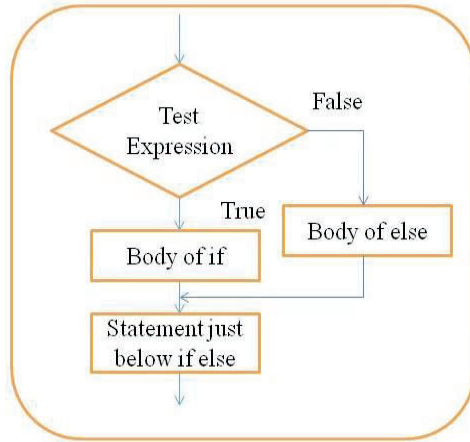
If-else is used to validate the condition and only runs one of the two sets of statements according to the condition outcome. The if-else argument tests the stated specifications. A set of statements is performed if it is True (true list). If the statement is Incorrect, a new declaration process (Bad process) will be carried out. This is the entire if-else expression syntax and execution phase.

**If Else Statement Syntax:**

```

If (Condition)
{
    _____
True Block of Statements;
    _____
}
else
{
    _____
False block statements
    _____
}
    
```

**If Else Statement Flow Chart:**



**Figure 7.2. Flow chart of if else statement**

If the condition consequence (TRUE or FALSE) is to be followed by two choices and only one is to be chosen, so if-else declaration is used.

#### **If else statement example Program:**

<pre> 1  #include &lt;stdio.h&gt; 2  #include &lt;conio.h&gt; 3  main() 4  { 5      int a; 6      clrscr(); 7      printf("enter any integer value\n"); 8      scanf("%d",&amp;a); 9      if(a%2==0) 10         printf("the user given number is even"); 11     else 12         printf("the user given number is odd\n"); 13     getch(); 14     return 0; 15 } </pre>	<pre> enter any integer value: 12 the user given number is even </pre>
--	--

#### **Nested If Statement:**

Nested If C Programming is put If declaration is put in other IF sentences. If you want to verify your condition in C, then it is helpful to nest. When the assertion Otherwise prints many claims depending on the phrase consequence (TRUE and FALSE). Often we have to verify again if it's Valid. But be vigilant when using the C Nested IF expression in those

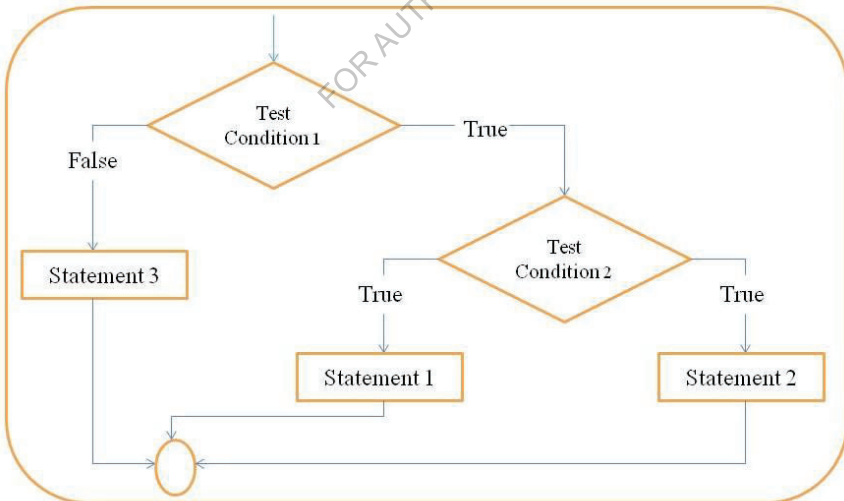


cases. They are not allowed to vote, for example, until everyone is 18 or older. Yet the government isn't going to send everybody. Therefore, another IF assertion named Nested If declaration in C is used to evaluate his birth certificate or other specified identity card of the elector.

**Nested If Statement Syntax:**

```
If (Condition1)  
{  
If (Condition2)  
{  
-----  
True Block of Statements 1;  
}  
-----  
}  
else  
{  
False block of condition1;  
}
```

**Nested If Statement Flow Chart:**



**Figure 7.3. Flow chart of Nested if statement**

If test condition1 is wrong, statement 3 will proceed. If the test requirement1 is correct, the test requirement 2 would be tested. For statement 1, another statement 2 is to be run for True. The nest if argument can be definitive by some mixture of basic if-and-if-else statements.

#### Nested if Statement Program Example:

```

1  #include <stdio.h>
2  #include <conio.h>
3  void main()
4  {
5  int a;
6  clrscr();
7  printf("enter any integer value");
8  scanf("%d",&a);
9  if(a<50)
10 {
11 printf("given number is below 50");
12 if(a<20)
13 printf("it is even");
14 else
15 printf("it is odd");
16 }
17 else
18 printf("given number is not below 50");
19 getch();
20 return 0;
21 }

```

enter any integer value:  
40  
given number is below 50  
it is even

#### If Else If Statement or If Else Ladder:

The if-else-if ladder declaration is an addition of the if-else argument. It is seen in a situation in which multiple cases have to be conducted in varying circumstances. If a condition is valid, the declarations described in the if block will be executed in the if-else-if-ladder argument. If a condition is true the statements stated on the other block are executed; in the other case the statements provided in the opposite block are executed if no condition are correct. Many other potential blocks are available. If the option is implemented rather than prevented, the transfer case declaration is identical if no case is balanced.

#### If Else Ladder or If Else If Statement Syntax:

```

If (Condition1)
{
  _____
True Block of Statements 1;
  _____
}
Else if (Condition2)
{
False block of condition1;
And
True block of condition2
}

```

### If Else Ladder or If Else If Statement Flow Chart

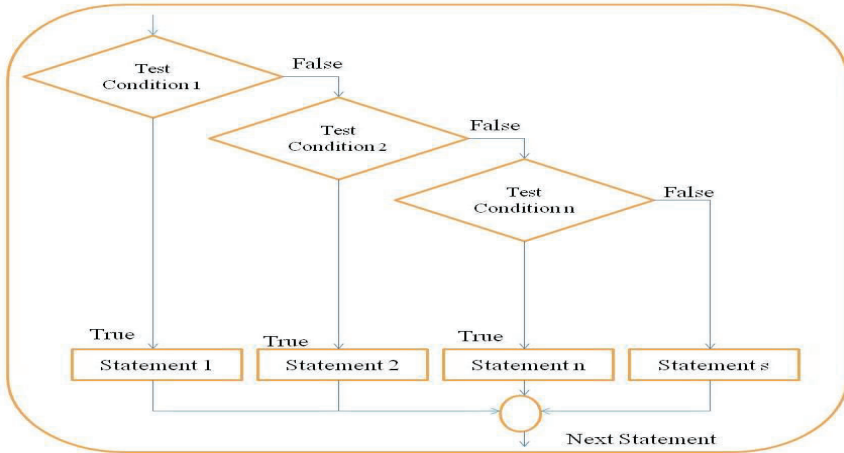


Figure 7.4. Flow chart of if else ladder or if else if Statement

### If Else Ladder or If Else If Statement Program Example

```

File Edit Search Run Compile Debug Project Options Window Help
ELSC171A.C 2=
#include <stdio.h>
#include <conio.h>
main()
{
    int a,b,c;
    clrscr();
    printf("enter any three integer numbers");
    scanf("%d%d%d",&a,&b,&c);
    if(a>=b)&=c)
        printf("a is the big number.");
    else if (b>=a)&=c)
        printf("b is the big number.");
    else
        printf("c is the big number.");
    getch();
    return 0;
}

enter any three integer numbers
15
16
12
16 is the big number
-
1:1
F1 Help F2 Save F3 Open Alt-F9 Compile F9 Make F10 Menu
  
```

### Switch Statement:

Imagine a scenario where we have multiple choices and only one choice has to be selected. If declaration will solve those issues, nested. There are concerns. But with the rising number of options, the difficulty of the system. This form of dilemma can be overcome quickly by the

turn comment. You can conveniently use the turn statement to choose one choice from multiple choices. We have a value that can be contrasted with each choice in the turn sentence. Whenever the value meets the value of an option, this feature starts execution. Per alternative in the switch statement is defined as an event.

### Switch Statement Syntax:

```

Switch (Expression or value)
{
  Case value 1: set of statements;
  —
  Case value 2: set of statements;
  —
  Case value 3: set of statements;
  —
  Case value 4: set of statements;
  —
  Case value 5: set of statements;
  —
  ..
  ..
  Default: set of statements;
}

```

### Switch Statement Flow Chart:

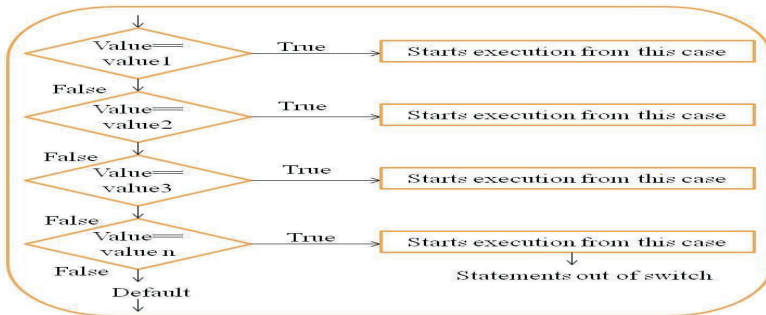


Figure 7.5. Flow chart of Switch Statement

The order contains one or more examples, each with an important purpose. At the start of the switch argument, if the execution begins in the first instance, the first case is compared to switchValue.

The second case value is compared to the SwitchValue, if not balanced with the switch instruction, and execution occurs in the second scenario. This loop persists until a match is made. If the case value in the switch statement does not fit switchValue, a particular case named default is operated.

If a value coincides with the switchValue, it begins for that particular event. The following case statements often begin with this execution phase. At the end of each situation, we use the "break" word to stop it. This means that the statement of split is used to conclude the statement in change. It's easy, though.

### Switch Statement Program Example:

```

#include <stdio.h>
#include <conio.h>
main()
{
    int a;
    clrscr();
    printf("Enter Integer digit\n");
    scanf("%d",&a);
    switch(a) {
        case 1: printf("Monday");break;
        case 2: printf("Tuesday");break;
        case 3: printf("Wednesday");break;
        case 4: printf("Thursday");break;
        case 5: printf("Friday");break;
        case 6: printf("Saturday");break;
        case 7: printf("Sunday");break;
        default: printf("wrong digit");
    }
    getch();
    return 0;
}

```

Output 1:

```

Enter Integer digit
5
Friday

```

Output 2:

```

Enter Integer digit
10
wrong digit

```

### The Conditional (?:) Operator:

The C language has an excellent customer for bidirectional judgments. This operator is a mix of? Then: and there are three operands. Typically this operator is known as the conditional operator. The ultimate syntax is the following in the conditional operator:

**Conditional expression1 ? expression2 : expression3;**

Initial test of the conditional expression. If the outcome is null expression 1, then the value of the conditional statement would be calculated and returned. Another approach is to test expression2 and return its meaning.

### Conditional Operator Flow Chart:

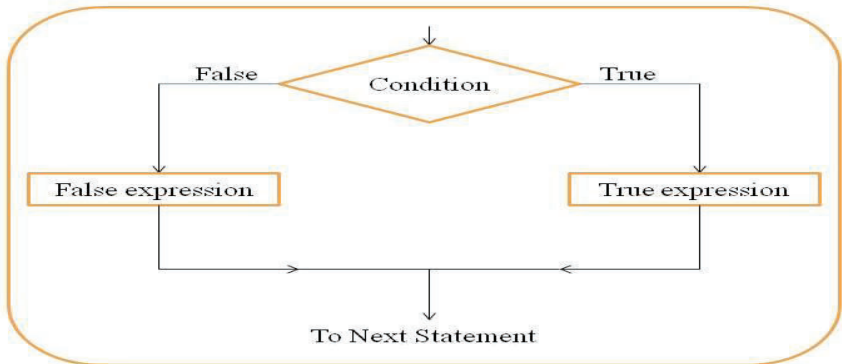


Figure 7.6. Flow chart of Conditional Operator

### Conditional (?:) Operator Example program:

The screenshot shows a C program in a code editor. The code defines a function `main()` that declares a `float p;`, prints the address of `p`, scans a value into `p`, and then uses the conditional operator `p < 0 ? p * 2 : p - 2;` to modify `p`. The program prints the value of `p` and then gets a character from the user. The output window shows the address `12.600000` and the value `-4.759999` after the conditional operation.

```

1 | #include <stdio.h>
2 | #include <conio.h>
3 | int
4 | (
5 | float p;
6 | clrscr();
7 | printf("Enter p:");
8 | scanf("%f",&p);
9 | p < 0 ? p * 2 : p - 2;
10 | printf("Value of p:");
11 | getch();
12 | return 0;
13 | )
  
```

```

enter p:
2.6
(2.600000)-4.759999
  
```

### Break, Continue and GOTO Statements:

In C, there are no control statements that are required to control programme execution. These control statements are referred to as unconditional control statements. C programming language c provides the following unconditional control statements

**Break**

**Continue**

**goto**

There is no prerequisite for the above three statements to monitor programme execution

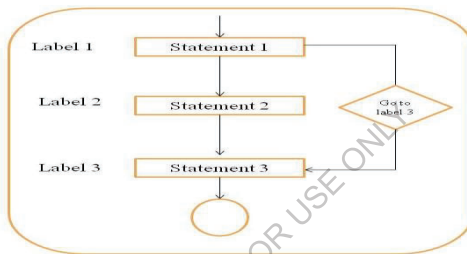
**Goto Statement:**

In the application, the Goto function is used to switch from line to side. We can save with goto statements from top to bottom or from bottom to top. The Goto declaration needs a marker to leap from line to side. Label is a direction or line name of the method. The execution power flows directly to the mark while we use a goto expression in the software.

**goto statement Syntax:**

**Goto label**

**Label : statement;**

**goto statement Flowchart:**

**Figure 7.7. Flow chart of goto Statement**

**goto Statement Example program:**

```

File Edit Search Run Compile Debug Project Options Window Help
GOTO.C 1:1
#include<stdio.h>
#include<conio.h>
main()
{
    int number,i=1;
    clrscr();
    printf("enter the number you want to print from the table");
    scanf("%d",&number);
    mathematicalTable;
    printf("table of %d, number=%d",number,*);
    ***;
    if(i<=10)
    goto mathematicalTable;
    getch();
    return 0;
}

```

```

enter the number you want to print from the table
10
10*1=10
10*2=20
10*3=30
10*4=40
10*5=50
10*6=60
10*7=70
10*8=80
10*9=90
10*10=100

```

1:1

F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu

**Multiple Choice Questions (MCQ)**

- Which one is the conditional Operator  
A) ? :    B) >    C) <    D) None

Ans: A

2. Label is a \_\_\_\_\_ or line name of the method  
B) Direction    B) Link    C) Method    D) None

Ans: A

3. Control statements are referred to as \_\_\_\_\_ control statements.  
B) Unconditional    B) Conditional    C) Addition    D) None

Ans: A

4. If a \_\_\_\_\_ is true the statements stated on the other block are executed  
B) Method    B) Operator    C) condition    D) None

Ans: C

5. The \_\_\_\_\_ declaration assesses the conditions specified  
D) if    B) if-else    C) Nested if    D) None

Ans: B

**Fill in the Blanks:**

1. Goto function is used to \_\_\_\_\_
2. The if-else-if ladder declaration is an \_\_\_\_\_
3. The \_\_\_\_\_ flow is implemented line by line in the programming language C.
4. The \_\_\_\_\_ declaration needs a marker to leap from line to side.
5. \_\_\_\_\_ is used to validate the condition and only runs one of the two sets of statements according to the condition outcome.

**Short Questions:**

1. Define Switch Statement?
2. Define goto statement?
3. Describe Conditional Operator?
4. Explain about if statement?

**Review Questions:**

1. Explain if else Statement with flowchart?
2. Explain Nested if Statement with Flowchart?

**Programming Exercises:**

1. Write a program Conditional Operator
2. Write a program on goto statement
3. Write a program on Nested if else statement
4. Write a program on Switch Statement?
5. Write a program on if statement?



## Unit VIII

### Decision Making and Looping

#### Main Objectives of the Unit:

It should be complete this unit:

- ❖ Know the computer mechanism and how it works
- ❖ Learning how to formulate logic using algorithms and maps

#### Introduction:

We have shown in the previous chapters that a portion of a programme can be repeatedly executed by adding a clock, and then using the if argument. Although for all practical purposes this approach is very adequate, a counter must be initialised and increased and its value must be checked at a suitable position in the software.

Find a situation where we execute the requisite number of statements in a single statement or sequence of statements over and over again. Such issues can be solved by looping definitions in C. For instance, assume we print a message 100 times. If we want to do this task without using looping statements, we have to both write 100 or 100 print statements in a single print statement, and write the same message.

Both approaches are challenging. The same job can be quite easily achieved with looping statements. The loop statements are used to execute a single statement or series of statements repeatedly until the conditions are Incorrect.

C programming language makes three comments on loops:

**While Statement**

**do-while Statement**

**for Statement**

#### While Statement:

Since a pretested loop is sometimes called the loop. Generally speaking, depending on the Boolean state, a certain loop allows several executions of a part of the code. If declaration can be viewed as a repeat. Typically the while loop is used if the number of iterations is not specified in advance.

#### While Statement Properties:

A conditional expression is used to validate state. The statements specified in the whilst loop are executed repeatedly until the condition fails. The condition is correct upon returning 0. If a non-zero number is returned the result is false.

In a loop the condition expression is obligatory.

Without a body it can run a loop for a while.

We may have more than one conditional expression in the while loop.

If there is only one declaration in the body of the loop the braces are optional.

### Syntax of While Statement:

**While (condition)**

```
{
```

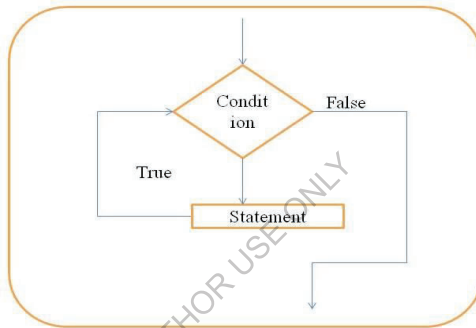
```
----
```

**Block of statements;**

```
----
```

```
}
```

### While Statement Flow chart:



**Figure 8.1. Flow chart of While Statement**

The condition given is evaluated first. If the condition is true the statement or block will be executed individually. If the execution is complete, the condition will be evaluated again. If it is True, it will execute the same statements again. Repeat the same procedure before determination of Right. Whenever FALSE is decided the control of execution moves out the block of the while.

### While Statement Example program:

```

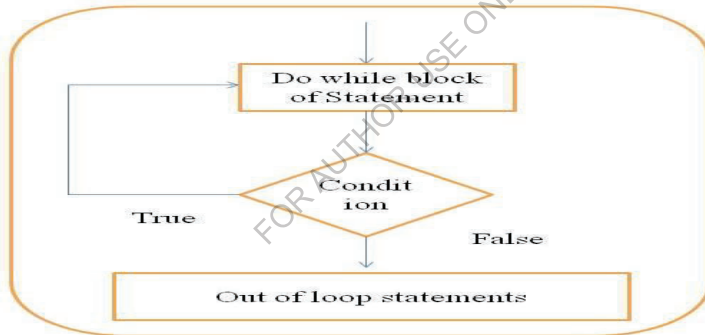
1 | #include <stdio.h>
2 | #include <conio.h>
3 | #include <math.h>
4 | #include <string.h>
5 | #include <stdlib.h>
6 | #include <time.h>
7 | #include <ctype.h>
8 | #include <unistd.h>
9 | #include <sys/types.h>
10 | #include <sys/stat.h>
11 | #include <fcntl.h>
12 | #include <sys/time.h>
13 | #include <sys/resource.h>
14 | #include <sys/mman.h>
15 | #include <sys/wait.h>
16 | #include <sys/queue.h>
17 | #include <sys/epoll.h>
18 | #include <sys/uio.h>
19 | #include <sys/xattr.h>
20 | #include <sys/sem.h>
21 | #include <sys/shm.h>
22 | #include <sys/ipc.h>
23 | #include <sys/mount.h>
24 | #include <sys/quota.h>
25 | #include <sys/rseq.h>
26 | #include <sys/sendfile.h>
27 | #include <sys/signalfd.h>
28 | #include <sys/timerfd.h>
29 | #include <sys/uinput.h>
30 | #include <sys/vfs.h>
31 | #include <sys/xfs.h>
32 | #include <sys/zfs.h>
33 | #include <sys/zpool.h>
34 | #include <sys/zscrub.h>
35 | #include <sys/zstd.h>
36 | #include <sys/zstd_legacy.h>
37 | #include <sys/zstd_legacy.h>
38 | #include <sys/zstd_legacy.h>
39 | #include <sys/zstd_legacy.h>
40 | #include <sys/zstd_legacy.h>
41 | #include <sys/zstd_legacy.h>
42 | #include <sys/zstd_legacy.h>
43 | #include <sys/zstd_legacy.h>
44 | #include <sys/zstd_legacy.h>
45 | #include <sys/zstd_legacy.h>
46 | #include <sys/zstd_legacy.h>
47 | #include <sys/zstd_legacy.h>
48 | #include <sys/zstd_legacy.h>
49 | #include <sys/zstd_legacy.h>
50 | #include <sys/zstd_legacy.h>
51 | #include <sys/zstd_legacy.h>
52 | #include <sys/zstd_legacy.h>
53 | #include <sys/zstd_legacy.h>
54 | #include <sys/zstd_legacy.h>
55 | #include <sys/zstd_legacy.h>
56 | #include <sys/zstd_legacy.h>
57 | #include <sys/zstd_legacy.h>
58 | #include <sys/zstd_legacy.h>
59 | #include <sys/zstd_legacy.h>
60 | #include <sys/zstd_legacy.h>
61 | #include <sys/zstd_legacy.h>
62 | #include <sys/zstd_legacy.h>
63 | #include <sys/zstd_legacy.h>
64 | #include <sys/zstd_legacy.h>
65 | #include <sys/zstd_legacy.h>
66 | #include <sys/zstd_legacy.h>
67 | #include <sys/zstd_legacy.h>
68 | #include <sys/zstd_legacy.h>
69 | #include <sys/zstd_legacy.h>
70 | #include <sys/zstd_legacy.h>
71 | #include <sys/zstd_legacy.h>
72 | #include <sys/zstd_legacy.h>
73 | #include <sys/zstd_legacy.h>
74 | #include <sys/zstd_legacy.h>
75 | #include <sys/zstd_legacy.h>
76 | #include <sys/zstd_legacy.h>
77 | #include <sys/zstd_legacy.h>
78 | #include <sys/zstd_legacy.h>
79 | #include <sys/zstd_legacy.h>
80 | #include <sys/zstd_legacy.h>
81 | #include <sys/zstd_legacy.h>
82 | #include <sys/zstd_legacy.h>
83 | #include <sys/zstd_legacy.h>
84 | #include <sys/zstd_legacy.h>
85 | #include <sys/zstd_legacy.h>
86 | #include <sys/zstd_legacy.h>
87 | #include <sys/zstd_legacy.h>
88 | #include <sys/zstd_legacy.h>
89 | #include <sys/zstd_legacy.h>
90 | #include <sys/zstd_legacy.h>
91 | #include <sys/zstd_legacy.h>
92 | #include <sys/zstd_legacy.h>
93 | #include <sys/zstd_legacy.h>
94 | #include <sys/zstd_legacy.h>
95 | #include <sys/zstd_legacy.h>
96 | #include <sys/zstd_legacy.h>
97 | #include <sys/zstd_legacy.h>
98 | #include <sys/zstd_legacy.h>
99 | #include <sys/zstd_legacy.h>
100 | #include <sys/zstd_legacy.h>
101 | #include <sys/zstd_legacy.h>
102 | #include <sys/zstd_legacy.h>
103 | #include <sys/zstd_legacy.h>
104 | #include <sys/zstd_legacy.h>
105 | #include <sys/zstd_legacy.h>
106 | #include <sys/zstd_legacy.h>
107 | #include <sys/zstd_legacy.h>
108 | #include <sys/zstd_legacy.h>
109 | #include <sys/zstd_legacy.h>
110 | #include <sys/zstd_legacy.h>
111 | #include <sys/zstd_legacy.h>
112 | #include <sys/zstd_legacy.h>
113 | #include <sys/zstd_legacy.h>
114 | #include <sys/zstd_legacy.h>
115 | #include <sys/zstd_legacy.h>
116 | #include <sys/zstd_legacy.h>
117 | #include <sys/zstd_legacy.h>
118 | #include <sys/zstd_legacy.h>
119 | #include <sys/zstd_legacy.h>
120 | #include <sys/zstd_legacy.h>
121 | #include <sys/zstd_legacy.h>
122 | #include <sys/zstd_legacy.h>
123 | #include <sys/zstd_legacy.h>
124 | #include <sys/zstd_legacy.h>
125 | #include <sys/zstd_legacy.h>
126 | #include <sys/zstd_legacy.h>
127 | #include <sys/zstd_legacy.h>
128 | #include <sys/zstd_legacy.h>
129 | #include <sys/zstd_legacy.h>
130 | #include <sys/zstd_legacy.h>
131 | #include <sys/zstd_legacy.h>
132 | #include <sys/zstd_legacy.h>
133 | #include <sys/zstd_legacy.h>
134 | #include <sys/zstd_legacy.h>
135 | #include <sys/zstd_legacy.h>
136 | #include <sys/zstd_legacy.h>
137 | #include <sys/zstd_legacy.h>
138 | #include <sys/zstd_legacy.h>
139 | #include <sys/zstd_legacy.h>
140 | #include <sys/zstd_legacy.h>
141 | #include <sys/zstd_legacy.h>
142 | #include <sys/zstd_legacy.h>
143 | #include <sys/zstd_legacy.h>
144 | #include <sys/zstd_legacy.h>
145 | #include <sys/zstd_legacy.h>
146 | #include <sys/zstd_legacy.h>
147 | #include <sys/zstd_legacy.h>
148 | #include <sys/zstd_legacy.h>
149 | #include <sys/zstd_legacy.h>
150 | #include <sys/zstd_legacy.h>
151 | #include <sys/zstd_legacy.h>
152 | #include <sys/zstd_legacy.h>
153 | #include <sys/zstd_legacy.h>
154 | #include <sys/zstd_legacy.h>
155 | #include <sys/zstd_legacy.h>
156 | #include <sys/zstd_legacy.h>
157 | #include <sys/zstd_legacy.h>
158 | #include <sys/zstd_legacy.h>
159 | #include <sys/zstd_legacy.h>
160 | #include <sys/zstd_legacy.h>
161 | #include <sys/zstd_legacy.h>
162 | #include <sys/zstd_legacy.h>
163 | #include <sys/zstd_legacy.h>
164 | #include <sys/zstd_legacy.h>
165 | #include <sys/zstd_legacy.h>
166 | #include <sys/zstd_legacy.h>
167 | #include <sys/zstd_legacy.h>
168 | #include <sys/zstd_legacy.h>
169 | #include <sys/zstd_legacy.h>
170 | #include <sys/zstd_legacy.h>
171 | #include <sys/zstd_legacy.h>
172 | #include <sys/zstd_legacy.h>
173 | #include <sys/zstd_legacy.h>
174 | #include <sys/zstd_legacy.h>
175 | #include <sys/zstd_legacy.h>
176 | #include <sys/zstd_legacy.h>
177 | #include <sys/zstd_legacy.h>
178 | #include <sys/zstd_legacy.h>
179 | #include <sys/zstd_legacy.h>
180 | #include <sys/zstd_legacy.h>
181 | #include <sys/zstd_legacy.h>
182 | #include <sys/zstd_legacy.h>
183 | #include <sys/zstd_legacy.h>
184 | #include <sys/zstd_legacy.h>
185 | #include <sys/zstd_legacy.h>
186 | #include <sys/zstd_legacy.h>
187 | #include <sys/zstd_legacy.h>
188 | #include <sys/zstd_legacy.h>
189 | #include <sys/zstd_legacy.h>
190 | #include <sys/zstd_legacy.h>
191 | #include <sys/zstd_legacy.h>
192 | #include <sys/zstd_legacy.h>
193 | #include <sys/zstd_legacy.h>
194 | #include <sys/zstd_legacy.h>
195 | #include <sys/zstd_legacy.h>
196 | #include <sys/zstd_legacy.h>
197 | #include <sys/zstd_legacy.h>
198 | #include <sys/zstd_legacy.h>
199 | #include <sys/zstd_legacy.h>
200 | #include <sys/zstd_legacy.h>
  
```

**do while Statement:**

A do while loop is similar to a while loop but only that the statements within the do-whereas structure are executed before checking the condition. On the other hand, the conditions are determined, and the conclusions are taken as the procedure is being carried out. Work on the while. Then you should conclude that if at first there is an incorrect condition then the time runs once, but the while loop does not work. In the sentence the following is syntax:

**do While Statement Syntax:**

```
do
{
-----
Block of statements;
-----
}
While (condition);
```

**do While Statement Flow chart:****Figure 8.2. Flow chart of do While Statement**

The first is to execute the single statement or block of statements in a do sequence. After the do block is executed, the situation is evaluated. When the condition is tested for TRUE, the single statement or block of do clause statements is executed again. If the execution is complete the state is re-evaluated. If true, the same declarations will be executed again. The same process is repeated before Wrong's condition is determined. When FALSE evaluates the state, the while block pushes the execution control out.

**do while statement Example program:**

```

#include <stdio.h>
#include <conio.h>
using namespace std;
int main()
{
    int a=0;
    clrscr();
    printf("even numbers below 30");
    do
    {
        if(a%2==0)
            printf("%d ",a);
        ++a;
    }
    while(a<=30);
    getch();
    return 0;
}

```

even numbers below 30  
0 2 4 6 8 10 12 14 16 18  
20 22 24 26 28 30

### For Loop Statement:

The declaration is used to implement a single statement or series of statements indefinitely, as soon as the assumptions are Valid. The following sentence syntax and flow diagram for loop statement.

The init step is first, and only once. In that step any loop control variables can be declared and initialised. As long as there is a semicolon, you don't have to make a point.

They would then calculate the condition. If real, the body of the loop is executed. If it is wrong, then the loop body will not run, the control flow will switch to the next sentence just after the 'for' loop.

After the body of the 'for' loop executes the control flow springs to the increment statement. By this statement you can change any variables in the loop function. This sentence can remain empty, as long as a semicolon appears after the condition.

The condition is now under re-evaluation. The loop will run if it is true and the process will be repeated (loop body, phase increase then condition again). When the condition becomes incorrect the 'pro' loop stops.

### For Loop Statement Syntax:

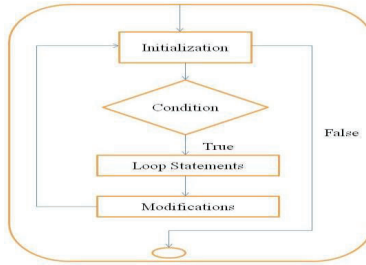
**For (Expression1; Expression2; Expression3)**

```

{
  _____
}

```

do While Statement Flow chart of:



**Figure 8.3. Flow chart of For loop Statement**

Initialization is performed for the sentence, and situation determination is pursued. When determining the TRUE situation, a single declaration or assertion block shall be performed for the declaration. The adjustment declaration is executed following implementation, and the situation is again measured. If it is Valid, so again the same arguments can be made. If FALSE is tested the same procedure will proceed. Whenever FALSE is evaluated the execution control shifts away from the mark.

#### Example program of for loop statement:

The screenshot shows a C++ IDE with a blue background. The code is as follows:

```

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

The IDE interface includes a menu bar with options: File, Edit, Search, Run, Compile, Debug, Project, Options, Window, Help. The status bar at the bottom shows: F1 Help, Alt-F8 Next Msg, Alt-F7 Prev Msg, Alt-F9 Compile, F9 Make, F10 Menu.

#### Jumps in Loops:

Loops perform a sequence of operations continuously before the component does not satisfy the measurement requirement. Replicating the amount of times a loop is replicated, and the test condition is written, is calculated in advance. If a situation arises as a loop is performed it is often preferable to miss or exit a loop. For eg, imagine looking for a given name in a list of 50 names of students. A software loop written for reading and testing student names has to be

terminated 50 times as soon as the required name is found. C lets you jump from one sentence to another in a loop, and leap out of a loop.

### Jumping Out of a Loop:

You will use the split statement and goto statement to get an early exit from a loop. In the turn statement, we noticed the usage of the violation and goto in if-else form. System statements can be used in loops, after and with loops.

### Break Statement:

The break is a C keyword for extracting control of the software from the loop. The break expression is used on internal loops or turn sentences. The sentence break the circuit one by one, i.e. splits the inner circuit first in the case of nested loops and then passes on to external loops. The split sentence in C can be seen in the following 2 scenarios:

- 1) The break state is used to finish the transfer case argument
- 2) The break statement is often used to start, while, do-while and to state the sequence.

If a break declaration is located within the switch case, assertion the execution force leaves the switch claim instantly. Think about the next plan, for example.

### Break Statement Syntax:

**Break;**

### Break Statement Flow chart:

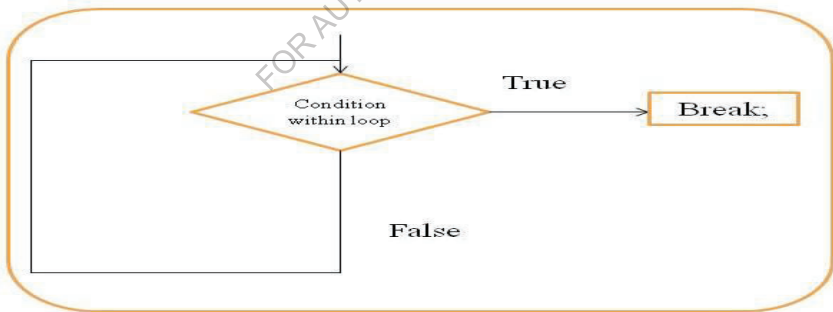


Figure 8.4. Flow chart of Break Statement

### Break Statement Example program:

```

0
1
2
3
4
5
6
7
8
9
10
came outside of loop a=10
-

```

### Continue Statement:

The continue statement is used to pass programme execution power to the start of loop sentence. When a looping declaration detects the continuation phrase, the execution control skips the remaining statements in the loop block and moves straight to the start of the loop.

The Continuing Argument may be used for sentences, after and with looping. The execution mechanism moves straight to the case by utilising continuing phrases and producing comments. When we use continued declaration for declaration, the execution control hops straight into the loop controller (increment / decrement / modification) section. The declaration continuous is as seen in the figure below.

### Continue Statement Syntax:

#### Continue

#### Flow chart of Continue Statement:

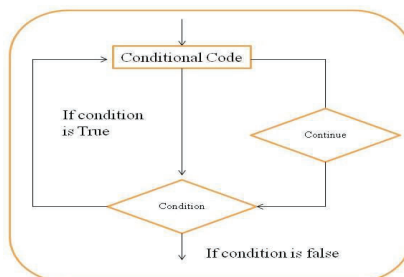


Figure 8.5. Flow chart of Continue Statement

#### Continue Statement Example program:

```

1 | CONTINUE.C | 2/1
#include<stdio.h>
#include<conio.h>
main()
{
int a;
 clrscr();
 for(i=0;i<6;i++)
 {
 if(i==3)
 continue;
 printf("%d",a);
 }
 getch();
 return 0;
}

```

012345

F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu

### Concise Test Expressions:

Even when generating and evaluating statements for branching decisions with nil us use test expressions on the occasion. Since every integer expression has a truth / false value, no clear zero comparisons are needed. For starters, if x is not zero, the term x is real, and if x is zero it is false. That is real! We may write succinct test statements, without using any procedural operators.

### Multiple Choice Questions (MCQ)

- Loops are built using C Language.?  
A) For Block    B) While Block    C) Do While Block    D) All of the Above

Ans: D

- What loop in C Language is faster for, while or Do While?  
A) While    B) For    C) do while    D) All of the Above

Ans: D

- What's the way out of or leave every Loop in C Language suddenly?  
A) Break    B) Continue    C) Leave    D) None

Ans: A

- Can't use the continuing argument of  
A) Switch    B) for    C) While    D) Do While

Ans: A

- Is goto ideal for jumping from key to function?



A) True B) False

Ans: **B**

**Fill in the Blanks:**

1. Switch Statement accepts \_\_\_\_\_
2. A labeled statement consist of an identifier followed by \_\_\_\_\_
3. This loop is supposed to be completed at least once \_\_\_\_\_.
4. The \_\_\_\_\_ is used to pass programme execution power to the start of loop sentence.
5. The \_\_\_\_\_ is a C keyword for extracting control of the software from the loop.

**Short Questions:**

1. Define Break Statement?
2. Define Continue statement?
3. Describe Concise Expression?
4. Explain about While Statement?

**Review Questions:**

1. Explain Do While Statement with flowchart?
2. Explain While Statement with a example Program?

**Programming Exercises:**

1. Write a program to display the first 50 natural numbers
2. Write a program on to display the cube of the number up to given an integer
3. Write a program on to display the multiplication table of a given integer
4. Write a program on to make such as a pattern like a pyramid
5. Write a program on to calculate the factorial of a given number

## Unit IX

### Arrays

#### Main goals of the Unit:

It should be complete this unit:

- ❖ Description about the Array
- ❖ Test how one-dimensional array can be tested and configured
- ❖ Realize the idea of two-dimensional arrays
- ❖ How to describe two-dimensional arrays and launch them?
- ❖ Describe multi-dimensional arrays
- ❖ The Demonstrate Complex Arrays.

#### Introduction:

Up until now we've used different data forms like char, int, float, double and single and double variants. While quite helpful, they are hampered by the reality that a component of this type should display just one value at any time.

But they can only be used with basic data processing. Nonetheless, more data has to be handled in several systems in terms of decoding, encoding and printing. To handle these large quantities of data we need an efficient data type, which allows it simpler to store, view, and modify data objects. C offers an assortment of common type of derivative data for these applications.

When dealing with a great deal of data values we need a variety of variables. Sometimes the procedure is more complex with the expanded number of variables and the author becomes frustrated about the names of the variable. There will be instances where we have to contend with a great many different data principles. To simplify this job, C programming language gives a term named "Collection". An array is a specific type of variable that helps to store several values of the same type of data.

An array includes identical data objects that are placed in a continuous memory position under a single term. A table is an arrangement of sequenced elements of the same form in a fixed dimension. A grouping of this kind of data is easy. At its simplest form an array may be used to represent a set of numbers or a list of names. Few situations the description of the array may be used in:

List of temperatures reported regularly, weekly or annually

- ❖ Set of staff of corporations

- ❖ Collection of products and their prices sold by a store
- ❖ The student's evaluation criteria
- ❖ Client list and frequency of interactions
- ❖ Daily table of precipitation

And so forth

If a collection provides an interesting data structure it is called a data structure in C. Such data structures contain databases, tables, lines, and trees. This text's breadth goes to a comprehensive discussion on all data systems. Nevertheless it is important to accept meanings in Chapter 11 and lists in Chapter 14.

### Declaration of an Array:

If we want to build an array in C programming language, we need to know the form of data value and the quantity of values contained in that array. In language for programming C.

We use the following syntax to create an array:

#### Syntax:

**Datatype Name of the Array [Size of the Array]**

Syntax for generating a list of initial values and measurements

**Datatype Name of the Array [Size of the Array] = {Val1,Val2,.....Valn};**

Syntax for array size and initial value formation

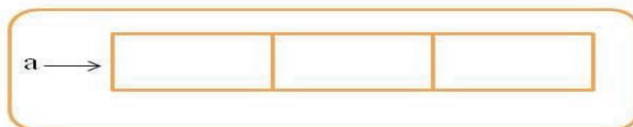
**Datatype Name of the Array [] = {Val1,Val2,.....Valn};**

The data type is the value type in that array and in the above syntax the maximum number of values can be stored in the array.

#### Example:

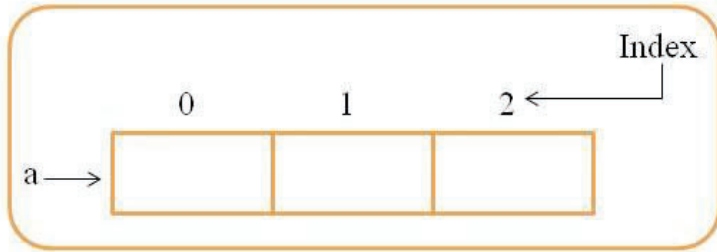
**Int a [3] ;**

The programmer assigns 6 bytes of contiguous 'a' storage positions in which 3 separate integer values (each in 2 bytes of memory) are stored by the programmer in 6 bytes of memory. For the aforementioned argument, the memory is structured accordingly



**Figure 9.1. Array Initialization**

In the aforementioned memory distribution, all three memory positions have a generic name 'a.' Relevant memory coordinates are often not readily available. The programmer then assigns the memory but also provides a numeric reference value to a single memory site in an array. For the latter example, index values are as follows: "index" or "subscript" or "indices"



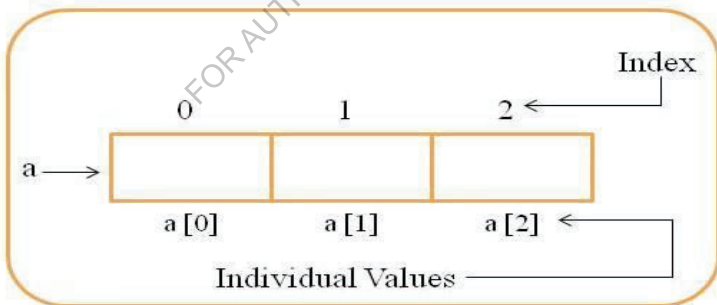
**Figure 9.2. Array Index Value**

#### Accessing Individual Elements of an Array:

The individual items in an array are listed in accordance with the name and index value of the collection. We use the following general syntax to reach individual elements

**Name of the Array [IndexValue]**

For this case, the different elements can be described as follows.



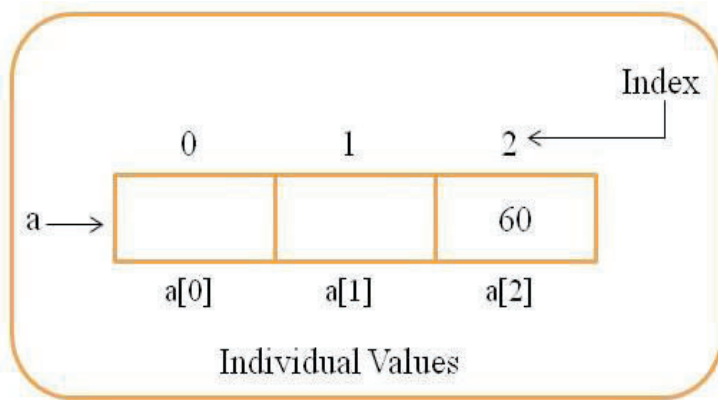
**Figure 9.3. Array Individual Elements of an array**

For eg, if the meaning of the second memory position of 'a' above field is allocated, we use the following argument...

#### Example:

`a [2] =60;`

The product of the aforementioned assignment statement



**Figure 9.4. Assigning individual values**

It is possible to build a succinct and productive programme, for example, a loop model, previously defined as a control variable to read the whole range and evaluate and print out data, through the use of a single name for a list of items and defining the item number.

We may use arrays to display data tables in two, three or more dimensions as well as basic value lists. In this UNIT we discuss how to describe a number and how to produce and execute the following array forms.

**One-Dimensional Arrays or Single Dimensional Array**

**Two-Dimensional Arrays**

**Multi Dimensional Arrays**

**Array Example Program:**

```

File Edit Search Run Compile Debug Project Options Window Help
ARRAY.C
#include<stdio.h>
#include<conio.h>
main()
{
int a[20];
int b,c;
clrscr();
for(i=0;i<20;i++)
{
[i]=50;
}
for(i=0;i<20;i++)
{
printf("Element %d is %d",i,[i]);
}
getch();
return 0;
}
element (0)-50
element (1)-51
element (2)-52
element (3)-53
element (4)-54
element (5)-55
element (6)-56
element (7)-57
element (8)-58
element (9)-59
element (10)-60
element (11)-61
element (12)-62
element (13)-63
element (14)-64
element (15)-65
element (16)-66
element (17)-67
element (18)-68
element (19)-69
1:1
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu

```

**One-Dimensional Arrays:**

A sequence of the same data type values is documented with single arrays for c programming. This implies that a set of values are contained in a single sequence of measurements. In a single dimensional set, data is processed linearly. One dimension arrays, regular arrays or only 1-d arrays are commonly referred to as single-dimensional arrays.

**Declaration of a Single or One Dimensional Array:**

We use the following general notation to define a single dimensional sequence.

**Datatype Name of the Array [Size of the Array];**

**Example:**

**int HTNO[20];**

The above single size array statement reserves 20 continuous storage places with HTNO name of 2 bytes each and tells the compiler to fit integer only in these storage places.

**Initialization of Single or One Dimensional Array:**

We use the following general syntax to define and configure a common range of sizes and initial values.

**Datatype Name of the Array [Size of the Array] = {Val1, Val2 .... Valn};**

**Example:**

**Int HTNO [4] = {001,002,003,004};**

The above statement reserves four neighbouring, 2 Byte, single-dimensional memory positions, each of them identified by the numbers, initialises 001 in the memory first, 002 in the second, 003 in the second third, 004 in the fifth position.

With the following general notation, we can also launch a single dimension array without specifying the size and initial values.

**Datatype Name of the Array [] = {Val1, Val2 .... Valn};**

The array must be initialised if it is generated without defining a size. The array size is calculated by the value numbers initialised.

**Example:**

**Int HTNO [] = {001,002,003,004};**

**Char Name of the Student [] = "Varshini";**

The size of the 'HTNO' array is four in this example and the size of the 'Student Name' array is eight. The compiler stores an extra character named \0 (NULL) at the edge. This is because

**Accessing Elements of Single or One Dimensional Array:**

We use the name of the collection preceded by the index value of an element to reach the elements of a single dimension list. in the programming language.

There must be the index number in the square braces. The index value for the element in the sequence is the reference number assigned to each element during the memory allotment. The index value of each dimension series begins with zero (0) for the first element and increases with one element for each segment. The index number is sometimes considered a graph.

For one-dimensional components we use the following general syntax ...

**Name of the Array [Index Value]**

**Example:**

**Salary [5]=10000;**

The '10000' value is allocated to the third member of the 'pay' list.

### Single or One Dimensional Array Example Program:

```

1  #include <stdio.h>
2  #include <conio.h>
3  using namespace std;
4  int a[5]={'101','102','103','104','105'};
5  int main()
6  {
7      clrscr();
8      printf("HTNO OF STUDENTS\n");
9      for(i=0;i<5;i++)
10     {
11         printf("%d\n",a[i]);
12     }
13     getch();
14     return 0;
15 }

```

HTNO OF STUDENTS  
a1-101  
a2-102  
a3-103  
a4-104  
a5-105

### Two Dimensional Array:

The two-dimensional C-language collection is a collection of arrays. If the data is regular, the One Dimensional Array is used. But the Multi-Dimensional Array must be utilised to deal with multi-level data.

The simplest multi-dimensional array is two dimensional arrays in C. In C Two Dimensional Series, data is saved as row and column like. The record can be accessed via the row indices as well as the column index.

### Declaration of Two Dimensional Array:

As seen below, in C Programming the simple syntax or two-dimensional array statement is:

**Datatype Name of the Array [Row Size] [Column Size]**

**Data Type:**

This defines the form of elements allowed in C by the two-dimensional collection. For starters, if we want integer values to be preserved, the form of data is declared, if we want float values to be kept, and then the float data type is declared.

**Name of the Array:**

This is the name you want to assign to this two-dimensional C series. Student name, ethnicity, qualifications, staff, etc.

**Row Size:**

In an array, the number of the row variable may be placed. For e.g, Row Size = 20 would have 20 rows.

**Column Size:**

You can place column numbers in a sequence. For eg, the list would have 10 columns, column size = 10.

**Example:**

```
Int Staff [4][3];
```

We also used int as a data form to define an array. Therefore, only integer over C two-dimensional array is permitted. If you decide to incorporate float values, you may get mistakes. Workers-C word for the double-dimensional series. The scale of the series is 4. This implies that only four integer values are recognised as groups. If more than 4 values are placed, it will render a mistake. You should hold fewer than 4. For instance, if you save four integer (which is 0), the remaining 2 values will be set to the default value. The scale of the series is 3. The number of staff implies that columns are only defined by three integer values.

If we choose to store more than 3, that will cause a mistake. We may hold less than three. For instance, the remaining 1 is allocated to the default (that is, 0) when two integrator values are placed.

**Two Dimensional Array in C Initialization:**

Initialized in a few respects the two-dimensional C set

**First Approach:**

```
Int Staff [4] [3] = {1,2,3,10,20,30,4,5,6,40,50,60};
```

The first three are the first group, the second three are the second, the third are the third and the five are the lowest, the fourth group is the last three. Here, we divide it into 3, for our column size = 3. The above two-dimensional definition may also be published



```
int staff [4] [3] = {{1,2,3},{10,20,30},{4,5,6},{40,50,60}};
```

The curved braces along each row({}) is rolled up here. It is still a safe idea to use the curly braces to distinguish rows.

### Second Approach:

```
int staff [] [] = {{1,2,3},{10,20,30},{4,5,6},{40,50,60}};
```

The scale of the row and of the column were not seen here. However, the compiler is clever enough to measure the size by testing the amount of objects in the row and column. You may also code this two-dimensional array as c

```
int staff [] [3] = {{1,2,3},{10,20,30},{4,5,6},{40,50,60}};
```

### Access of Two Dimensional Array Elements:

We may reach Two Dimensional Array with indexes in C-programming components. Variable in the array may be accessed or updated individually through the index. The index value for n-1 is 0 and n is the size of column or rows.

### Two Dimensional Array in C Example:

In this two-dimensional array system, we can announce and initialise 2 two dimensional arrays with some values. In order to store the sum of both arrays we shall then define a further double-dimensional array in C.

The screenshot shows a C program in a text editor. The code defines a 2D array 'arr' with 4 rows and 3 columns, and another 2D array 'arr2' with 4 rows and 3 columns. It then calculates the sum of corresponding elements in 'arr' and 'arr2' and stores the result in a third 2D array 'arr3'. The program prints the sum of each element and returns 0.

```

1  #include <stdio.h>
2  #include <conio.h>
3  int main()
4  {
5      int a=0, b=0;
6      int arr[4][3]={{1,2,3},{10,20,30},{4,5,6},{40,50,60},{7,8,9}};
7      int arr2[4][3];
8      for(a=0;a<4;a++)
9      {
10         for(b=0;b<3;b++)
11         {
12             arr3[a][b]=arr[a][b]+arr2[a][b];
13         }
14     }
15     getch();
16     return 0;
17 }
```

The output window shows the following values for the sum array:

```

arr[0][0]=1
arr[0][1]=2
arr[0][2]=3
arr[1][0]=10
arr[1][1]=20
arr[1][2]=30
arr[2][0]=4
arr[2][1]=5
arr[2][2]=6
arr[3][0]=40
arr[3][1]=50
arr[3][2]=60
```

### Multi Dimensional Array:

The multi-dimensional array in C is a multi-dimensional field. In our previous post, we addressed the Two Dimensional Array that is the simplest C Multi Dimensional Array form.

We may announce the n-dimensional array in which n is the C Program Language dimensional number by putting n bracket number[].

For example:

**Int a[2][3][4];**

### **Multi Dimensional Array Syntax:**

**Data type Name of the Array [Tables] [Row Size] [Column Size]**

### **Data Type:**

It shall determine the form of elements it recognises. For starters, if we want integer values to be preserved, the form of data is declared, if we want float values to be kept, then the float data type is declared.

### **Name of the Array:**

In the C multi-dimensional set, this is the term.

### **Tables:**

It can specify the amount of tables an array will accommodate. There is still a two-dimensional table with a central row of lines and columns. In C, on the other side, multi-dimensional arrays are more than one row and column set.

### **Row Size:**

In an array, the number of the row variable may be placed. The number is limited to ten rows, such as Row Size = 10.

### **Column Size:**

You can place column numbers in a sequence. For e.g, for column size = 8, the list would be 8 columns. The maximum number of three-dynamic elements can be calculated by using: [Tables] \* [Row Size] \* [Column Size].

### **Example:**

**Int Staff [2][4][3];**

- ❖ Back we used int as data form to define an array. This set embraces only integer numbers. When you attempt to incorporate float values, it throws a mistake.
- ❖ The name of the table is die Workers
- ❖ The amount of the table = 2. Therefore, this collection comprises up to two data types (rows and columns).
- ❖ The row number of the series is 4. This implies that only 4 integer values are allowed for the workers set.
- ❖ Ah! If we want to store more than 4, it throws a mistake. There are less than four that we can hold for storage. Where 2 integer values have been processed, for example the other two values have a default value (that is 0). The duration of the number is three. It indicates that only three column integer values are allowed by the workers list.

- ❖ Attempts to store more than 3 throws a mistake.
- ❖ At least 3 should we stock. There are only 3. For example, if we store 1 integer value the remaining 2 values are given the default value, which is 0.

Finally, the workers list has the capacity to store a limit of 24 numerical values ( $2 * 4 * 3 = 24$ ).

### Multi-Dimensional Array Initialization:

The C multi-dimensional set is initially feasible in numerous ways

#### First Approach:

##### Int Staff

```
[2][][3]={{(1,2,3),(4,5,6),(7,8,9),(10,11,12),(13,14,15),(16,17,18),(19,20,21),(22,23,24)}}
```

There's 2 tables and the 1st table has 4 rows\*3 columns, while the 2nd table\*3 rows has 4 rows as well.

The first table is comprised of the 3rd parts, second row, third row and fourth row, third row, and last row. Due to the column size = 3, we divided it into 3 here and enclosed each row with curled legs. It is still essential to use curly braces to break the sections.

#### Second Approach:

##### Int Staff

```
[2][][3]={{(1,2,3),(4,5,6),(7,8,9),(10,11,12),(13,14,15),(16,17,18),(19,20,21),(22,23,24)}}
```

Here we didn't notice the row height. However, the compiler is clever enough to evaluate the size of the elements by testing the number in the sequence.

### Multi-Dimensional Array Example:

For some values in this C method, we can declare and initialise Three Dimensional series.

Each value is shown on the for loop in the index sequence.

```

File Edit Search Run Compile Debug Project Options Window Help
MULTDIM.C
#include<stdio.h>
#include<conio.h>
main()
{
    int i,j,c;
    int staff[2][2][3]={{(1,2,3),(4,5,6),
    (7,8,9),(10,11,12)}};
    clrscr();
    for(i=0;i<2;i++)
    for(j=0;j<2;j++)
    for(c=0;c<3;c++)
        printf(" staff[%d][%d][%d]=%d\n",i,j,c,staff[i][j][c]);
}
 getch();
 return 0;
}
1:1
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu
staff[0][0][0]=1
staff[0][0][1]=2
staff[0][0][2]=3
staff[0][1][0]=4
staff[0][1][1]=5
staff[0][1][2]=6
staff[1][0][0]=7
staff[1][0][1]=8
staff[1][0][2]=9
staff[1][1][0]=10
staff[1][1][1]=11
staff[1][1][2]=12

```

### Dynamic Arrays:

The programming language C is allocated to memory variables in the space called the stack. If the software is assembled, the memory allocated to the stack is set. The size of the statement itself must be defined when constructing an array and cannot be modified during programme execution. When we don't realise how many values a show can hold, this is a huge issue. To overcome this, we use the concept of dynamic memory allocation. The heap disc memory assignment is complex. Dynamic resource distribution is as follows:

Dynamic memory allocation during programme execution is named. Resource allocation. Dynamic memory assignment is the method of dynamically assigning the memory while operating the application.

We use predefined or default library functions to dynamically distribute resources. In the 'stdlib.h' file header, FOR default library features are defined. Such are they

**Malloc()**

**Calloc()**

**Realloc()**

**Free()**

#### Malloc():

Malloc () is the default library feature used in the byte and vacuum number block assignment. Any data type on the void pointer can be put as well. If memory cannot be allocated by malloc (), for whatever purpose it returns the NULL pointer.

#### Syntax:

**Void \* malloc (size in bytes)**

#### malloc()Example program:

The image shows a screenshot of a C program being compiled and run. The program uses the malloc() function to dynamically allocate memory for a string. The code is as follows:

```

#include <stdio.h>
#include <stdlib.h>
int main()
{
    char *topic;
    clrscr();
    topic = (char *) malloc(20);
    strcpy(topic, "BASICS OF C LANGUAGE");
    printf("Enter a subject name: ", topic, topic);
    getch();
    return 0;
}

```

The output of the program is shown in the terminal window on the right:

```

string=BASICS OF C LANGUAGE
,address=1400

```

The screenshot also shows the menu bar of the IDE with options: File, Edit, Search, Run, Compile, Debug, Project, Options, Window, Help. The status bar at the bottom indicates the current line and column: 1:1.



```

1 | #include <stdio.h>
2 | #include <stdlib.h>
3 | int main()
4 | {
5 |     char *s;
6 |     clrscr();
7 |     s=(char *)malloc(20);
8 |     strcpy(s, "larshini");
9 |     printf("string address is %d",s);
10 |    s=(char *)realloc(s,30);
11 |    printf("%s",s);
12 |    printf("string address is %d",s);
13 |    free(s);
14 |    getch();
15 |    return 0;
16 | }

```

```

string=larshini,address=1494
string=larshini.in,address=1518

```

### Free ():

Free () has previously been assigned malloc () or calloc () as the basic library function for the memory block. The vacuum pointer function returns free ().In the assigned memory created via calloc () the free () function is used, then all blocks are assigned.

#### Syntax:

**Void free (\*pointer)**

### Free () Example program:

```

1 | #include <stdio.h>
2 | #include <stdlib.h>
3 | int main()
4 | {
5 |     int *ptr = malloc(40*sizeof(*ptr));
6 |     clrscr();
7 |     if (ptr!=0)
8 |     {
9 |         *(ptr+4)=100;
10 |        printf("value of the 4th integer is %d",*(ptr+4));
11 |    }
12 |    free(ptr);
13 |    getch();
14 |    return 0;
15 | }

```

```

value of the 4th integer is 100

```

**Multiple Choice Questions (MCQ)**

1. Array initialization value starts with

- A) 2      B) 1      C) 3      D) 0

Ans: **D**

2. A \_\_\_\_\_ are always stored in sequential memory locations.

- A) Array Element      B) String      C) Pointer      D) None

Ans: **D**

3. \_\_\_\_\_ can be considered as set of elements stored in consecutive memory locations but having same data type

- A) Array      B) String      C) Function      D) None

Ans: **A**

4. \_\_\_\_\_ is an example of compile time memory allocation

- A) Array      B) String      C) Pointer      D) Do While

Ans: **A**

5. The \_\_\_\_\_ of an array is known as elements of an array

- A) Class      B) Objects      C) String      d) None

Ans: **B**

**Fill in the Blanks:**

1. A group of elements of same data type is known as \_\_\_\_\_
2. Types of Arrays \_\_\_\_\_
3. Elements of an array are numbered as 0,1,2,3 these numbers are called \_\_\_\_\_.
4. Array is \_\_\_\_\_ data type in C Language
5. Size of an array is called as \_\_\_\_\_

**Short Questions:**

1. Define Array?
2. Define Malloc ()?
3. Describe Ralloc ()?
4. Explain about Calloc ()?

**Review Questions:**

1. Explain One Dimensional Array with an example?
2. Explain Multi-Dimensional Array with an example?

**Programming Exercises:**

1. Write a program to store elements in an array and print it

2. Write a program to find the sum of all elements of an array
3. Write a program to copy the elements of one array into another array
4. Write a program to print all unique elements in an array
5. Write a program to merge two array to third array

FOR AUTHOR USE ONLY



## Unit X

### Character Arrays and Strings

#### Main goals of the Unit:

It should be complete this unit:

- ❖ Description of a String
- ❖ Explain how to define and configure string variables
- ❖ Explain how terminal strings are read
- ❖ Explain how screen strings are written
- ❖ Define how to manipulate strings

#### Introduction:

String includes a number of characters in double quotation marks. In the programming language C strings are generated with a single dimensional data array. String is enclosed in double quotes and terminated in C programming language with the NULL (\0) character. When c compiler encounters a string value, it will automatically add a NULL (\0) character at the end. Formal interpretation of string is as follows:

There are two methods in C programming language for constructing strings and these are the following:

- ❖ Use of one dimension array type of data (static memory assignment)
- ❖ Usage of a data-type pointer array (dynamic memory allocation)

#### Creating String in C Programming Language:

Strings are formed in C as a specific data-type sequence. The assignment of static and dynamic memory can be used. If a string is generated, the array size must be one larger than the current number of characters to be saved. For saving NULL (\0) string termination, this additional block of memory is used. In the following paragraph, a string of five characters is stored.

**Char Str [10];**

The following sentence produces a string variable of a certain size during software execution.

**Char \*str=(char\*)malloc(20);**

#### Declaring and Initializing String Variables:

C accepts strings as data sort. It helps us to represent strings as character arrays, however. Thus, in C, every appropriate C variable name is specified as a string variable and a character sequence. The basic type of a declaration for a string variable is:

**Char string name [Size of the string];**

Size specifies the number of characters in the name of the series. Such instances are as follows:

**Char State [20];****Char Name of the State [40];**

A null character (`'\0'`) is immediately supplied at the end of the line when a character set is allocated to one character list by the programmer. The actual number of characters should then balance the gap in a string plus one.

Character arrays may be initialised, including integer arrays. C makes the initialization into one of two types of a character array:

```
Char State [20] = "Telangana";
```

```
Char State [20] = {'T', 'e', 'l', 'a', 'n', 'g', 'a', 'n', 'a', '\0'};
```

This state must have 10 elements because the string of Telangana includes 9 characters and one element of space for the null terminator. Note that we have to have null directly Terminator by specifying the elements while initialising a character collection.

C also helps you not to set up a collection of character features. In such instances the size of the collection is calculated dynamically depending on the number of items initialised. For example, the argument

```
Char string [] = {'e', 'x', 'c', 'e', 'l', 'l', 'e', 'n', 't', '\0'};
```

A set of 10 items can define the array series.

It is also necessary to render the initializer size far bigger than the string length. This is the statement. That is the statement.

```
Char str[20] = "excellent";
```

A machine can produce, if enabled, an array of twenty characters, placed "Excellent," end with a zero character, and start all other null items. It's like carrying.

e	x	c	e	l	l	e	n	t	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----	----	----	----	----

However, the accompanying provision is unconstitutional.

```
Char str3[2]="excellent";
```

This leads to a compilation error. Notice that it cannot be distinguished from the statement by the initialization. That is to say:

```
Char str3[10];
```

```
Str3 = "excellent";
```

### Terminating Null Character:

"The final character, who do we need?" You may wonder, "A series, as we know, is not a C-type data but an array-specified data structure. The string is a variable length form and is placed in a set sequence. The image is not always wider than the string placed on the computer. The screen display is not always smaller. Consequently, the last part of the sequence must not be the peak. It is important to determine the end of the string date and the null character is the end of the series.

### String and its Character Frequency Example Program:

```

1  #include <stdio.h>
2  #include <string.h>
3  int main()
4  {
5      char str[500],ch;
6      int count=0,i;
7      clrscr();
8      printf("Enter a string:\n");
9      fgets(str,sizeof(str),stdin);
10     printf("Enter a character to find its frequency:\n");
11     scanf("%c",&ch);
12     for(i=0;str[i]!='\0';i++)
13     {
14         if(ch==str[i])
15             ++count;
16     }
17     printf("Frequency of %c is %d",ch,count);
18     getch();
19     return 0;
20 }

```

```

enter a string:
this is c language programming
enter a character to find its frequency:
g
frequency of g=4

```

### Reading Strings from Terminal:

#### Using Scanf() function:

The standard scanning function may be used in a character string with the definition of a percentage scale.

#### Example:

```
Char name[10]
```

```
Scanf("%s",&name);
```

The issue with scanf () is that its input finishes with the first white space detected. The blanks, buttons, returns, feed type and new lines are used in a white field. If the terminal joins the next text section.

Only after the term 'sai' the string is interpreted by the array address is interpreted as the blank space.

The scan function immediately terminates the string with a zero character and the feature list should be sufficiently wide to contain the input string plus a zero. Notice that in comparison to previous calls, ampersand (&) is not required before the variable name.

The address list is provided in the memory below:

S	A	I	\0	?	?	?	?	?	?
0	1	2	3	4	5	6	7	8	9

Must be mindful that abandoned sites are packed with garbage. If we want to read the whole 'SAI BABA' section, we may use two character arrays of the required duration.

```
Char address1[10],address2[10];
Scanf("%s%s",address1,address2);
```

For the document lines

**SAI BABA**

Assigns address1 to the string "SAI" and address2 to the string "BABA."

Five terms may appear on the computer throughout the show sequence. Notice that the string "sai sri" is treated as two terms, and the string "sai sri" is treated as one phrase.

**Words Using Scanf Function Example program:**

```

E File Edit Search Run Compile Debug Project Options Window Help
STRINGS.C 3-1-1
#include<stdio.h>
#include<conio.h>
main()
{
char name1[10],name2[10],name3[10],name4[10],name5[10];
clrscr();
printf("enter name1\n");
scanf("%s",name1);
scanf("%s",name2);
scanf("%s",name3);
printf("%s\n",name1);
printf("%s\n",name2);
printf("%s\n",name3);
printf("%s-%s\n",name1,name2);
printf("%s-%s-%s\n",name1,name2,name3);
printf("%s-%s-%s-%s\n",name1,name2,name3,name4);
printf("%s-%s-%s-%s-%s\n",name1,name2,name3,name4,name5);
getch();
return 0;
}
enter name:
sri
ramla
sai
sri
varshini
name1-sri
name2-ramla
name3-sai
name4-sri
name5-varshini
6:18
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu

```

In the scan statement the field width may be defined with type percentage ws to interpret a given number of characters of the input set.

```
scanf("%ows",address);
```

Here are two things that may happen:

1. The width *w* is equal or larger than the amount of characters reached. The whole string is stored in the string attribute.
2. The *ws* of the string is smaller than the character total. The influx is minimised and left unread.

Find the points below:

```
Char name[20];
```

```
scanf("%s",address);
```

Shri will be preserved as input string:

s	h	r	i	\0	?	?	?	?	?	?	?	?	?	?	?	?	?	?	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

The input string Nagaprasad will be stored as:

n	a	g	a	p	R	a	s	\0	?	?	?	?	?	?	?	?	?	?	?
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

### Reading a Line of Text:

We also found that a `scanf()` with percent `s` or percent `ws` will interpret only strings without whitespaces. In other terms, a document of one or more terms cannot be used for comprehension. However, C uses a particular format known as the percent editing conversion code to read a line containing a variety of characters, including whitespaces. Note that this conversion method is included in chapter 6.

### Using `getchar` and `gets`:

We discussed the way to read the book from a single character in chapter 6. We are able to interpret and position successive characters repeatedly from the input into a character list. Therefore, a complete text line may be read in an array and processed. You enter the newline character ("`\n`") and attach the null character at the bottom of the number. The lecture is done. The type for a call is as follows:

```
Char ch;
```

```
Ch = getchar();
```

Please notice that there are no `getchar` conditions.

### Getchar Example program:

```

#include<stdio.h>
#include<conio.h>
main()
{
char a;
clrscr();
printf("you are a good boy (press Y OR N)");
a=getchar();
if(a=='y')
{
printf("yes you are a good boy;");
}
else{
printf("oh? you are not doing good (press)");
}
getch();
return 0;
}

```

you are a good boy (press Y OR N)  
y  
yes you are a good boy\_

A simpler approach to interpret a string of text comprising whitespaces is to use a library feature in the <stdio.h> header format. This is a basic function with the following parameter list:

**gets(str);**

str is a vector in string properly defined. The keyboard reads characters to a new character sheet. Unlike scanf (), it doesn't miss whitespaces.

#### Writing Strings to Screen:

We used the print feature in a percentages format, in addition to the printing strings on the screen. For presenting a set of zero characters, the percentage s format may be used. For e.g, the statement:

**Printf(“%s”,name);**

You will see the whole contents of the array label.

You may also designate the shown sequence for accuracy. The specification specifies, for instance, that in 10 field width columns, the first four characters should be written.

#### Arithmetic Operations on Characters:

Arithmetic functionality was used to execute arithmetic operations such as adding and subtracting character C. It is used to modify strings. The characters are immediately translated to integer for use during the arithmetic process, i.e. ASCII meaning individual. Here's an illustration of the C language arithmetic character,

#### Arithmetic Operation on Characters Example program:



```
If(studentname1==studentname2)
```

```
If(studentname=="na g")
```

Not approved. Not necessary. Therefore, the two lines must be matched, character by character to be tested. The connexion is formed until there is a discrepancy or one of the strings ends in zero.

### Comparison of Two Strings Example program

```

1:() COMPARI.S.C
2:#include<stdio.h>
3:#include<string.h>
4:main()
5:{
6:char p[50],q[50];
7:clrscr();
8:printf("Enter a string\n");
9:gets(p);
10:printf("Enter a string\n");
11:gets(q);
12:if(strcmp(p,q)==0)
13:printf("The strings are same\n");
14:else
15:printf("The strings are not same\n");
16:getch();
17:return 0;
18:}

```

enter a string  
sai  
enter a string  
sai  
the strings are same:

### String handling functions in C:

C Programming language has a collection of pre-defined functions, called string handling functions, known to deal for string values. The string handling functions are specified in a header file named <string.h>. If you choose to use string processing, you have to use the file header named <set.h>. able below includes regular feature for string handling.



S.No.	Name of the Function	Syntax	Example	Explanation
1	strcpy()	strcpy	(string1,string2)	Copies the value string2 to string1
2	Strncpy()	Strncpy	(string1,string2,5)	Copy string 2 to 1 of the first five characters
3	Strlen()	strlen	(string1)	Returns total string character numbers 1
4	Strcat()	strcat	(string1,string2)	String2 blends into string1
5	Strncat()	strncpy	(string1,string2,4)	Applies string2 to string1 for first 4 characters.
6	Strcmp()	strcmp	(string1, string2)	Returns 0 where string1 and string2 are identical;
7	Strncmp()	strncmp	(string1,string2,4)	If string1 < string2 is less than 0, if string1 > string2 greater than 0
8	Strcmpi()	strncpi	(string1,string2)	The case ignoring case (upper or lower) compares two strings, str1 and string2
9	Stricmp()	stricmp	(string1, string2)	Compare two strings (such as strcmp()) (of string1 and string2 by disregarding case
10	Strlwr()	strlwr	(string1)	Conversion of all string1 characters to lower case.
11	Strupr()	strupr	(string1)	Turns all string1 to upper case characteristics.
12	Strdup()	string1 = strdup	(string2)	String2 is given a duplicated value for string1
13	Strchr()	strchr	(string1, 'b')	Returns the first time the character 'b' is shown in string1
14	Strrchr()	strchr	(string1, 'b')	Returns a pointer to string 'b' for the last occurrence1
15	Strstr()	strstr	(string1, string2)	Returns a signal for the first string2 event in string1
16	Strset()	strset	(string1, 'B')	Sets all string1 characters to 'B' character defined.
17	Strnset()	strnset	(string1, 'B', 5)	Sets first five string1 characters to the given 'B' character.
18	Strrev()	strrev	(string1)	The value of string1 is reversed

### Table of Strings:

We also use strings files, including a pupil class names chart, a business name chart, a position name list, etc. A string table can be shown in the list and a two-dimensional character sequence can be shown for the whole set.

### Multiple Choice Questions (MCQ)

1. What is the maximum length of a C String.?  
A) 32    B) 64    C) 256    D) 0

Ans: **D**

2. The \_\_\_\_\_ character is used to indicate end of string in C programming.  
A) \*\*      B) &&      C) 1      D) '\0'

Ans: **D**

3. The char data type in C program occupies \_\_\_\_\_ byte of space in memory.  
A) 1      B) 2      C) 3      D) None

Ans: **A**

4. \_\_\_\_\_ Function can be used to read two words at a time.  
A) Gets()      B) puts()      C) getch()      D) None

Ans: **A**

5. \_\_\_\_\_ Function can be used to read two words at a time.  
A) St ()      B) strnset ()      C) str ()      d) None

Ans: **B**

**Fill in the Blanks:**

1. \_\_\_\_\_ are formed in C as a specific data-type sequence.  
2. C Programming language has a collection of \_\_\_\_\_  
3. Strings are formed in C as a specific \_\_\_\_\_.  
4. Which of the following is a \_\_\_\_\_

**Short Questions:**

1. Define Table of Strings?  
2. Define strcmp?  
3. Describe strrev?  
4. Explain about strchr?

**Review Questions:**

1. Explain String Handling function in C Language?  
2. Comparison of Two Strings with example?

**Programming Exercises:**

1. Write a program to Comparison of Two Strings  
2. Write a program Strings Together  
3. Write a program to Arithmetic Operation on Characters  
4. Write a program to getchar example  
5. Write a program to Words Using Scanf Function

## Unit 11

### User Defined Functions

**Main goals of the Unit:**

It should be complete this unit:

- ❖ User-specified outline functions
- ❖ Classify user-defined function elements
- ❖ Clarify the various types of features
- ❖ Learn the concept of recurrence
- ❖ Define the array process
- ❖ Discuss the value of storage class variables, visibility and lifespan.

**Introduction:**

An earlier described was one of the strengths of C language functions. It's simple to describe and use. Up to now we have used functions in every programme we've discussed. However, the three functions were mainly constrained, namely main, printf, and scanf. In this unit we shall take into account in detail the following.

- ❖ How to build a function
- ❖ How to include a function in a programme
- ❖ How two or more functions can be combined
- ❖ How they converse with each other

C functions can be divided into two groups, namely the library and the specified functions. Main is an example of a user defined function. It belongs to the group printf and scanf library function. The key distinguishing feature between these two types is that we don't need library functions to be written when a user-defined function is to be created by the one used during programme writing. However, the specified user function can later become a part of the C programme library. We can use other functions in the library, such as sqrt, cos, strcat etc. On the face this is one of the strengths of the C language.

**Need for User Defined Functions:**

The main function in C is recognised in particular, as mentioned above. Each software must have a principal feature representing the programme's execution. While any application that just uses the key feature can be programmed, it creates several issues. The programme is too broad and difficult to test, review and manage as well. It separates a module and each variable can then be independently coded and merged into one structure. Independently programmed sub-programmes can be understand, debugged and checked more quickly. C applies to these parts as features.

At various stages in the device these forms in operations or calculations also occur. For starters, we could use a factorial number at several points in the software. We may repeat programme definitions in such cases, where appropriate. The call and use when appropriate is another way to build a function. It saves time and money. This separation strategy specifically offers different benefits:

In this programming form, the theoretically high level of the overall issue is addressed first, while the particulars of each feature at the lower level are explored later.

The length of a source programme is decreased by utilising functions at the right spot. This feature is particularly relevant for microcomputers with a small memory space. A defective mechanism is easy to find and separate for further examination.

Functionality can be seen for many other applications. A C programmer should expand from what others have already achieved instead of beginning it from scratch.

#### **A Multi-Function Program:**

Functionality is a self-contained block of code that executes a particular purpose. A black box may be used to take any details from the main software and give a value back after a function has been created and packaged. Like the rest of the system, the operational requirements are ambiguous. All regarding a function is understood in the programme: what is and what is. A series known as black box functions can be configured for any C system. Take into account the following declarations:

```
Void printline(void)  
{  
Int a;  
for(a=0; a<20; a++)  
printf(" ");  
printf("\n");  
}
```

The above statements collection specifies a printline feature that prints 39-character line duration. In the following application, this function may be used:

```

1 | #include <stdio.h>
2 | #include <conio.h>
3 | int main()
4 | {
5 |     printf();
6 |     clrscr();
7 |     printf("Defining C Functions\n");
8 |     printline();
9 | }
10 | void printline()
11 | {
12 |     int a;
13 |     for(a=0;a<39;a++)
14 |         printf(" Language\n");
15 |     printf("%d\n", a);
16 |     getch();
17 |     return 0;
18 | }

```

There are two user functions described in the above programme;

The users in the above software define two functions:

**main() function**

**printline () function**

The key role often starts with the implementation of the software as we know. The first declaration of the key feature is

**Printline ();**

It demonstrates that the position of the print-line must be performed. The control software is then converted to the printer feature. After carrying out the printline feature, a 39-character line is generated; the power is passed back to the main line.

At the moment of the call the operation can now begin. After the Printf statement is introduced to again print the document, the controller is moved back to the printer line feature.

This key function calls the user specified printline and the print library function twice. We should remember that 39 times the library printf function is named.

There may be a need for some other function. She should call herself individually as well. A "called method" may otherwise be called a feature; a feature may be named more than once. This is perhaps one of the most significant technological functions.

The roles that shape a complete programme do not comprise of any predestined partnerships, precedent codes, or hierarchies, but for the starting point. The roles may be put in any sequence. Before or after the request, a feature named may be created. All the roles you name are also natural to conclude.

### **Modular Programming:**

This is a technique utilised in the design and production of information systems. The concept is that a wide-ranging programme is divided into tiny, separate programmes defined as individually called modules. These components are specifically implemented in a system specific context. This is essentially a separated and fragmented way of troubleshooting.

The modules are described and designed to be implemented in a hierarchical system. Module in C corresponds to a specific feature of work.

Any modular programming features are given below:

- ❖ Towards a module only one thing should be accomplished.
- ❖ A calling element facilitates intermodal contact.
- ❖ The module can only be named by a higher level
- ❖ The models not are explicitly intertwined without a partnership named.
- ❖ Both modules have been developed as independent input systems with single output control structures.

### **Elements of User Defined Functions:**

A variety of data forms and variables have been addressed and included in our systems. The key feature was, however, the declaration and usage of certain variables. Therefore, as every other attribute in C systems, we may describe and use functions. Therefore, it is no wonder that certain parallels occur in C regarding functions and variables.

- ❖ Identifiers shall be called the names of all attributes and variable names, thus keeping with the identifier laws.
- ❖ The functions have forms, including variables, correlated with them
- ❖ Like variables, until use of the programme names and feature types must be specified and described.
- ❖ To use a user-defined function, we must create elements relevant to functions.

#### **Function Definition**

#### **Function Call**

#### **Function Declaration**

It is a different software module built to satisfy the specifications of the application. We must use this function in the right place in the software. This is also the calling feature. The calling

method, or feature, is called the calling software. The method of calling. Every subsequent purpose should be specified in the software. This is regarded as the feature declaration or system test.

**Definition of Functions:**

The function specification, also known as feature implementation, shall include the following components.

Function Name

Function Type

List of the Parameters

Local Variable Declaration

Function Statements

Return statement

All six components are split into two components,

The headers are the first line of text, feature form and parameter list. The armbands function as a compound statement is the functional body.

**Function Header:**

The function header consists of three sections, the function class, often called the name of the return function, and the formal parameter list. Please notice that a semicolon is not used at the end of the feature header.

**Name and Type:**

The feature class determines the form of value to return to the programme naming the option as either float or double. C assumes that the retour form is not clearly specified. The return form must be defined as void if the function returns nothing. Note that Void is one of the essential C data forms. And if an integer is a reasonable programming technique, the specification specifically codes the form returning. The value returned is the performance generated by the function.

The feature name is a legitimate C identifier and may thus obey the same laws of definition as other variable names in C. The name should be fitting for the purpose of the feature. Although it must be taken care not to repeat standard library names or instructions.

**Formal Parameter List:**

To conduct the stated procedure the input data for the process is used. They are also called formal parameters, since they represent the actual nature of the data. The parameter list displays the variables from the calling device obtaining results. These parameters can also be used to give the calling programmes values, which are later, included in our functions review.

Parameters are often called claims. The set parameter involves commas separating and parenthesizing variables definitions.

There is no approach to acquire values even in the calling programme. In these cases, there are no specific function parameters. We use the vacuum keyword in these brackets to suggest that the parameter list is null

```
Void printline (void)
{
-----
}
```

No input or meaning is provided to this function. This function, most compilers recognise and do no such thing as null parentheses.

```
Void printline()
```

However it is a programming style to use void to show a parameter chart.

### Function Body:

The functional body shall contain declarations and statements necessary to carry out the task.

The body is in braces in the following order:

- ❖ Local statements defining the feature variables expected
- ❖ Statement of the task performing function
- ❖ The return statement returns the value of the evaluated function;

If the function does not return any value, we may remove the return comment. Remember, however, that the form of return should be specified as void, and it is also good for the return declaration also for void functions.

### Return Values and their Types:

As stated above, a function may either return or return a calling method. If so, the declaration of return is included. While certain values can be transferred to the function named, the function called can return only one value per request. One of these types may be the return declaration:

```
Return;
```

Or

```
Return expression;
```

The first one, the easy return, provides little value; it functions as the mechanism closing cap. If a return is detected, the control is returned immediately to the calling method. An example of a simple return is given as follows:



If (error)

Return;

The second return sort returns the value of the expression, e.g. the expression function.

```
Int add(int a,int b)
{
Int c;
C=a+b;
Return (C);
}
```

Returns the c value arising from both a and b. The following may be merged in the two above statements:

```
Return(a+b);
```

There could be more than one return sentence. The value returned is dependent on such criteria. For example, this condition happens

```
If (a==0)
Return(0);
Else
Return(1);
```

What kind of data does a feature return? Both roles are restored by design to the data form.

But what happens if a function returns another type? When a feature header form specifier is used as previously mentioned, we may compel a feature to return a specific data form.

When a value is retrieved, the value is immediately added to the feature type. The returned value of functions using duplicate but return ints is reduced to an integer. For example, the value 7 is returned only in the integer portion of the test.

```
Int item(void)
{
Return(2.5*3.0);
}
```

**Function Calls:**

### Function Calls:

If parentheses are detected, functionality may be named by simply using the function name and the actual parameters set.

```
main()
{
Int a;
c=add(2,3);
Printf("%d\n",a);
}
```

If the compiler detects a feature order, the instruction is forwarded to the add () feature. The mission will be implemented according to the description line by line and a value will be returned when a return statement is detected. It is given one name. The following is displayed:

The call method transmits two integer values 2 and 3.

```
Int add(inta,intb)
```

It is for a and b allocated. The method determines the item a and b adds the outcome to the local variable c and returns the value to the header where b is allocated again. There are numerous ways to call a role.

#### **Function Call:**

A calling feature is a postfix expression. The operator (..) is proceeding. So a function call for an expression first checks a function call, until it is used to alter the order of precedence using parentheses.

In a function call, the operand is the function name and the parenthesis is determined by the operator (..). The parameters commonly used must be contrasted to the formal parameters by sort, order and number. The commas have some individual criteria to differentiate.

#### **Function Declaration:**

As with the variables, all features of the software must be defined before they are called. A statement of four features

**Function Type**

**Function Name**

**Parameter List**

**Terminating Semicolon**

#### **Category of C Functions:**

In C can be called an argument or a non-argument element. This function will return the call function with values or may not return them. Both C functions may be called by the C software, either with arguments or without arguments.

Such attributes may either be recovered, or cannot. Therefore, the reference functions of a function in C:

Function with No arguments and No return values

Function with arguments but No return values

Function with No arguments but return values

Function with arguments and return values

### Function with No arguments and No return values:

If there is no statement in the method, no data is obtained in the calling function. Likewise, when a value is not received, the called function would not collect any data from the called feature.

**Syntax:**

**Function Declaration:** void function();

**Function Call:** Function();

**Function Definition:**

Void function()

{

Statements;

}

### Example Program:

```

File Edit Search Run Compile Debug Project Options Window Help
FUNCTION.C
#include<stdio.h>
#include<conio.h>
main()
{
void multiplication();
int a,b;
clrscr();
printf("enter two values:\n");
scanf("%d %d",&a,&b);
printf("c=%d",a*b);
getch();
return 0;
}

enter two values:
5
5
c=25_

F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu

```

### Function with arguments but No returns values:

If there are reasons for a method the calling method collects data but returns no information.

**Function Declaration:** `void function( int);`

**Function Call:** `Function(a);`

**Function Defination:**

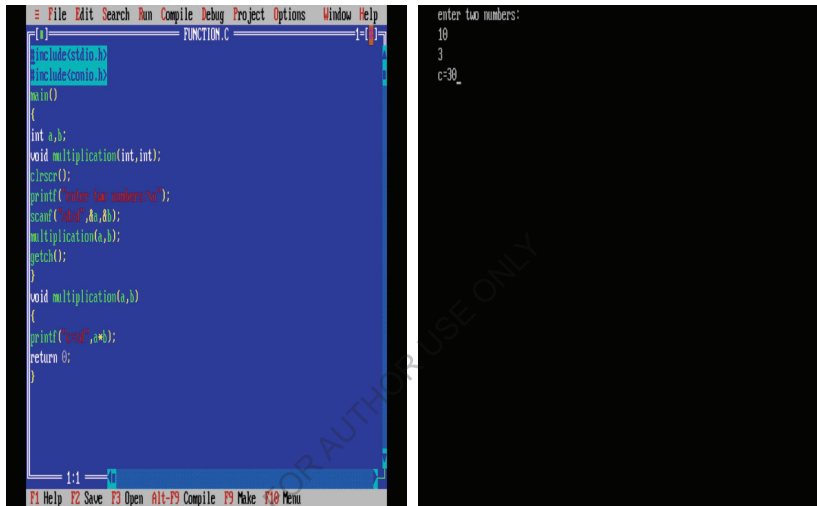
**Void function(int a)**

{

Statements;

}

### Example Program:



```

1  #include <stdio.h>
2  #include <conio.h>
3
4  int main()
5  {
6      int a,b;
7      void multiplication(int,int);
8      clrscr();
9      printf("Enter two numbers:\n");
10     scanf("%d %d",&a,&b);
11     multiplication(a,b);
12     getch();
13 }
14
15 void multiplication(a,b)
16 {
17     printf("%d * %d = %d\n",a,b,a*b);
18     return 0;
19 }

```

enter two numbers:  
10  
3  
c=30

### Function with No arguments but return values:

Often we have to construct functionality that can not contest but return a value to the calling user. An example is `getchar` because it gives a symbolic integer data of a character.

**Function Declaration:** `void function( );`

**Function Call:** `Function();`

**Function Defination:**

**int function()**

{

Statements;

Return a;

}

### Example Program:

```

File Edit Search Run Compile Debug Project Options Window Help
[+] RTN1981.C 2-1
#include<stdio.h>
#include<conio.h>
#include<math.h>
main()
{
int c;
int m1();
clrscr();
c=m1();
printf("Sum=%d",c);
getch();
}
int m1()
{
int a,b;
printf("Enter two numbers:\n");
scanf("%d %d",&a,&b);
return(a+b);
}
:1
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu

```

```

enter two numbers:
6
6
sum=36_

```

### Function with arguments and return values:

The data is transferred from the call function (parameters) and also from the call function (return value). Data is passed from the call function. The execution control transfers from the call function to the parameters named function and execute a procedure, and then returns the call function with a return value.

**Function Declaration:** void function( int );

**Function Call:** Function(a);

**Function Definition:**

```
int function(int a)
```

```
{
```

**Statements;**

**Return a;**

```
}
```

### Example Program:

```

File Edit Search Run Compile Debug Project Options Window Help
[+] RC.C 1-1
#include<stdio.h>
#include<conio.h>
main()
{
int a,b,c;
int m1(int,int);
clrscr();
printf("Enter two numbers:\n");
scanf("%d %d",&a,&b);
c=m1(a,b);
printf("Sum=%d",c);
getch();
}
int m1(int a,int b)
{
return (a+b);
}
:11
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu

```

```

enter two values:
4
5
sum=9_

```

### Nesting of Functions in C:

Some programmers think that a role in another role is called a "Nesting element." But, in reality, it is a lexical spreading feature rather than an embedded job. The lexical C domain is not relevant because the compiler is unable to reach / find the correct location in internal memory.

C is not accepted since a function in C cannot be represented in another feature in the nesting system. We may announce a feature inside a class, but it isn't an in-between feature.

Because surrounding block's local variables cannot be accessed through nested function definitions, only global module variables are accessible. This is achieved to stop a check of global variables through the user. As in C, there are two nesting areas: local and global (and moreover integrated). Nesting properties are often limited.

### Recursion in C:

A chaining process takes place when a feature is simply named. Recursion is a special case that calls a function. A quite well described recursion is as follows:

```
main()
{
Printf("example of recursion\n");
main();
}
```

Such a method can produce a production if it is applied:

```
example of recursion
example of recursion
example of recursion
```

That's an immediate end or it could carry on forever.

Another important definition of recursion is the estimation of variables for a specified number. As seen below, the factorial of a number n is a repeated multiplication series:

Factorial of  $a = a(a-1)(a-2)\dots 1$ .

For example factorial of 5 =  $5 \times 4 \times 3 \times 2 \times 1 = 120$

The following is a method for evaluating the factor n

```
Factorial(int a)
{
  Int fact;
  If(a==1)
  Return(1);
  Else
  Fact=a*factorial(a-1);
  Return(fact);
}
```

See how the recursion functions. Expect a = 3. Expect. Because the priority of a person is not 1, the argument is made.

```
Fact a*factorial (a-1);
It's going to be run with a=3. That is what is happening.
Fact=3*factorial(2);
```

That will be assessed. A factorial call with a=2 is the word to the right. The value below is provided by this request.

```
2*factorial (1)
```

Once more factorial is labelled with a=1. This time the function returns 1. In summary, the operating sequence can:

```
Fact = 3 * factorial (2)
= 3 * 2 * factorial (1)
= 3 * 2 * 1
= 6
```

In order to successively extend the same approach to subsets of the problem, recursive functions may be used to solve problems. If you compose recursive functions, you need an if declaration such that the feature will return without a recursive request being needed. The role would never be performed elsewhere.

**Example Program of Recursion:**

```

File Edit Search Run Compile Debug Project Options Window Help
RECR.C 4(1)
#include <math>\langle math \rangle</math>
#include <math>\langle math \rangle</math>
int fact(int number)
{
    int a;
    if (number==1)
        return 1;
    else
        a=number*fact(number-1);
    return a;
}
void main()
{
    int b;
    clrscr();
    b=fact(4);
    printf("factorial of %d:",b);
    getch();
}
17:26
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu
factorial 24

```

### Passing Arrays to Functions:

If you want to shift one-dimensional array as a claim to a location, you can announce the formal parameter in one of three forms, with identical results for all three declarations since each one states that you would give the whole point to the compiler. Multi-dimensional arrays may also be moved along as structured parameters.

#### Method 1:

Return type function(type name of the array[])

The widely used technique is blank subscription notation[].

#### Method 2:

Return type function(type name of the array[Size of the Array])

In subscript notation[] we can set the size as an option

#### Method 3:

Return type function(type \*name of the array)

The concept of a pointer can also be used. We'll learn about this in the pointer chapter.

### Example Program:



```

File Edit Search Run Compile Debug Project Options Window Help
PRSSINGR.C
#include<stdio.h>
#include<conio.h>
void find(int arr, unsigned int n)
{
    int i;
    for(i=0; i<n; i++)
    {
        if(arr[i]==arr)
        {
            return i;
        }
    }
    return -1;
}
int main()
{
    int arr[]={10, 20, 30, 40, 50, 60, 70, 80};
    unsigned int n =sizeof(arr)/sizeof(arr[0]);
    int arr;
    printf("Enter a number:");
    scanf("%d",&arr);
    int i=find(arr, n);
    if(i!=-1)
    {
        printf("Element found at index %d", i);
    }
    else
    {
        printf("Element not found");
    }
}
3:34
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu
1020094658667808

```

### Searching and Sorting:

One of the most difficult activities is the quest and selection of collections. The list is scanned using two approaches: linear or sequential quest and binary scan. Any array variable is monitored before a match is identified in a linear quest. The following is a sample function that scans the array for a single object and returns the array location if it has not been located or -1 return. The data in the collection is scanned for binary purposes and before the right match is found the core of the contract collection is verified.

### Sorting the Data:

There can be three types of arrays: sorting, sorting and bubble form. The scan, as you can see, requires one loop for each array and each tab, sorting includes looping, and the array is crossed by n-1.

The first place in the list starts with the selection process. In the first run through the sequence, we find the smallest weight. We change the value at the first location of the list with the lowest value, whichever is corresponding to the newly discovered value.

Here is an example of sorting type: when the insertion is written, you first have one object, so you take a new component, and then you order the two items in addition. Then you take a new object and add the three items to each other.

Sorting your hands on the bottom is like arranging a card box. The bigger object, the less, the wider, the greater, the greater and the N-1. You exchange neighbours in a sort of circle.

### Passing Strings to Functions:

Character arrays can be stored in C, but the guidelines for basic array operations are somewhat close.

### Basic Rules:

- ❖ The strings to be moved must be considered a structured statement in order to describe the method.

```
Void display(char item name[])
{
----
----
}
```

The prototype functionality may illustrate that the statement is a string. The prototype can be described above as a function.

```
Void display (char str []):
```

A call method requires a string list name without subscribers as its individual arguments. Panel (names); where the properly defined string sequence is names in the calling method

### The Scope, Visibility and lifetime of variables:

C factors vary as to actions from the ones of most other languages. The importance of the software, for example, persists with the simple application as a whole. That's not really the case in C. All depends on the storage class that a variable will carry.

In addition to providing a data form in C, all variables often have a storage form. The following vector management groups are the most significant features.

**Local Variables**

**Global Variables**

**Static Variables**

**Register Variables**

We may briefly address the scale, functionality and reliability of each of the above variables. The number of variables specifies the software area under which variable can be used. Longevity is that an attribute holds a defined meaning in a software execution to specifically affect its utility. Longevity implies longevity. Visibility requires the accessibility of a memory element.

Depending on the position of their announcement, the factors can also be generally defined internally and externally. In a specified method, external variables are stated and outside a system, external variables are declared. It is essential to understand the definition and utility of the storage groups in order to construct efficient multi-function structures.

**Automatic Variables:**

For-specified variable feature or row, this is the default storage class. Therefore, when writing C language applications, the auto keyword is seldom used. Auto variables may be accessed only inside the specified block / feature and beyond it (defining its range). This will naturally be accessed in nested blocks of the auto variable block / function. But the description of the points given here showing the precise place of the memory in which the variables exist also indicates that they can be located beyond their domain. If declared, a waste attribute is allocated by design.

**External Variables:**

External storage class tells us that the variable is listed somewhere and not in the same used unit. The value in another block is currently delegated and it may be changed / overwritten in another row. Therefore, an external variable is nothing more than a national, lawful value variable loaded for other applications. It can be found in any feature / unit. Moreover, the "remote" keyword in every feature / block prior to statement / definition may be used to construct an additional standard global variable. This simply implies that a new variable is not initialised but just the existing one used / accessed. The key objective is to reach two separate files that are part of wide software by utilising external variables.

**Static Variables:**

This class is used to define static variables that are normally found in C language in software writing. Static variables hold their importance even though they are out of control. Therefore, in the final context, static variables maintain their value. We should then conclude that the programme only operates once and until the completion. Any fresh experiences would be transferred and they are not declared. The access of their specific function is local. External static variables may be accomplished in any section of the method. Null is the 0 value of the programmer.

**Register Variables:**

The registry variables reported with the same output as the auto variable. The only difference is that the programmer wants to store certain variables in the microprocessor register when a free register is available. The register variables are also easier to use than those placed in memory at application runtime. If no free register is open, it is preserved in the memory only. In general, only a few variables reached very frequently in this application are defined by the register keyword that increases the software runtime. One fascinating and significant aspect is that no references can be used to access the register variable key.

**Multi File Programs:**

We also hitherto presumed that all functions are specified in a single file. However, in the real-life programming world more than one file source may be compiled separately and later linked to shape executable object code. It's really good because improvements in a specific file don't impact any files because you have to recompile the entire software.

Multiple source files can share a variable if the variable is considered external. Variables exchanged with two or more files are global variables, but they have to be specified correctly in one file and specifically described in other files with different variables.

**Multiple Choice Questions (MCQ)**

1. A function calling itself is renamed.  
 A) Auto B) Self C) Static D) Recursive

Ans: **D**

2. How many C function values will a time return  
 A) 0 B) 1 C) 2 D) 3

Ans: **B**

3. What are the C language function categories  
 A) Library Functions B) User Defined Functions C) Both D) None

Ans: **C**

4. Function can return structure in C  
 A) True B) False C) Standard D) None

Ans: **A**

5. The default parameter passing mechanism is  
 A) Result B) Call by Reference C) Call by Value d) None

Ans: **C**

**Fill in the Blanks:**

- Block of statements is known as \_\_\_\_\_
- Function can return structure in C \_\_\_\_\_
- All functions are specified in a \_\_\_\_\_.
- The registry variables reported with the same output as the \_\_\_\_\_

**Short Questions:**

- Define Register Variables?
- Define Global Variables?
- Describe Multi File Programs?
- Explain about External Variables?

**Review Questions:**

1. Explain passing array techniques with example?
2. Explain Recursion in C language?

**Programming Exercises:**

1. Write a program to show the simple structure of a function
2. Write a program to swap three numbers using function
3. Write a program to print all perfect numbers in given range
4. Write a program to find cube of any number using function
5. Write a program to check whether a number is even or odd using function.

FOR AUTHOR USE ONLY

## Unit 12

### Structure and Unions

**Main Goals of the Unit:**

It should be complete this unit:

- ❖ I'll demonstrate how a device uses programmes
- ❖ Explain how to control configuration and participant variables
- ❖ Install Applications and arrays
- ❖ Hierarchical processes, architectures and functions are seen
- ❖ Determine how organisations and groups differ in their management methods

**Introduction:**

In addition, data artefacts in the same type including int or float are obtained for arrays. We should not however be able to use a collection of different types of data sets with a common name. Fortunately, the method of packing system of various type C is supported by a system build shape known as structures. A structure is a multifaceted way of processing logically connected posts of data. For example a number of attributes such as employee name, employee identification and compensation are reflected.

Structures appear to be more efficient at organising complex data. It's a positive word we'd still use in planning our programme. This unit discusses the study of frameworks and implementations for software development. There is also a related word, called unions.

**Defining a Structure:**

A structure is a collection of elements in C's programming language with different types of knowledge. The framework is used to construct a user-determined data form in C language programming. A user-defined data form specification is often referred to as a "User-defined data sort" in C.

In other words, a structure is an arrangement of non-homogeneous components. The framework helps one to describe different categories of data known as user-defined data, which involve various values. The description of the formal framework is the following:

Structure is a category of many entity types that act as a user-determined data form in C with the same name.

Structures are commonly used for representing a database of c programming. Structures facilitate mixing data types with category components. The concrete components are called concrete components.

**Memory Allocation of Structure:**

When constructs are used in c programming language, the memory does not allocate structured definitions. The memory is delegated to build a certain configuration attribute. As long as a configuration vector is established no memory can be reserved. The processing capacity assigned to the systemic members is proportional to the cumulative amount of memory available.

### Arrays vs Structures:

Structures and arrays are also recognised as structured data forms because they enable one to view and modify data easily. However, there are a number of related areas:

An array is an array with the same data unit type. The framework may have different product forms. An array is a generalised data form while an array is a particular programmer.

An array is like a built-in data form. What we are going to do is announce and use an entity list. First, in the case of a system previously specified and used variables of this type, we need to build and declare data structures.

### Creating Structure Variables:

To build a c structure we use the keyword "template." We use the following syntax to construct structures in c programming language.

```
Struct <structure name>
{
Data type member1;
Data type member2, member3 ;
----
----
};
```

This is an overview of how a student keeping organisation has been formed.

### Example:

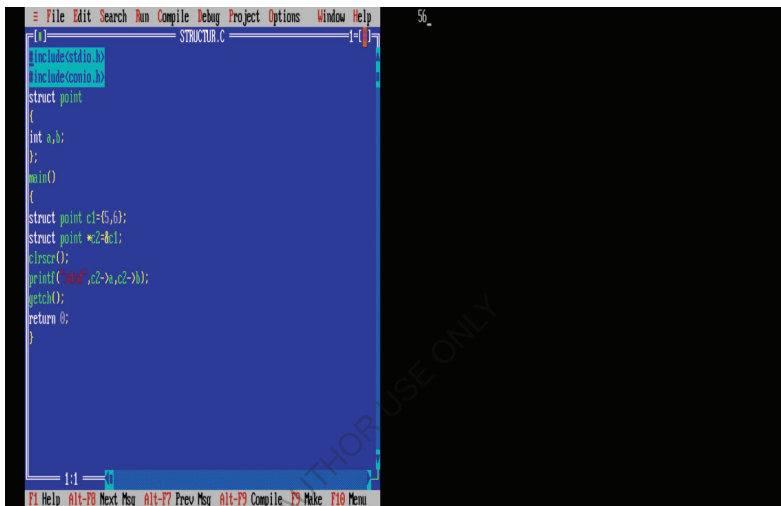
```
struct employee
{
char employee name [20];
int employee id;
float employee salary;
};
```

**Structure variables are created and used:**

In c programming language there are two methods of building structural variables. We can also create a structure variable when describing a structure and use a structure keyword after a structure has been constructed.

We access the structural members through a structure variable by the pointer operator (). And take the following sample code:

#### Structure Example program:



```

1  #include <stdio.h>
2  #include <conio.h>
3  struct point
4  {
5      int a,b;
6  };
7  main()
8  {
9      struct point c1=(5,6);
10     struct point *p2=&c1;
11     clrscr();
12     printf("%d %d\n",c1.a,c1.b);
13     getch();
14     return 0;
15 }

```

#### Type Defined Structures:

The C language programming includes a keyword named typedef, for which you can mark a new form. This is an example of a single message BYTE term description

```
typedef unsigned char BYTE;
```

The BYTE Marker may be used as an abbreviation depending on the definition for the unsigned char type.

```
BYTE a1,a2;
```

By convention letters in the uppercase, the user can note that the style names are merely a symbolic abbreviation.

```
typedef unsigned char byte;
```

You may also name the defined data types of your user with typedef. For e.g, by using a structure typedef you can create a new data type and use it directly to identify variables of structure.



### Typedef Example program:

```

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

typedef struct books {
char title[40];
char author[30];
char subject[50];
int book_id;
}
book;

int main()
{
book book;
clrscr();

strcpy(book.title,"basics of c");
strcpy(book.author,"yogesh");
strcpy(book.subject,"ansi c");
book.book_id=789;

printf("book title:\n",book.title);
printf("book author:\n",book.author);
printf("book subject:\n",book.subject);
printf("book book id:\n",book.book_id);
getch();
return 0;
}

```

### Accessing Structure Members:

The members in the method are able to view and delegate beliefs in a range of ways. As described above, the representatives themselves are not variables. They should be added to the framework variables to allow them usable participants. For starters, the title of the word does not imply the title of the term in book3. A partnership between a participant and a variable is defined with a component operator. Often referred to as dot operator or time operator, for instance.

#### Book1.price

is a vector book price1 and can be interpreted like any other generic value. Here is the worth to be offered to book members1.

### Accessing Structure Member Example Program:

The image shows a screenshot of a C program in a text editor. The code defines a structure named 'car' with fields for wheels, name, color, owner, and number. It then initializes a variable 'a1' with specific values and prints them out. The output on the right side of the image shows the printed values: car no of wheels:4, carname:Maruthi New Zire, carcolor:white, carowner:NagaPrasad, and carnumber:4546.

```

File Edit Search Run Compile Debug Project Options Window Help
=() ASM.C =[]
#include<stdio.h>
#include<conio.h>
struct car {
int carwheels;
char carname[30];
char carcolor[20];
char carowner[20];
int carnumber; }
a1={4,"Maruthi New Zire","white","NagaPrasad",4546};
int main() {
clrscr();
printf("car no of wheels:%d",a1.carwheels);
printf("carname:%s",a1.carname);
printf("carcolor:%s",a1.carcolor);
printf("carowner:%s",a1.carowner);
printf("carnumber:%d",a1.carnumber);
getch();
return 0;
}
1:1
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu
car no of wheels:4
carname:Maruthi New Zire
carcolor:white
carowner:NagaPrasad
carnumber:4546

```

### Structure Initialization:

The memory for the uninitialized variable is not reserved when defining a configuration. We may start structure variable in various ways, let us speak about a familiar structure example.

#### Method 1:

##### Struct employee

```

{
Char employee_name[30];
Int id;
Float salary;
}
Emp1={"nayan",75,87,98};

```

The definition has been defined in the above snippet and the variable after declaration has been initialised.

```
Emp1={"nayan",75,87,98};
```

This is the code that is used in C programming to launch the variable structure

**Method 2:****Struct employee**

```
{
Char employee_name[30];
Int id;
Float salary;
}
Emp1={"nayan",75,87,98};
Emp2={"varshini",85,97,96};
```

In this case, two structure variables were mentioned above code. We initialised two variables after the variable statement.

```
Emp1={"nayan",75,87,98};
Emp2={"varshini",85,97,96};
```

**Method 3:****Struct item**

```
{
Int price1;
Int price2;
Int price3;
} price1={95};
```

Although three structural components remain, only one is initialised, although two others are initialised by none. If there are other variables of the data form, their first values are

Data Type	Default value if not initialized
Integer	0
float	0.00
Char	Null

The system memory is not reserved if we designate a configuration. i.e., until the sentence below is written, we must start a machine variable to add storage to the device.

**Copying and Comparing structure variables:**

Two variables may be replicated as normal in the same structure type. The foregoing definitions shall occur if Object1 and Object2 belong to the same class.

```
Object1=object2;
```

```
Object2=object1;
```

In the event of structural variables contrasts, we can evaluate human participants. C doesn't include logic functions of structure variables.

```
Object1==object2;
```


```
Object1!=object2;
```

Statement is not authorised.

### Operations on Individual Members:

Here you can figure out how structural variables should work. A systemic operation will basically be done here. It shall be allocated to the task. Other stacking procedures, such as equivalent controls or other, are not eligible.

### Operations on Individual Member Example Program:



```

1  File Edit Search Run Compile Debug Project Options Window Help
2  OPERATIO.C
3  #include<stdio.h>
4  #include<conio.h>
5  struct point
6  {
7  int p;
8  int q;
9  };
10 int main()
11 {
12 struct point a1=(100,200);
13 struct point a2={1;
14 };
15 printf("a1.p=%d,a1.q=%d\n",a1.p,a1.q);
16 getch();
17 getch();
18 return 0;
19 }
20
21 Help Alt-P9 Next Msg Alt-F7 Prev Msg Alt-P9 Compile F9 Make F10 Menu
a2.p=100,a2.q=200

```

### Arrays of Structures:

Structures in C Programming are helpful in structurally arranging the data for grouping various categories of data. Arrays are used to obtain the same data form values. We will provide you a realistic description in this article of the spectrum of structures in the C definition.

For e.g. we store information about the employee, such as name, Email, age, address and salary. We normally group them as an employee organisation with the above participants. The framework variable may be generated to view or change the framework members. An organisation with ten or hundred employees, how are 100 employees? In C-Programming the above issue can be conveniently overcome by the combination of two large C structural arrays. We are trying to create our employees' system. Then, instead of generating the list, we build the layout attribute.

### Declaring C Array of Structures:

Let me announce a series of configurations at the beginning of the method in C

```
Struct student
{
Int id;
Char name[30];
Int marks;
}
Students[2]={{01,"Mahesh",76},{02,"Nare sh",87}};
```

The student framework is structured to store student information such as age, name and identity markings. We only generated a number of student variable structures [2] (size2) throughout the statement process. The values for both students were initialised by each member of the system.

#### Declaration of C array of main() function structures:

```
Struct student
{
Int id;
Char name[30];
Int marks;
};
```

```
Create the student structure variable within the main () function struct Student Students[2];
Students[2] = {{25, 'Suresh', 25000}, {24, 'Tutorial', 28000}};
```

### Arrays within Structures:

- ❖ A configuration may be as ordinary as int float members. Members of list
- ❖ We should define an array if we wish to store several values inside the framework.
- ❖ The arrangement can include as many array members as we need.
- ❖ Syntax for defining an array inside a framework is no different from the traditional syntax. The only difference is the declared interior structure.
- ❖ Imagine the following example to store student records, in which we have a set of marks that enable us to store and show marks of six subjects on the computer.
- ❖ The looping framework must be used because we have to store marks for six subjects, but does not refer to a single sequence of dimensional characters.

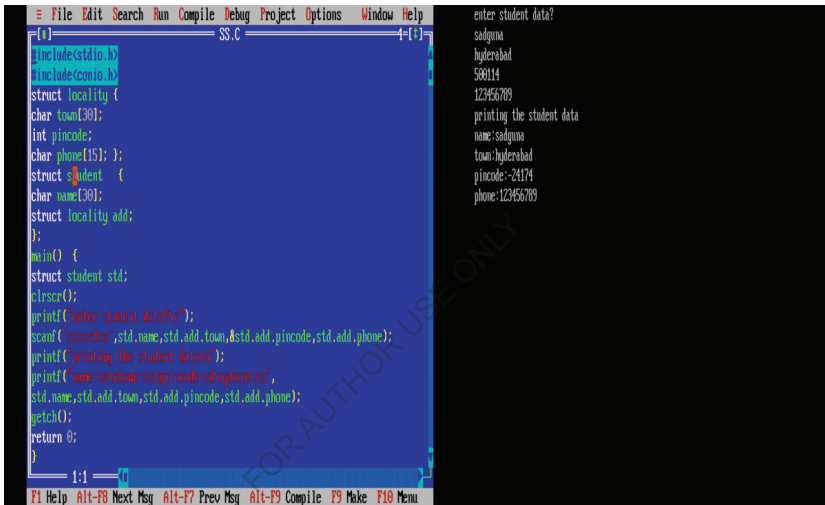
### Purpose of an array within Structure:

You may select a table within the framework where the string value is held. The List is also capable to save the same data sort as the name goes under the object form alone.

### Structure within Structures:

C offers us the challenge of nesting an individual framework inside a collective framework that produces complex data forms. For e.g, we would also need to store the address of a person in a database. You might have sub sections such as lane, location, and status and pin code for the address of the attribute. To register the address of the employee, the employee 's address must be placed in a different database and the company address incorporated into the employee's file. Take into account the following strategy.

### Structure within Structures Example Program:



```

File Edit Search Run Compile Debug Project Options Window Help
(1) SS.C +(-)
#include<stdio.h>
#include<conio.h>
struct locality {
char town[30];
int pincode;
char phone[15]; };
struct student {
char name[30];
struct locality add;
};
main() {
struct student std;
clrscr();
printf("enter student data\n");
scanf("%s%s%d",std.name,&std.add.town,&std.add.pincode,&std.add.phone);
printf("printing the student data\n");
printf("name:%s\ttown:%s\tpincode:%d",
std.name,&std.add.town,&std.add.pincode,&std.add.phone);
getch();
return 0;
}
1:1
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu
enter student data?
sadguna
hyderabad
500114
123456789
printing the student data
name:sadguna
town:hyderabad
pincode:-24174
phone:123456789

```

This can be done to build the arrangement.

### By Separate Structure

In multiple systems, two structures are formed and the capture is that the dependent framework needs to be worked as a component of the main structure.

### By Embedded Structure

For the architecture within the framework, the built-in layout in C is used. This method decreases the total amount of lines of code but for a variety of data structures it is not feasible to use it.

### Structures and Functions:

The C programming facilitates the propagation of frameworks as feature parameters. Please refer to the functions of the C article before reading this post. You may understand the meaning of the function.

Three functions may be translated to the structures C:

Transfer each entity as a system function argument. It is equivalent to the transfer of universal principles as claims. And if it's trivial to introduce, we are not complaining with a system's size becoming a little greater. Five Consider as a product in the whole structure.

The framework address can even be modified (pass by reference).

The explanations below demonstrate how processes operate in C through meaning and contrast.

Drag and drop the framework to C meaning.

The modifications made to the variable members of the system inside the feature would not represent the original members as the attribute is moved on to another class.

Structures and functions are needed for this C, consumer, student name, first year mark and programme of second year marks. This curriculum explores whether the recipient is entitled to a scholarship by the application of certain principles.

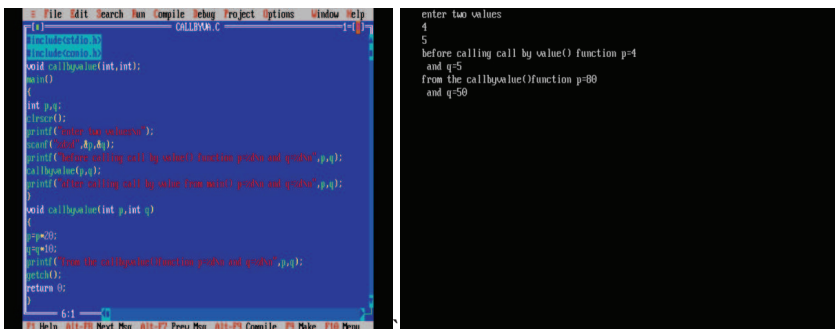
For the distinction between the passes and the relative transfers, see Call by Value and Call by relation. We would address the second and third methods here.

### Call by Value:

Call for C is the most convenient way to access functions. The values of the variables listed were transferred as parameters via this process. If the words are conveyed.

The converter actually renders the original value copy or replicate and transfers the replica values on to the feature rather than the original values. Therefore, the original values cannot be interpreted by modifications made to variable values within a function. Take an illustration to grasp the C sense call better the consumer will insert 2 integer values for this application. Then, we move these values to a separate feature for certain measurements through value method with a call C.

### Call By Value Example Program:



```

1 int main()
2 {
3     int a=4;
4     int b=5;
5     int sum=callbyvalue(a,b);
6     printf("Sum of a and b is %d",sum);
7     return 0;
8 }

```

```

enter two values
4
5
before calling call by value() function p=4
and q=5
from the callbyvalue()function p=88
and q=50

```

### Call by Reference:

In this C request, the reference method in the functional parameters transmitted the address of the defined variables. In addition, the feature accesses the address of the initial value when the vector address is passed to the application (domain reference). Any adjustment in the variables values in the equation would result in modifying the original values. Please be cautious while you deal with this type. You will end up with inaccurate values because all calculations are focused on the original values.

### Call By Reference Example program:

```

1 | #include <stdio.h>
2 | #include <conio.h>
3 | void swap(int *a1,int *a2);
4 | main()
5 | {
6 |     int a1=10,a2=14;
7 |     swap(&a1,&a2);
8 |     printf("a1=%d\n",a1);
9 |     printf("a2=%d\n",a2);
10 |    return 0;
11 | }
12 | void swap (int* a1,int* a2)
13 | {
14 |     int t;
15 |     *a1=*a2;
16 |     *a2=*a1;
17 |     *a1=t;
18 |     getch();
19 | }

```

Output:

```

a1=14
a2=10

```

### Unions:

A union is a certain type of C data which can be processed for different forms in the same memory location. A union can be created comprising of many participants, but only one individual may carry a value at any time. The Syndicates are an easy way to utilise the same shop venue for different purposes.

You would use the syndicate like you used to describe a syndicate form. The specification describes a new data form of more than one component of the software. The following form of union agreement.



```
Union [union tag]
{
Member def;
Member def;
----
Member def;
}
[one or more union variables];
```

The meaning of any component, whether int I or float f, or any variable summary, is an optional union tag. One or more union variables may be defined, but optional, before the final semicolon at the end of the union description. This is how a syndicate type called Data with three I f and str members is represented.

```
Union method
{
Int a;
Float b;
Char str[30];
}
Method;
```

In a variable data sort, you can save an integer, floating point number or character set. This ensures you can store different forms of data in the same storage location with one attribute. You may use the built-in or defined user data type in syndication, based on your particular needs.

The conscience of the Community must remain strong enough to maintain the greatest leader of the Community. For e.g, data type is the maximum field that a string will occupy. The data type is around 30 bytes of memory space. The following illustration indicates the overall union buffer space.

**Union Example program:**

```

File Edit Search Run Compile Debug Project Options Window Help
UNION.C 1-0
#include <stdio.h>
#include <conio.h>
#include <string.h>
union method
{
int a;
float b;
char str[30];
};
int main()
{
union method method;
clrscr();
printf("data occupied by memory size is %d",sizeof(method));
getch();
return 0;
}
1:1
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu
data occupied by memory size:30

```

### Accessing Union Members:

To contact every union delegate, we use the Worker Control Operator (). The control operator for a Worker is identified with the name of the group and the union member we want to reach. Membership operator the keyword union helps to describe variables of the union form. The following example illustrates how to use unions in one software.

### Accessing Union Members Example Program:

```

File Edit Search Run Compile Debug Project Options Window Help
UNION.C 1-0
#include <stdio.h>
#include <conio.h>
#include <string.h>
union method {
int a;
float b;
char str[100];
};
int main()
{
union method method;
method.a=20;
method.b=30.5;
strcpy(method.str,"programming language is c");
clrscr();
printf("method.a is %d",method.a);
printf("method.b is %f",method.b);
printf("method.str is %s",method.str);
getch();
return 0;
}
4:14
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu
method.a:2026
method.b:1139756508.0078036000000000000000
method.str:programming language is c

```

### Size of Structures:

In conjunction with each other, the machine gathers various kinds of info. The scale of the system is calculated in numerous ways:

**Various methods of structural scale estimation:****By Observation****By using sizeof Operator****Without using sizeof Operator**

- ❖ Connect each device component to a particular scale.
- ❖ Space Equivalent (No of \* Size(char) elements) is needed for the array of characters
- ❖ The Integer needs 4.2 bytes for the Borland C++ 3.0.
- ❖ The size of the operator is the same, but the library feature is non-scale for calculating data type, vector, express output. Sizeof is an attribute of size. The scale of C User.
- ❖ We may calculate the structure size without using the size operator. In order to identify structural proportions, we should differentiate them.

**Bit Fields:**

The scale of the frame structure or union may be shown in C. You easily utilise the memory if you realise that you don't have a certain cap or that there are fields beyond the target set.

Take the circumstance of the C-program, which has number TRUE / FALSE variables with a hierarchical type name: TRUE / FALSE:

```
Struct
```

```
{  
Unsigned int haveclass;  
Unsigned int haveroom;  
}
```

```
Decided;
```

The configuration above includes 8-byte address space so you must hold either 0 or 1 for each of the above variables. The C-language offers a better way of using room in this setting.

If the variables are defined in the form of a specification, the vector width is expressly defined such that the programmer requires only such bytes. You will compose the following layout code for:

```
Struct
```

```
{  
Unsigned int haveclass: 1;  
Unsigned int haveroom: 1;  
}
```

```
Decided;
```

The above snippet code (structure programme) occupies 4 bytes of the state variable's memory space. In this case, the configuration took 4 bytes of memory for a certain variable but only 2 bits were used to save the values.

### Declaring Bit Fields:

Bit fields are referred to as variables specified by the default width or height. This bit field can interrupt more than one bit. The specification and syntax for bit field statements in a structure is as follows:

```
Struct
{
Data type[nameofmember]: width of bit field;
};
```

### Data Type:

Establish the data structure representing the bit field meaning representation and understanding. The form of data may be easy, signed or unsigned.

### Name of member:

Place the name inside the bit field member background

### width of bit field:

In the appropriate bit-field, insert the number of bits. The bit field width should be less than or at least equal to the defined type dimension.

### Multiple Choice Questions (MCQ)

1. A \_\_\_\_\_ is a collection of elements in C's programming language with different types of knowledge.  
A) String B) Array C) Union D) Structure

Ans: **D**

2. The \_\_\_\_\_ is used to construct a user-determined data form in C language programming.  
A) String B) framework C) Array D) None

Ans: **B**

3. The \_\_\_\_\_ is delegated to build a certain configuration attribute.  
A) Library Functions B) User Defined Functions C) Memory D) None

Ans: **C**

4. An \_\_\_\_\_ is an array with the same data unit type.  
A) Array B) String C) Standard D) None

Ans: A

5. The C language programming includes a keyword named \_\_\_\_\_

- A) Result B) Call by Reference C) typedef d) None

Ans: C

**Fill in the Blanks:**

1. The scale of the frame structure or \_\_\_\_\_ may be shown in C.
2. A \_\_\_\_\_ is a certain type of C data which can be processed for different forms in the same memory location.
3. The \_\_\_\_\_ may be easy, signed or unsigned.
4. The C programming facilitates the propagation of frameworks as feature parameters.
5. \_\_\_\_\_ the keyword union helps to describe variables of the union form.

**Short Questions:**

1. Define Call by Value?
2. Define Call by Reference?
3. Describe Bit Fields?
4. Explain about Unions?
5. Define Structure?

**Review Questions:**

1. Define Structure with example?
2. How to create a structure Variable with example?

**Programming Exercises:**

1. Write a program to how to create union variable
2. Write a program to difference between union and Structure
3. Write a program to accessing union members
4. Write a program to Call By Reference Example program
5. Write a program to Call By value Example program

## Unit 13

### Pointers in C

**Main Goals of the Unit:**

It should be complete this unit:

- ❖ Know the idea of pointers
- ❖ Determine how pointer variables in a program are used.
- ❖ Describe the pointer chain.
- ❖ Expressions Illustrate
- ❖ Think about points and collection
- ❖ Explain how pointers for functions and structures are used

**Introduction:**

A pointer is a C-derived data type. It is built of one of the basic types of C data. Remember that their value is in pointers. Remember. Since these memory addresses are locations in the memory of the computer that store program instructions and data, indicators can be used for memory-based data access and manipulation.

C's most distinctive and exciting features are without doubt pointers. The strength and versatility of the language have been increased. They seem awkward and difficult for a beginner to grasp, but they are a powerful tool to use when mastered.

Points are used mostly in C because they have a range of advantages for programmers.

Incorporated:

The argument function points can return multiple values from a function.

Pointers enable function references and therefore make it possible to transmit functions as arguments to other functions

Using arrays of pointer to save memory space in data storage.

The Pointers require the dynamic memory to be handled by C

Pointers provide an important way to handle complex data structures such as tables, linked lists, columns, stacks and trees.

The true power of C is, of course, to correctly use the pointers. The measures and how they can be used when developing programming are discussed in detail in this chapter.

**Understanding Pointers:**

The memory of the machine is a sequential storage cell array. A number called the address connected to each cell is generally known as the byte. Addresses are usually numbered from 0 upwards consecutively. The memory size depends on the last address. The latter address will be 65,535 for a computer device with 64 K memory.

Once a variable is declared, the program assigns the correct variable value to somewhere in the memory. That byte has a single address number and will have its own address number at this spot. Take the following point into consideration.

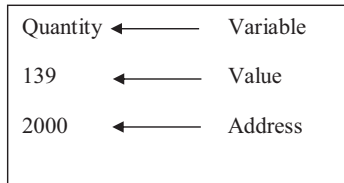


Figure: Representation of a variable

Int quantity – 139;

This statement instructs the program to locate an integer quantity position and sets the value 139. Suppose the program had selected for quantity the address location 2000. As shown in Figure 13.0, we can describe this. The framework also combines the number of names with the address 2000 during the execution of the program. By either using the name quantity or the address 2000, we can have access to the value 139. Since memory addresses simply are numbers, certain variables which can be saved like every other variable can be allocated to the memory. These variables that are called pointer variables that have memory addresses. Therefore, the pointer variable is nothing but an address-containing variable that is a position of another memory variable.

Recall that since a variable is a pointer its value is saved to another location also in the memory. Suppose we assign a quantity address for the variable p. A relation can be shown between the variables p and quantity.

### Underlying Concepts of Pointers:

The three basic principles are based on the criteria.

Memory addresses are called constants within the device. We can't alter them. Only data values can be processed using them. Like house numbers, they're. The memory value cannot be saved directly by us. Only the address operator (&) helps us to obtain the value through the variables stored in it.

The resulting value is referred to as the pointer value. From one program execution to another the value of the pointer may change.

The value can be stored in another variable until we have a pointer value. Pointer variable is referred to as the variable that holds a reference value.

The exact location of a memory variable is system-dependent, which means that we do not know the address of a variable straight away. How then will we evaluate the variable's address using the operator & accessible in C? In the scanf function, we saw the use of this address operator. The operator returns the corresponding variable's address immediately preceding a variable.

Only a single variable or array element can be used by the & operator. Illegal use of address operator is the following.

### **Pointers in C:**

We use standard variables in the c programming language to store user data values. When a variable is declared, the compiler assigns the appropriate name to the memory. Each variable has a name, data type, value, class and address in the c programming language. In order to store the address of another variable with the same data type we use a different kind called a pointer. The next move is a pointer.

Pointer is a variable type used to store a variable's memory location address.

We can create pointer variables for any data type in the c programming language. Every pointer saves the variable address with just the same data type. This means that the integer pointer is used to store the integer variable address only.

### **Accessing the Address of Variables:**

We use reference operator "&" in c programming language to control variable address. For instance, we use "& marks" for accessing the address of a variable. We use the print statement below to show the variable 'marks' memory location address.

### **Example Code:**

```
Printf("Address:%u", & marks);
```

In the example above %u is used to display the variable mark address. Any location address is an unsigned integer number.

### **Declaring Pointers:**

For c, the Pointer Variable declaration is identical to a standard variable but the name is prefixed by \*. The following syntax is used to define a variable pointer.

```
Data type * Pointer Name;
```

An \* prefixed declaration attribute is an attribute of the pointer.

### **Example Code:**

```
Int *ptr;
```

The variable 'ptr' is a pointer that can be used to save a whole variable address in the above example declaration.



**Assigning Address to Pointer:**

We use the following syntax to assign the address to the pointer variable.

Pointer Variable Name = &Variable Name;

For example, consider the following variables declaration...

```
Int x, *ptr;
```

The variable "x" is the variable "natural integer" and "ptr" is the integer point variable in the above paragraph. We are using the following sentence if we want to add "x" attribute to the "ptr" attribute pointer.

**Example Code:**

```
Ptr = &x;
```

The variable "x" is assigned to the pointer variable "prt," in this paragraph. We say here that the pointer variable ptr is x.

**Accessing Variable Value Using Pointer:**

Pointer variables are used to store other variables addresses. This address can be used to access the variable's value using its pointer. In front of the variable name we use the symbol "\*" in order to reach the variable value that the indicator refers to. We use the general syntax below:

```
*Pointer Variable Name
```

**Memory Allocation of Pointer Variables:**

Every variable of the pointer is used to save another variable's address. An unsigned integer value is contained in the computer memory address of any memory location. Unsigned integer requires 2 bytes of memory in the c programming language. So any pointer variable with 2 bytes of memory is allocated, regardless of pointer data type.

**Chain of Pointers:**

A pointer is made to point another pointer and a chain of pointers is created thereby. For example, the pointer variable 'ptr2' contains the address of the 'ptr1' variable pointer, indicating where the desired value is located. Multiple indirections are known as these.

A variable that is a pointer must be declared in front of a name with additional indirect operator symbols. The statement 'int\*\*ptr2' says that 'ptr2' implies a pointer to an int-type reference. Remember that the 'ptr2' pointer is not an integer pointer, but an integer pointer. Through applying the indirection operator twice, we can reach the destination value indirectly indicated by a point.

**Pointer Expression:**

Usually, expressions with pointers comply with the same rules as other words. This section discusses some specific aspects of pointer words, such as jobs, transformations and arithmetic.

### **Pointer Arithmetic Operations in C:**

Pointer variables are used to save variable addresses. Every variable's address is an unsigned integer, i.e. a numeric value. Thus, arithmetic operations can be performed on the lines. So if we perform arithmetic operations on the vector pointing, it depends on the quantity of memory the pointer points to.

The following arithmetic operations can be performed on pointers in c programming language.

- 1) Addition
- 2) Subtraction
- 3) Increment
- 4) Decrement
- 5) Comparison

### **Pointer Increments and Scale Factor:**

We saw that the measures can be increased as

**P1=p2+2;**

**P1=p1+1;**

And so on. And so forth. So on. Yet mind a word like that.

**P1++;**

The pointer p1 is made to point to its next form value. If, for example, p1 is an integer pointer of an original, say 2800, then the value of p1 would be 2804 after operation  $p1 = p1 + 1$ , and not 2801. That's because we add a pointer to increase its value by the length of the data type, it is presumed that int is 4 bytes in a 32 bit machine. The scale factor is called this range.

Diverse data types are the following lengths for a 32-bit computer:

Char

Int

Float

Long int

Double

Depending on the device, the amount of bytes used for storing various data types can be identified by using the operating size. For example, if the variable is X, the number of bytes

available for the variable would be `sizeof(x)`. 32 bit systems use 4 bytes for integer storage and 2 bytes for short integer storage.

### Rules of pointer Operations:

When carrying out operations with pointer variables, the following rules apply.

- ❖ A variable pointer may be given another variable's address
- ❖ The values of another pointer variable can be assigned
- ❖ A null or zero value pointer variable can be initialized
- ❖ A vector of pointers with increment or decrement operators may be set pre set or post fixed.
- ❖ The integer value of the variable pointer can be added or removed
- ❖ A single pointer variable can be excluded from another if two pointers point to the same list.
- ❖ The objects of the same data type may be contrasted by relational operators if two pointers are indicated.
- ❖ Cannot multiply a constant by a pointer variable
- ❖ Unable to add two pointer variables
- ❖ An arbitrary address cannot be assigned a value.

### Pointer and Arrays:

The compiler must assign a specific address and sufficient storage quantity to accommodate all the array items in continual memory locations while the array is being declared. The first item of the array is located at the base address. The compiler also describes the name of the array as the first element's constant level. Assume that we announce the following array:

```
Int a[5]={10,20,30,40,50};
```

Assume an address of 1000, with the five elements saved as follows considering that each integer requires four bytes:

Name `a` is a constant point to the first variable, `a[0]` and the value of `a` is then 1000, where `a[0]` is kept.

```
a=&a[0]=1000
```

If we consider `p` to be an integer pointer, we would be able to add the `p` pointer to an array through the assignment following.

```
P=a;
```

It is equal to

```
P=&a[0];
```

In order to switch from one element to another, you can now access any value from a `p++`.

This illustrates the connexion between `p` and `a`:

```
P=&a[0](=1000)
```

```
P+1=&a[1](=1004)
```

```
P+2=&a[2](=1008)
```

```
P+3=&a[3](=1012)
```

```
P+4=&a[4](=1016)
```

### **Pointers and Character Strings:**

We saw in unit 9 that strings are viewed as arrays and hence declared and initialised as follows:

```
Char str[5]="NAGA";
```

The null character `\0` at the end of the string is immediately added by the programmer. C provides an alternate approach for string formation using char style pointer variables. For example:

```
Char * str = "NAGA";
```

This produces a string for the literal and stores it in the vector `str` of the pointer.

The `str` shows the first character of the "NAGA" series as:

### **Arrays of Pointers:**

A list of indicators is an indexed set of variables in computer programming, where variables are pointers (referencing a position in a memory).

Points are an essential computer science technique in order to build, use and break data structures of all sorts. For the same purpose, a list of pointers benefits and any list is useful: it enables a wide number of variables to be numerically indexed.

Below is a series of C pointers, which refer to an integer in a separate sequence in each line. By dereferencing the points the value of each integer is written. In other terms, this code shows the meaning in memory of the stage.

### **Pointers as Function Arguments:**

Pointer is used to carry addresses of arguments passed during function call as a function parameter. It is also known as referral calling. If a function is called by reference the original variable would be influenced by any changes made to the reference variable.

### **Functions Returning Pointers:**

Even a function can return the calling function to a pointer. You need to be careful in this situation, since local function variables don't live outside the system. They only have scope

within the feature. But if you return a pointer linked to a local variable, when the function ends, the pointer will point to nothing.

### Pointers to Functions:

A pointer pointing to a function can be declared, which can then be used as argument in another function. It declares a pointer to a function as follows:

```
type (*pointer-name) (parameter);
```

### Example:

```
int (*sum) ();
```

```
int *sum();
```

When the name of that function is assigned a function pointer will point to a specific function.

### Pointers and Structures:

We often have a set of configuration factors, including the list of integers, a set of pointers, etc. We can use structure type indicators to use the collection of structure variables effectively. We may also have a single structure variable, but it is often used for the collection of structural variables.

### Multiple Choice Questions (MCQ)

1. A \_\_\_\_\_ is a C derived data type.  
A) String B) Array C) Union D) Pointer

Ans: D

2. The memory of the machine is a \_\_\_\_\_ cell array.  
A) String B) Sequential Storage C) Array D) None

Ans: B

3. Memory addresses are called \_\_\_\_\_ within the device.  
A) Library Functions B) User Defined Functions C) Constants D) None

Ans: C

4. \_\_\_\_\_ Variables are used to save variable addresses.  
A) Pointer B) String C) Standard D) None

Ans: A

5. A pointer pointing to a function can be declared, which can then be used as argument in \_\_\_\_\_  
A) Result B) String C) Another Function. d) None

Ans: C

**Fill in the Blanks:**

1. C programming language to store user \_\_\_\_\_
2. \_\_\_\_\_ is a variable type used to store a variable's memory location address.
3. Every pointer saves the variable address with just the \_\_\_\_\_
4. Pointer variables are used to store other \_\_\_\_\_
5. An \_\_\_\_\_ is contained in the computer memory address of any memory location.

**Short Questions:**

1. Define Pointer?
2. Define Pointers to Functions?
3. Describe Array of Functions?
4. Explain Array of Pointers?
5. Define Pointer Expression?

**Review Questions:**

1. Rules of Pointer Operations?
2. Pointer Arithmetic Operations in C

**Programming Exercises:**

1. Write a program to show the basic declaration of pointer
2. Write a program to demonstrate how to handle the pointers
3. Write a program to add three numbers using pointers.
4. Write a program to find the maximum number between two numbers using a pointer.
5. Write a program to sort an array using Pointer.

## Unit 14

### File Management in C Language

#### Main Objectives of the Unit:

It should be complete this unit:

- ❖ Explain the opening and closing of files think of the data.
- ❖ Application operations to data
- ❖ Decide how errors are handled during input and output processes
- ❖ Links to random files explanted
- ❖ Know the Command Line arguments

#### Introduction:

We used to read and write details with scanning and printing features. These are consoles dependent input / output functions that often use the terminal. This works fine when the data is small. However, many real life issues require vast quantities of data, and the input / output console activities in these cases are two major problems.

- ❖ Time is required to handle large quantities of data through terminals.
- ❖ All data is lost when the programme finishes and the machine is turned off.

There is therefore a need for a more versatile approach where data is stored and read without losing data when necessary. The definition of data storage is used in this approach. A file is a disc location with a group of related data. C supports a variety of features like most other languages, including: Simple File Operations.

Name of the File

Opening of a File

Reading the from the File

Writing the from the File

Closing the File

There are two different types of the C file method. First of all, poor rate of I / O, and calls from UNIX. The second solution is called high-level I / O operations which utilises the C basic ad Output library software. This unit explores the critical file administration functionality in the C collection.

#### Defining and Opening a File:

When we try to store data in a secondary storage file, it is important to suggest to the operating system some details regarding the file. The following are included:

We have to grasp the simple file definition in C System, File Forms, before we open the file. If we want a message shown on the file screen, then in read mode we need to open it.

We have to grasp the simple file definition in C System, File Forms, before we open the file. If we want a message shown on the file screen, then in read mode we need to open it.

**File Name**

**Data Structure**

**Purpose**

Very first task in File handling is to open file

**File Name:**

Two fields are the File Name. Field first is field term and field second is field extension. Optional extension field. The name and length of both files are distinguished by time or mark.

**Data Structure:**

In the library of standard I / O functions the file structure of the data is specified as FILE. In short, the form variable FILE must be declared.

**Purpose:**

In C Programming we can open the file in different modes, for example, read, write, and connect, based on the intent of the file.

Opening Mode	Purpose	Previous Data
Reading	File is accessed for read purposes only	Retained
Writing	File is accessed only for the purposes of writing	Flushed
Appending	File is accessed to connect everything to the list	Retained

**Closing a File:**

When all activities have been done, a file has to be locked. This guarantees that unresolved file details is deleted from buffers and that all file connections are broken. An unintended abuse of the file is therefore avoided. If the amount of files to be held open concurrently is limited, it may help to open the appropriate files by closing unnecessary files. An additional instance of closure is where we use a new mode to restart the same file. A method to use this is provided by the I / O library. The following structure takes:

`int fclose (file *fp);`

The tab shared with the tab pointer \* fp will be locked.



```
#include <stdio.h>
#include <conio.h>
int main ()
{
FILE *fp;
fp = fopen("file.txt", "a");
fprintf(fp, "%s", "This is C Book");
fclose(fp);
return(0);
}
```

Let us create a file file.txt, compile the above programme, and then enter the text line and then close the file with the fclose () function.

### Output:

**This is C Book**

### Input / Output Operations on Files:

The following types of I / O file operations are categorised:

**Character Input / Output Operations**

**Integer Input / Output Operations**

**String Input / Output Operations**

**Formatted Input / Output Operations**

**Block Read / Write Input / Output Operations**

### Character Input / Output Operation:

#### fputc():

This method is used to write in the place of the current file and then raise the file pointer. If efficient, it returns an integer reflecting the written character and returns EOF on mistake.

**int fputc(int c, file \*fp)**

#### fgetc():

This method reads a file character and raises the location of the file pointer. When good, it gives an integer representing the written character and returns EOF on mistake.

**int fgetc(FILE \*fp):**

#### Getc () and putc ():

The functions done with getc () and put () are similar to the operations conducted with fgetc () and fputc (), the exception being fgetc () and fputc ().

### Integer Input / Output Operations:

#### Putw ():

This function writes to a file indicated by `fp` an integer number. When an mistake happens, it recovers the integer inserted into the register.

```
int putw(int value, FILE *fp)
```

### **Getw ():**

The value of the whole file connected with the point file is returned by this feature. It will return integer on progress and EOF on error from the input code.

```
int getw(FILE *fp);
```

### **String I/O:**

#### **fputs()**

```
int fputs(const char *str, FILE *fp);
```

This feature writes the terminated null string to a register. This null character which marks the end is not inserted in the register. As efficient, the last written character and EOF will be returned on mistake.

#### **fgets():**

```
char *fgets(char *s,int n, FILE *fp);
```

This feature is used to interpret the file's characters and these characters are contained in the `s` series. It reads `n-1` in the file where `n` is the second statement. This feature returns a string indicated by on progress, and the NULL on EOF. `FP` is a file indication that point to files from which a character is read.

### **Formatted I/O:**

#### **fprintf():**

This function is identical to that of `printf ()`, but instead of the default output it enters formatted data into the file. This function has the same parameter as `printf ()`, but a more parameter is a FILE form reference. It returns the output of characters to the success file and error to EOF.

```
fprintf(FILE *fp, const char *format[,argument,...]);
```

#### **fscanf()**

It's the same feature as `scanf ()`, except instead of the normal display, it reads data from the disc. The parameter of this feature is a FILE form reference. It returns a set of statements that have been given those performance values and EOF error values.

```
Fscanf(FILE *fp,const char *format[,address,...]);
```

### **Block read / Write:**

**fwrite()**

Used to compose a full block in the paper.

```
size_t fwrite(const void *p, size_t size, size_t n, FILE *fp);  
size_t is defined in stdio.h as  
typedef unsigned int size_t;
```

p - It means the memory block containing the data to be loaded into the register.

size - The length of each item is indicated in bytes

n - The amount of objects to be included in the file

fp - The device on which the data is transferred is a Device stage.

**fread()**

This feature reads a block from a specified file in its entirety.

```
size_t fread(void *p, size_t size, size_t n, FILE *fp);
```

Here,

p - This is a pointer representing the memory block processing the data and reading from the register.

Size - object is in bytes in the duration

n - That is the amount of objects from the file to be interpreted

fp - It is the device reference to the device that reads the data. When successful, it reads n papers and returns n, when a mistake or EOF happens, a value less than n is retrieved. This state can be verified with `error ()` or `feof ()`.

**Error Handling During I/O Operations:**

C programming thus does not explicitly help error management; however, it allows you lower access in the form of return values since it is a machine programming language. In case of some mistake, most C or even Unix functions call back -1 or NULL and set error code. It is known as a global variable and shows an error during a function request. In the < error.h > header section, you can find different error codes.

A C programmer will then verify the return values and behave according to the return value. The `errno` to 0 at the moment the software is begun is a safe idea. There is no fault on the software with a value of 0.

The programming language for C gives the `error ()` (and `strerror ()`) functions to view the `errno`-related text message.

The `error ()` function reveals the string that you transfer to the stack, a space, then the text of the current `errno` value.

The `strerror ()` method, which returns a reference to the current `errno` value.

Try to replicate the mistake and attempt to access a non-existent computer. I use both options here to demonstrate the need; however one of more methods to print your mistakes may be included. Second critically, you can use the stream of stderr file to render all the errors possible.

### **Random Access to Files:**

If we wish to view a certain database, there is no need to read any document sequentially. C supports certain random access file processing functions.

#### **Fseek()**

#### **Ftell()**

#### **Rewind()**

#### **Fseek():**

This method is used to check for the location of the pointer in the byte format.

#### **Syntax:**

```
fseek( file pointer, displacement, pointer position);
```

**File Pointer:** This is the key to the register.

**Displacement:** The amount of bytes that are recovered from the current location is either positive or negative. It is associated with L since it's an integer length.

**Pointer Position:** Sets the location of the pointer in the paper.

#### **Ftell ():**

The present pointer location in a file is returned by this feature. The variable value is counted from the start of the list.

#### **Syntax:**

```
Ftell(fptr);
```

#### **Rewind ():**

This role transfers the file pointer to the start of the specified section.

#### **Syntax:**

```
Rewind(fptr);
```

### **Command Line Arguments:**

Point for the command line is an activated function argument. The statement of the command line is an essential C programming principle. It is often used when the software has to be managed from outside. The main () function is moved on to the command line arguments.

#### **Syntax:**

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

Argc is a pointer array of char-type points that point against the arguments passed through the programme and argv[] is a pointer array.

### Multiple Choice Questions (MCQ)

1. There are \_\_\_\_\_ different types of the C file method.  
A) Zero B) One C) Three D) Two

Ans: **D**

2. EOF is a stdio integer form. Hand is worth hand  
A) 0 B) -1 C) 1 D) None

Ans: **B**

3. What justification is used for truncating the mode below?  
A) p B) a C) w D) None

Ans: **C**

4. For binary files, a \_\_\_\_ must be appended to the mode string.  
A) "b" B) "B" C) binary D) None

Ans: **A**

5. What file form cannot be accessed with fopen()  
A) .htm B) .txt C) .c d) None

Ans: **D**

### Fill in the Blanks:

1. fflush(NULL) flushes all \_\_\_\_\_
2. FILE is of type \_\_\_\_\_
3. FILE reserved word is \_\_\_\_\_
4. If fopen() cannot open a file, it will return \_\_\_\_\_
5. getc() returns EOF when \_\_\_\_\_

### Short Questions:

1. Define File?
2. Define Data Structure?
3. Describe Purpose?
4. Explain Closing a File?
5. Define fprintf()?

### Review Questions:

1. Explain Input / Output Operations on Files?
2. Explain about Error Handling During Input / Output Operations

**Programming Exercises:**

1. Write a program to read name of the student and Marks of the student of multiple number of students and store in a file.
2. Write a program to open a file Write on it and close the file
3. Write a program to open a file Read on it and close the file
4. Write a program to read an existing file
5. Write a program to delete a line from a file

FOR AUTHOR USE ONLY

FOR AUTHOR USE ONLY

**More  
Books!** 



yes  
**I want morebooks!**

Buy your books fast and straightforward online - at one of world's fastest growing online book stores! Environmentally sound due to Print-on-Demand technologies.

Buy your books online at  
**[www.morebooks.shop](http://www.morebooks.shop)**

Kaufen Sie Ihre Bücher schnell und unkompliziert online – auf einer der am schnellsten wachsenden Buchhandelsplattformen weltweit! Dank Print-On-Demand umwelt- und ressourcenschonend produziert.

Bücher schneller online kaufen  
**[www.morebooks.shop](http://www.morebooks.shop)**

KS OmniScriptum Publishing  
Brivibas gatve 197  
LV-1039 Riga, Latvia  
Telefax: +371 686 20455

[info@omniscryptum.com](mailto:info@omniscryptum.com)  
[www.omniscryptum.com](http://www.omniscryptum.com)

OMNIScriptum





FOR AUTHOR USE ONLY