

2020

C Language

FUNDAMENTALS OF C
NARESH LIMBA

Contents

Short History of Programming Languages.....	5
Binary or Machine Language (0,1)	5
Assembly Language	5
High Level Languages	5
Introduction to C (History of C)	6
Features of C	6
1) Simple	6
2) Machine Independent or Portable	6
3) Mid-level programming language	6
4) Structured programming language	6
5) Rich Library.....	6
6) Memory Management.....	6
7) Speed	6
8) Pointer	6
9) Recursion.....	6
10) Extensible	6
Character set in C	7
C Tokens.....	7
Semicolons	7
Comments.....	7
Identifiers	7
Keywords	8
White Space	8
Data Types	8
Basic/Built-in Data Type:	9
Variables.....	10
Constants	11
Operator.....	11
Precedence of Operators in C.....	14
Escape Sequence or Backslash Character.....	14
Installation of C (For Turbo C++)	15
How to Start C	15
Some Shortcut keys	15
Structure of a C Program	16
header files.....	16
void main()	16
variable declaration	16

clrscr()	16
statements1	16
calculation	16
statements2.....	16
getch()	16
Programming Examples	17
Control Statement	25
1) Conditional/Branching/Decision Making Statements:	25
i) Simple if statement:	25
ii) if else statement:	26
iii) Nested if else statement:	29
iv) Ladder else if Statement:	33
v) Switch Statement:	35
2) Looping/Iterative Statements.....	38
1) while Loop.....	39
2) do while Loop.....	42
3) for Loop	45
Nested Looping.....	48
Armstrong Number	74
Palindrome Number	75
Prime Number.....	76
Fibonacci series.....	77
Factorial	78
Array	79
Types of Array	79
1) 1-D Array	79
2) 2-D Array or Multi-Dimensional Array.....	82
Sorting.....	90
1) Selection sort	90
2) Bubble sort	92
Searching.....	94
Linear Search	94
Binary Search	95
Strings.....	98
One Dimensional Array of Characters: (Character Array or String)	98
Input and Display a String	98
Library functions for Strings:	104
Pointer	107

Single Pointer:	107
Functions	111
Library/In-built/pre-define functions	111
User defined function	111
a) Declaration of a function	111
b) Definition of a function.....	111
c) Calling a function	112
Properties of Function:	112
Types of Functions	112
i) No Argument and No Return:	113
ii) With argument and No Return	114
iii) No argument and With Return	115
iv) With argument and With Return	117
Passing arguments to a Function	118
1) Call by Value	118
2) Call by Reference	119
Pointer and String.....	121
1) Compile Time	121
2) Run Time	121
Array to Function.....	122
1) Passing Array using Call by Value	122
2) Passing Array using Call by Reference	122
Structure	126
Defining a Structure	126
Defining a Structure Variable/Object.....	126
1) As Global Scope	126
2) As Local Scope	127
Accessing Structure Data Member	127
Array of Structure	129
Structure as Function Argument	131
Pointer to Structure.....	133
Union.....	136
Defining a Union	136
Accessing Union Members.....	137
File Handling	139
Opening a File	139
fopen()	139
Opening Modes.....	140

Closing a File	140
fclose()	140
Writing a File	140
1) fprintf()	141
2) fputc()	141
3) fputs()	142
Reading a File	142
1) fscanf()	142
2) fgetc()	143
3) fgets()	143
Checking weather, a File exists or not	144
Operator Precedence & Associativity	145

Short History of Programming Languages

Before start learning **C Language**, let's have a look on previous programming languages-

Binary or Machine Language (0,1)

Advantage

- Speed was fast.

Disadvantage

- Machine Depended
- Hard to learn Machine Codes

Assembly Language

In this language to perform a task of command symbols were used to programming. It is also called "**MNEMONIC**".

For Example: - #, *, -, /, + etc.

Advantage

- Easy to programming with symbols rather than remembering the binary codes.

Disadvantage

- Machine Depended
- Assembler was used so speed was slower than Machine Language.

High Level Languages

Today more of software we used are designed and coded in **high level languages**. For Example: - C, C++, Java, Visual Basic etc.

Advantage

- Easy to programming with high level languages (in English).
- Machine independent.

Disadvantage

- **Compiler** or **interpreter** is used to convert the **high-level language** program into low level or machine language so the speed is slower than Machine Language.

Introduction to C (History of C)

C is a **high-level programming** language. C was developed by "**Dennis M. Ritchie**" in **1972** at **AT&T Bell Laboratory (USA)**. It was developed to overcome the problems of previous languages such as **B, BCPL** etc.

Initially, **C language** was developed to be used in UNIX operating system. It inherits many features of previous languages such as B and BCPL.

Features of C

C is the widely used language. It provides a lot of features that are given below.

1) Simple

C is a simple language in the sense that it provides structured approach (**to break the problem into parts**), rich set of library functions, data types etc.

2) Machine Independent or Portable

Unlike assembly language, c programs can be executed in many machines with little bit or no change. But it is not platform-independent.

3) Mid-level programming language

C is also used to do low level programming. It is used to develop system applications such as kernel, driver etc. It also supports the feature of high-level language. That is why it is known as mid-level language.

4) Structured programming language

C is a structured programming language in the sense that we can break the program into parts using functions. So, it is easy to understand and modify.

5) Rich Library

C provides a lot of inbuilt functions that makes the development fast.

6) Memory Management

It supports the feature of dynamic memory allocation. In C language, we can free the allocated memory at any time by calling the **free()** function.

7) Speed

The compilation and execution time of C language is fast.

8) Pointer

C provides the feature of pointers. We can directly interact with the memory by using the pointers. We can use pointers for memory, structures, functions, array etc.

9) Recursion

In c, we can call the function within the function. It provides code reusability for every function.

10) Extensible

C language is extensible because it can easily adopt new features

Character set in C

There are following character sets are available in C Language-

Alphabet	'A' to 'Z' / 'a' to 'z'
Digits	0 to 9
Special Symbols	!, #, %, ^, &, *, (,), {, }, [,], +, -, / etc.

C Tokens

The smallest unit of a program is called C Token. There are following C Tokens are in C-

- Semicolons
- Comments
- Identifiers
- Keywords
- White Space
- Data Types
- Variables
- Constants
- Operators
- Escape Sequence or Backslash Character

Semicolons

In C program, the semicolon is a statement terminator. Each individual statement must be ended with a semicolon. It indicates the end of one logical entity.

For example:

```
printf("Hello, World! ");
```

Comments

Comments are like helping text in our C program and they are ignored by the compiler. They start with `/*` and terminates with the characters `*/` or for single line comment `//` as shown below:

```
/* my first program in C */  
//my first program in C.
```

[For multi-line comments]
[For single-line comments]

Identifiers

A **C identifier** is a name used to identify a variable, function, or any other user-defined item. An identifier starts with a letter **A to Z** or **a to z** or an **underscore** `_` followed by zero or more letters, underscores, and digits (0 to 9).

C does not allow punctuation characters such as `@`, `$`, and `%` within identifiers. **C is a case sensitive programming language. Thus, 'A' and 'a' are two different identifiers in C.** Here are some examples of acceptable identifiers:

```
a    A    a1    saa    _fa    _f9a    a_b9a
```


Keywords

Keywords are reserved words in C. The mean and use of these keywords have been defined to compiler, we can't change their mean. The following list shows the reserved words in C. These reserved words may not be used as constant or variable or any other identifier names.

int	float	char	short	long	double	if	else	switch
case	Do	while	for	break	continue	auto	static	goto
default	struct	void	const	return	sizeof	enum	unsigned	typedef

Table: Reserved words in C

White Space

A line containing only whitespace, possibly with a comment, is known as a blank line, and a C compiler totally ignores it. Whitespace is the term used in C to describe blanks, tabs, newline characters and comments.

Whitespace separates one part of a statement from another and enables the compiler to identify where one element in a statement, such as `int`, ends and the next element begins. Therefore, in the following statement:

```
int age;
```

There must be at least one whitespace character (usually a space) between int and age for the compiler to be able to distinguish them. On the other hand, in the following statement:

```
fruit = apples + oranges;    // get the total fruit
```

No whitespace characters are necessary between fruit and =, or between = and apples, although you are free to include some if you wish for readability purpose.

Data Types

In the C programming language, data types refer to an extensive system used for declaring variables or functions of different types. The type of a variable determines how much space it occupies in storage and after declaring a variable, which type of value stored in this variable, this is maintain by Data Type. The types in C can be classified as follows:

S.N.	Type and Description
01.	Basic Types/Built-in/Pre-define Data Type: They are arithmetic types and consists of the two types: (a) integer types and (b) floating-point types.
02.	Enumerated types: They are again arithmetic types and they are used to define variables that can only be assigned certain discrete integer values throughout the program.
03.	The type void: The type void indicates that no value is available.
04.	Derived Types/User Define Data Type: They include (a) Pointer types, (b) Array types, (c) Structure types, (d) Union types and (e) Function types.

Table: Different Data Types in C

Basic/Built-in Data Type:

Integer Types

We can categories integer types into following -

Type	Storage Size	Value Range	Access Specifier
int	2 Bytes (x86 bit) 4 Bytes (x64 bit)	-32768 to 32767	%d or %i
short	2 Bytes (x86 bit) 4 Bytes (x64 bit)	-32,768 to 32,767	%d or %i
unsigned short	2 Bytes (x86 bit) 4 Bytes (x64 bit)	0 to 65,535	%u
long	4 Bytes	-2,147,483,648 to 2,147,483,647	%ld
unsigned long	4 bytes	0 to 4,294,967,295	%lu

Table: Integer Types

Character Type

We can categories character (**char**) types into following -

Type	Storage Size	Value Range	Access Specifier
Char	1 Bytes	-128 to 127	%c
unsigned char	1 Bytes	0 to 255	%uc
signed char	1 Bytes	-128 to 127	%c

Table: Character Data Type

Floating-Point Types

There are following categories as **float** type-

Type	Storage Size	Value Range	Access Specifier
Float	4 Bytes	1.2E-38 to 3.4E+38	%f
Double	8 Bytes	2.3E-308 to 1.7E+308	%lf
long double	10 Bytes (x86 bit) 12 Bytes (x64 bit)	3.4E-4932 to 1.1E+4932	%lf

Table: Character Data Type

The void Type

The void type specifies that no value is available. It is used in three kinds of situations:

S.N. Types and Description	
01.	Function returns as void There are various functions in C which do not return value or you can say they return void. A function with no return value has the return type as void . For example, void exit (int status);
02.	Function arguments as void There are various functions in C which do not accept any parameter. A function with no parameter can accept as a void. For example, int function1(void);
03.	Pointers to void A pointer of type void * represents the address of an object, but not its type. For example a memory allocation function void *malloc(size_t size); returns a pointer to void which can be casted to any data type.

Table: void Types

Variables

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

Rules to declare a variable-

- The name of a variable can be composed of letters, digits, and the underscore character.
- It must begin with either a letter or an underscore.
- Upper and lowercase letters are distinct because C is case-sensitive.
- Special symbol and space are not allowed.
- Keywords cannot be used as variable.

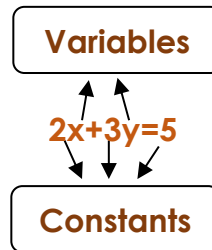
Based on the basic types, there will be the following basic variable types:

Type	Description
char	Typically, a single octet (one byte).
int	The most natural size of integer for the machine.
float	A single-precision floating point value.
double	A double-precision floating point value.

Table: Types of variable

Constants

The constants are treated just like regular variables except that their values cannot be modified after their definition. In other words, we can say that constant never changed their value during execution of program.



Constant can be divided into following:

- **Integer Constant:** Integer constant are without decimal point values. In other words, we can say that a number without having floating point value are called integer. **For example:** - 1, 2, 3, 4, 65, 155, 890 etc.
- **Floating point/ Real Constant:** Real constant has a fractional part, a decimal point. **For Example:** 1.21, 23.5, 65.0000, 78.09 etc.
- **Character Constant:** Any alphabet, any digit, any special symbol that is enclosed with single quote (') are called Character Constant. **For example:** 'a', 'v', 'D', 'T', '8', '#', '*' etc.
- **String Constant:** String is a group (collection) of character and it is enclosed with double quote ("). **For example:** "Naresh", "Hello world", "My Town" etc.

Operator

Operators are the special symbols that are used for calculation to check a specific condition in C Language. There are following 8 types of various operators in C –

- 1) Mathematical or Arithmetical Operator
- 2) Relational Operator
- 3) Logical Operator
- 4) Conditional Operator
- 5) Assignment Operator
- 6) Special Operator
- 7) Bitwise Operator

1) Mathematical or Arithmetical Operator:

Following table shows all the arithmetic operators supported by C language.

Assume variable A holds 10 and variable B holds 20 then:

Operator	Description	Example
+	Adds two operands	A + B will give 30
-	Subtracts second operand from the first	A - B will give -10
*	Multiples both operands	A * B will give 200
/	Divides numerator by de-numerator	B / A will give 2

%	Modulus Operator and remainder of after an integer division	A % B will give 0
++	Increments operator increases integer value by one	A++ will give 11
--	Decrements operator decreases integer value by one	A-- will give 9

Table: Mathematical or Arithmetical Operator

2) Relational Operator:

These operators are used to compile the program according to the condition. **If the condition is true then result will be true ('1') otherwise the result will be also false ('0').**

Assume variable A holds 10 and variable B holds 20, then:

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

Table: Relational Operator

3) Logical Operator:

Logical operators are used to combine multiple conditions.

Assume variable A holds 1 and variable B holds 0, then:

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

Table: Logical Operator

4) Conditional Operator:

Logical operators are used to combine multiple conditions. i.e. **A=1, B=0**

Operator	Description	Example
?:	Called Ternary Operator. Use to check the condition. Expressed by 3 parts – condition ? true part : false part	(A>B)? "A is greater is greater" Result : A is greater

Table: Conditional Operator

5) Assignment Operator:

Assignment operators are used to assign value from right side to left side.

Operator	Description	Example
=	Simple assignment operator, assigns values from right side operands to left side operand	C = A + B will assign value of A + B into C

Table: Assignment Operator

6) Special Operator:

There are also some additional operators are supported by C language-

Operator	Description	Example
&	Returns the address of a variable.	&a; will give actual address of the variable.
*	Pointer to a variable.	*a; will pointer to a variable.
Sizeof()	Returns the size of a variable.	sizeof(a) , where a is integer, will return 2.

Table: Special Operator

7) Bitwise Operators:

The Bitwise operators supported by C language are listed in the following table. Assume variable A holds 60 and variable B holds 13 then:

Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) will give 12 which is 0000 1100
	Binary OR Operator copies a bit if it exists in either operand.	(A B) will give 61 which is 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) will give 49 which is 0011 0001
~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) will give -61 which is 1100 0011 in 2's complement form due to a signed binary number.
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240 which is 1111 0000

>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15 which is 0000 1111
----	---	--

Table: Bitwise Operators

Precedence of Operators in C

The precedence of operator species that which operator will be evaluated first and next. The associativity specifies the operator direction to be evaluated; it may be left to right or right to left. Certain operators have higher precedence than others; for example, the multiplication operator has a higher precedence than the addition operator. [Click here](#) to see names of operators.

Let's understand the precedence by the example given below:

int value=10+20*10;

The value variable will contain 210 because * (multiplicative operator) is evaluated before + (additive operator). The precedence and associativity of C operators is given below, Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first-

Category	Operator	Associativity
Postfix	() [] -> . ++ --	Left to right
Unary	+ - ! ~ ++ -- (type)* & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR		Left to right
Logical AND	&&	Left to right
Logical OR		Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %= >>= <<= &= ^= =	Right to left
Comma	,	Left to right

Table: Precedence & Associativity of Operators

Escape Sequence or Backslash Character

Escape sequence or backslash character are used for formatted output, these are—

\n	Enter in new line
\t	For tab space
\a	Audible bell (If Available in PC)
\b	For backspace
\?	For insert a Question Mark
\\	For Backslash
\'	For Single quote

Installation of C (For Turbo C++)

In **Windows 7** or above versions of **Windows**:

- First of all, download the **compiler** from this [link](#).
- Now extract the zip file and then open **Turbo.C.3.2.zip**.
- Now open the extracted folder and run **setup.exe** file.
- Then follow the installation steps for setting up.

How to Start C

In **Windows 7** or above versions-

Open RUN by pressing **Windows + R**

Type **C:\TurboC++\DOSBox.exe** and press **ENTER** key.

OR

Open My Computer and navigate to **C:\ TurboC++** and then run **DOSBox.exe**

Some Shortcut keys

F2	-	Save a file
F3	-	Open a file
F5	-	Full Screen
F9	-	Compile
Ctrl+F9	-	Compile & Run
Alt + F5	-	To check previous output
Alt+F3	-	Close a file
Alt + x	-	Close C
ctrl + insert	-	Copy selected text
shift + delete	-	cut selected text
shift + insert	-	Paste cut/copied text

Structure of a C Program

header file 1

header file 2

header file 3

```
void main()
{
    variable declaration;
    clrscr();
    statements1;
    calculations;
    statements2;
    getch();
}
```

header files

A header file is a file with extension **.h** which contains **C** function declarations and macro definitions to be shared between several source files.

There are two types of header files: the files that the programmer writes and the files that comes with your compiler.

void main()

There are two things; void is **return type**, it represents type of a value that a function will return or not wherever the function is called.

main() is a **inbuilt function** that is defined by a programmer. This is execution point of every program in **C**.

variable declaration

This is the place where we can define various type of variable that we can use in our program to store various type of data to work with it. Basically, a variable can be an integer, float or a character type.

clrscr()

It is an inbuilt function of C language founded in a header file named as **conio.h**, it is used to clear the recent output from the console window.

statements1

In this part of a C program we generally put some instructive statements so that a use can understand what to do after executing a program.

calculation

this part includes various calculation and logics that we define to solve any given problem using various fundamentals of C language.

statements2

This section of statements is generally including output of the program.

getch()

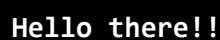
It is a predefine function that is use to hold the output screen until use press any key from keyboard.

Programming Examples

Que 1: Write a program to print a message on output screen.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    clrscr();
    printf("Hello there!!");
    getch();
}
```

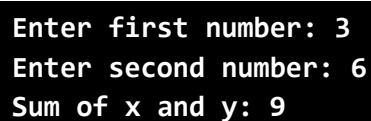
OUTPUT:

A black rectangular box representing a terminal window. Inside, the text "Hello there!!" is displayed in a white, monospaced font.

Que 2: Write a program to input two numbers and display their sum.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int x,y,z;
    clrscr();
    printf("Enter first number: ");
    scanf("%d",&x);
    printf("Enter second number: ");
    scanf("%d",&y);
    z=x+y;
    printf("Sum of x and y: %d ",z);
    getch();
}
```

OUTPUT:

A black rectangular box representing a terminal window. It contains three lines of white, monospaced text: "Enter first number: 3", "Enter second number: 6", and "Sum of x and y: 9".

Que 3: WAP to input height and width of a rectangle and display its area.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int h,w,a;
    clrscr();
    printf("Enter height of rectangle: ");
    scanf("%d",&h);
    printf("Enter width of rectangle: ");
    scanf("%d",&w);
    a=h*w;
    printf("Area is: %d",a);
    getch();
}
```

OUTPUT:

```
Enter height of rectangle: 7
Enter width of rectangle: 5
Area is: 35
```

Que 4: WAP to input two number and display reminder.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int x,y;
    clrscr();
    printf("Enter first number: ");
    scanf("%d",&x);
    printf("Enter second number: ");
    scanf("%d",&y);
    printf("Reminder is: %d ",x%y);
    getch();
}
```

OUTPUT:

```
Enter first number: 7
Enter second number: 2
Reminder is: 1
```

Que 5: WAP to input two numbers and swap them using third variable.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a,b,c;
    clrscr();
    printf("Enter first number: ");
    scanf("%d",&a);
    printf("Enter second number: ");
    scanf("%d",&b);
    printf("Before swap a=%d and b=%d",a,b);
    c=a;          //a=30 => c=30
    a=b;          //b=10 => a=10
    b=c;          //c=30 => b=30
    printf("\nAfter swap a=%d and b=%d",a,b);
    getch();
}
```

OUTPUT:

```
Enter first number: 30
Enter second number: 10
Before swap a=30 and b=10
After swap a=10 and b=30
```

Que 6: WAP to input two numbers and swap them without using third variable.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a,b;
    clrscr();
    printf("Enter first number: ");
    scanf("%d",&a);
    printf("Enter second number: ");
    scanf("%d",&b);
    printf("Before swap a=%d b=%d",a,b);
    a=a+b;          //a=20, b=25 => a=45
    b=a-b;          //a=45, b=25 => b=20
    a=a-b;          //a=45, b=20 => a=25
    printf("\nAfter swap a=%d b=%d",a,b);
    getch();
}
```

OUTPUT:

```
Enter first number: 20
Enter second number: 25
Before swap a=20 and b=25
After swap a=25 and b=20
```

Que 7: WAP to input three-digit number and display sum of digits.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int n,x,y,s;
    clrscr();
    printf("Enter a three-digit number:");
    scanf("%d",&n);
    x=n%10;
    n=n/10;
    x=n%10;
    n=n/10;
    s=x+y+n;
    printf("Sum is: %d",s);
    getch();
}
```

OUTPUT:

```
Enter a three-digit number: 123
Sum is: 6
```

Que 8: WAP five subject marks and display total obtained marks (100/subject) and percentage.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int h,e,m,s,sst,p;
    long t;
    clrscr();
    printf("Enter marks of Hindi: ");
    scanf("%d",&h);
    printf("Enter marks of English: ");
    scanf("%d",&e);
    printf("Enter marks of Math: ");
    scanf("%d",&m);
    printf("Enter marks of Science: ");
    scanf("%d",&s);
    printf("Enter marks of S.ST: ");
```

```

scanf("%d",&sst);
t=h+e+m+s+sst;
p=(t*1)/5;
scanf("%d",&h);
printf("Obtained marks are: %d",t);
printf("Percentage are: %d",t);
getch();
}

```

OUTPUT:

```

Enter marks of Hindi: 89
Enter marks of English: 78
Enter marks of Math: 88
Enter marks of Science: 85
Enter marks of S.ST: 72
Obtained marks are: 412
Percentage are: 82

```

Que 9: WAP to input a character in lower case and display it in upper case.

```

#include<stdio.h>
#include<conio.h>
void main()
{
    char ch;
    clrscr();
    printf("Enter a character in lower case: ");
    scanf("%c",&ch);
    ch=ch-32;
    printf("Character in upper case: %c",ch);
    getch();
}

```

OUTPUT:

```

Enter a character in lower case: a
Character in upper case: A

```

Que 10: WAP to input a character in upper case and display it in lower case.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    char ch;
    clrscr();
    printf("Enter a character in upper case: ");
    scanf("%c",&ch);
    ch=ch+32;
    printf("Character in lower case: %c",ch);
    getch();
}
```

OUTPUT:

```
Enter a character in upper case: Z
Character in upper case: z
```

Que 11: WAP to input basic salary and calculate total salary by following.

DA 12%; TA 11%; HRA 15%; TAX 07%;

```
#include<stdio.h>
#include<conio.h>
void main()
{
    long int bs,ta,da,hra,tax,ns;
    clrscr();
    printf("Enter your basic salary: ");
    scanf("%ld",&bs);
    da=(bs*12)/100;
    ta=(bs*11)/100;
    hra=(bs*15)/100;
    tax=(bs*7)/100;
    ns=bs+da+ta+hra-tax;
    printf("Net salary is: %ld",ns);
    getch();
}
```

OUTPUT:

```
Enter your basic salary: 5000
Net salary is: 6550
```

Que 12: WAP to input two number and check greatest number using conditional operator.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a,b;
    clrscr();
    printf("Enter first number: ");
    scanf("%d",&a);
    printf("Enter second number: ");
    scanf("%d",&b);
    printf((a>b?"A is greater":"B is greater"));
    getch();
}
```

OUTPUT:

```
Enter first number: 5
Enter second number: 3
A is greater
```

Que 13: WAP to input seconds and display how much hours, minutes, seconds will be in given seconds.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int h,m,s;
    clrscr();
    printf("Enter seconds: ");
    scanf("%d",&s);
    h=s/3600;
    s=s%3600;
    m=s/60;
    s=s%60;
    printf("HH:MM:SS\n%d:%d:%d",h,m,s);
    getch();
}
```

OUTPUT:

```
Enter seconds: 5500
HH:MM:SS
1:31:40
```


Que 14: WAP to input rupees from user and count how many notes of 1000, 500, 100, 50, 20, 10, 5, 2, 1 can be in given rupees.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int r,th,fh,h,f,tt,tn,fv,tw,o;
    clrscr();
    printf("Enter rupees: ");
    scanf("%d",&r);
    th=r/1000;    r=r%1000;
    fh=r/500;    r=r%500;
    h=r/100;    r=r%100;
    f=r/50;    r=r%50;
    tt=r/20;    r=r%20;
    tn=r/10;    r=r%10;
    fv=r/5;    r=r%5;
    tw=r/2;    r=r%2;
    o=r/1;    r=r%1;
    printf("Notes of 1000: %d\nNotes of 500: %d\nNotes of 100: %d\n",th,fh,h);
    printf("Notes of 50: %d\nNotes of 20: %d\nNotes of 10: %d\n",f,tt,tn);
    printf("Notes of 5: %d\nNotes of 2: %d\nNotes of 1: %d\n",fv,tw,o);
    getch();
}
```

OUTPUT:

```
Enter rupees: 5214
Notes of 1000: 5
Notes of 500: 0
Notes of 100: 2
Notes of 50: 0
Notes of 20: 0
Notes of 10: 1
Notes of 5: 0
Notes of 2: 2
Notes of 1: 0
```

Control Statement

In 'C' programming to take over control of the program to a specific block of statements we use various control statements, these are –

- 1) Conditional/Branching/Decision Making Statements
- 2) Looping/Iterative Statements

1) Conditional/Branching/Decision Making Statements:

Decision making structures require that the programmer specify one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is true, and optionally, other statements to be executed if the condition false.

In 'C' language conditional statement are used to check the condition. "if" statement is used for the condition checking, "if" statement can be used in following form.

- i) Simple **if** Statement
- ii) **if else** Statement
- iii) Nested **if else** Statement
- iv) **else if** ladder Statement
- v) **switch** Statement

i) Simple **if** statement:

In this type of structure, a condition is given. If the given condition is true than the block part after the condition will be execute otherwise not.

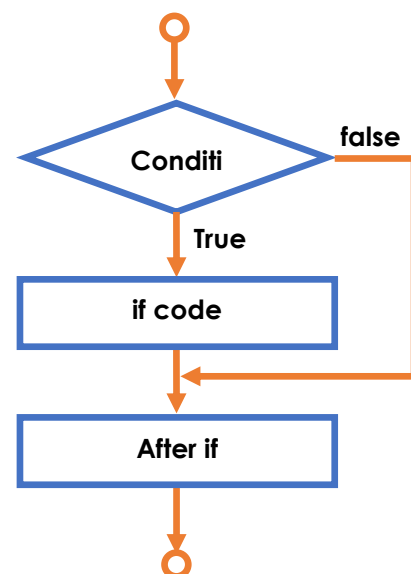
Syntax: -

```
if(<expression>)  
{  
    //code to be executed  
}
```

For Example:

Que 15: WAP to check greater number between two.

```
#include<stdio.h>  
#include<conio.h>  
void main()  
{  
    int a=10,b=14;  
    clrscr();  
    if(a>b)  
    {  
        printf("A is greater: %d",a);  
    }  
    if(b>a)  
    {  
        printf("B is greater: %d",a);  
    }  
    getch();  
}
```



OUTPUT:

```
B is greater: 14
```

Que 16: WAP to input basic salary and calculate net salary according to following – DA is 7% of basic salary, TA is 8% of basic salary, HRA is 12% of salary If salary >= 15000 then tax will be 10% of basic salary.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    long int bs,ta,da,hra,tax=0,ns;
    clrscr();
    printf("Enter your basic salary: ");
    scanf("%ld",&bs);
    da=(bs*7)/100;
    ta=(bs*8)/100;
    hra=(bs*12)/100;
    if(bs>=15000)
    {
        tax=(bs*10)/100;
        printf("Tax is :%d",tax);
    }
    ns=bs+da+ta+hra-tax;
    printf("\nNet salary is :%ld",ns);
    getch();
}
```

OUTPUT:

```
Enter your basic salary: 15500
Tax is: 1550
Net salary is: 19840
```

ii) if else statement:

If else statement is used to execute true block if condition is true otherwise false part will be executed.

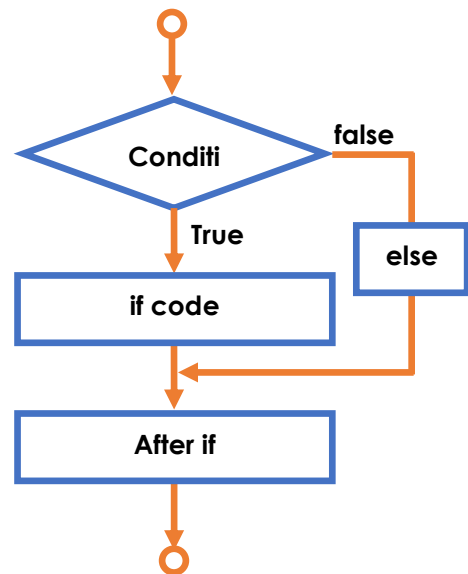
Syntax: -

```
if(<expression>)
{
    //code to be executed when condition is true.
}
else
{
    //code to be executed when condition is false.
}
```

For Example:

Que 17: WAP to check which is greater between two using if-else.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a=10,b=14;
    clrscr();
    if(a>b)
    {
        printf("A is greater: %d",a);
    }
    else
    {
        printf("B is greater: %d",b);
    }
    getch();
}
```



Que 18: WAP to input a number and check it is even or odd.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a;
    clrscr();
    printf("Enter a number: ");
    scanf("%d",&a);
    if(a%2==0)
    {
        printf("The number is EVEN: %d",a);
    }
    else
    {
        printf("The number is ODD: %d",a);
    }
    getch();
}
```

OUTPUT:

```
Enter a number: 15
The number is ODD: 15
```

Que19: WAP to input a character and display its vice-versa.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    char ch;
    clrscr();
    printf("Enter a character: ");
    scanf("%c",&ch);
    if(ch>=65 && ch<=90)
    {
        ch=ch+32;
    }
    else
    {
        ch=ch-32;
    }
    printf("%c",ch);
    getch();
}
```

OUTPUT:

```
Enter a character: a
A
```

Que 20: WAP to input age and check eligibility for voting.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int age;
    clrscr();
    printf("Enter your age: ");
    scanf("%d",&age);
    if(age>=18)
    {
        printf("Eligible for voting.");
    }
    else
    {
        printf("Not eligible for voting.");
    }
    getch();
}
```

OUTPUT:

```
Enter your age: 21
Eligible for voting.
```

iii) *Nested if else statement:*

Using nested if else we can use one if or else if statement inside another if or else if statement(s).

Syntax: -

```
if(<condition1>)
{
    if(<condition2>)
    {
        //code to be executed when conditions 1 and 2 are true.
    }
}
else
{
    //code to be executed when condition 1 is false.
}
```

OR

```
if(<condition1>)
{
    if(<condition2>)
    {
        //code to be executed when conditions 1 and 2 are true.
    }
}
else
{
    if(<condition>)
    {
        //code to be executed when condition is true.
    }
    else
    {
        //code to be executed when condition is false.
    }
}
```

For example:

Que 21: WAP to check which is greater among three numbers.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a=10,b=14,c=15;
    clrscr();
    if(a>b)
    {
        if(a>c)
        {
            printf("A is greater: %d",a);
        }
        else
        {
            printf("C is greater: %d",c);
        }
    }
    else
    {
        if(b>c)
        {
            printf("B is greater: %d",b);
        }
        else
        {
            printf("C is greater: %d",c);
        }
    }
    getch();
}
```

OUTPUT:

C is greater: 15

Que 22: WAP to calculate net amount according to purchase amount and the type of clothes, conditions are following –

Handloom

If purchase>25K than discount 20%, If pur>15K and <=25K than discount 12% otherwise dis. Is 7%.

Mill

If purchase>20K than discount 25%, if pur>15K and <=20K than dis. is 12%, Ohterwise no discount.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    long int pur,na,dis=0;
    char ch;
    clrscr();
    printf("Press 'h' for Handloom and 'm' for Mill clothes: ");
    scanf("%c",&ch);
    printf("Enter purchasing amount: ");
    scanf("%ld",&pur);
    if(ch=='h')
    {
        if(pur>25000)
        {
            dis=(pur*20)/100;
            printf("Discount is :%d",dis);
        }
        else
        {
            if(pur>15000 && pur<=25000)
            {
                dis=(pur*12)/100;
                printf("Discount is :%d",dis);
            }
            else
            {
                dis=(pur*7)/100;
                printf("Discount is :%d",dis);
            }
        }
    }
    if(ch=='m')
    {
        if(pur>20000)
        {
            dis=(pur*25)/100;
            printf("Discount is :%d",dis);
        }
    }
}
```



```

    }
    else
    {
        if(pur>15000 && pur<=20000)
        {
            dis=(pur*20)/100;
            printf("Discount is :%d",dis);
        }
        else
        {
            printf("\nNo discount applicable on this amount");
        }
    }
}
na=pur-dis;
printf("\nNet amount: %ld",na);
getch();
}

```

OUTPUT:

```

Press 'h' for handloom and 'm' for Mill clothes: m
Enter purchasing amount: 1500
No discount applicable on this amount
Net amount: 1500

```

Que 23: WAP to input subject name and marks and if subject is science and marks are greater than 60 then grant admission otherwise not.

```

#include<stdio.h>
#include<conio.h>
void main()
{
    int marks;
    char sub;
    clrscr();
    printf("Enter 's' for science: ");
    scanf("%c",&sub);
    printf("Enter marks: ");
    scanf("%d",&marks);
    if(sub=='s')
    {
        if(marks>60)
        {
            printf("Admission confirmed");
        }
        else
        {

```

```

        printf("Admission decline due to marks less than 60");
    }
}
else
{
    printf("\nSubject must be science for admission");
}
}

```

OUTPUT:

```

Enter 's' for science: s
Enter marks: 59
Admission decline due to marks are less than 60

```

iv) Ladder else if Statement:

Syntax: -

```

if(<condition1>)
{
    //code to be executed when condition1 is true.
}
else if(<condition2>)
{
    //code to be executed when condition1 is false and condition2 is true.
}
else if(<condition3>)
{
    //code to be executed when condition2 is false and condition3 is true.
}
else
{
    //code to be executed when condition3 is false.
}

```

For Example: WAP to check either a number is positive or negative.

```

#include<stdio.h>
#include<conio.h>
void main()
{
    int a=10;
    clrscr();
    if(a>0)
    {
        printf("A is positive: %d",a);
    }
    else if(a<0)

```

```

        {
            printf("A is negative: %d",b);
        }
        else
        {
            printf("A is not a number!!!");
        }
        getch();
    }
}

```

OUTPUT:

```
A is positive: 10
```

Que 24: WAP to calculate wages according to working hours as per following rates –

For first 4 hours	-	10RS/h.
For next 3 hours	-	20RS/h.
For next 2 hours	-	30RS/h.
For remaining hours	-	45RS/h.

```

#include<stdio.h>
#include<conio.h>
void main()
{
    int h,w;
    clrscr();
    printf("Enter working hours: ");
    scanf("%d",&h);
    if(h<=4)
    {
        w=h*10;
    }
    else if(h>4 && h<=7)
    {
        w=(h-4)*20+40;
    }
    else if(h>7 && h<=9)
    {
        w=(h-7)*30+100;
    }
    else
    {
        w=(h-9)*45+160;
    }
    printf("Total wages: %d",w);
    getch();
}

```

OUTPUT:

```
Enter working hours: 10
Total wages: 205
```

Que 25: WAP to calculate Electricity bill according to units as per following rates-

For first 100 Units - 4RS/unit

=>

For next 70 Units - 5RS/unit.

For next 50 Units - 6RS/unit.

=>

For remaining Units - 7RS/unit.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int u,bill;
    clrscr();
    printf("Enter Electricity Units: ");
    scanf("%d",&u);
    if(u<=100)
    {
        bill=u*4;
    }
    else if(u>100 && u<=170)
    {
        bill=(u-100)*5+400;
    }
    else if(u>170 && u<=220)
    {
        bill=(u-170)*6+400+350;
    }
    else
    {
        bill=(u-220)*7+400+350+300;
    }
    printf("Total Bill: %d",bill);
    getch();
}
```

OUTPUT:

```
Enter Electricity Units: 170
Total Bill: 750
```

v) Switch Statement:

In 'C' a switch statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each switch case.

In other words, we can say that switch statement is used when we've a list of different choice for single variable or expression. If value of the switched variable is matched with any of **case** within switch than associated statement will be executes and after that **break** keyword is used to exit the switch. If any of case value doesn't match with variable than **default** case will be execute.

CHECKPOINT: *The variable after case should be an integer or character type. It cannot be floating type.*

Syntax:

```
switch(expression)
{
    case <constant-expression>:
        //code to be executed;
        break;                                //optional
    case <constant-expression>:
        //code to be executed;
        break;                                //optional
    default:
        //code to be executed if all cases are not matched;
}
```

In a switch statement any number of cases can be exists. There is no such limit for associating cases inside switch statement.

For example:

Que 26: WAP to print greeting message using switch statement.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    char grade = 'B';
    switch(grade)
    {
        case 'A':
            printf("Excellent!!");
            break;
        case 'B':
            printf("Good!!");
            break;
        case 'C':
            printf("Fair");
            break;
        case 'D':
            printf("Bad!!");
            break;
        default:
            printf("Not a Grade! Please try again!!");
    }
    getch();
}
```


OUTPUT:

Good!!

Que 27: WAP to enter a character and check it is vowel or consonant.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    char ch;
    clrscr();
    printf("Enter a character: ");
    scanf("%c",&ch);
    switch(ch)
    {
        case 'a':    case 'A':
        case 'e':    case 'E':
        case 'o':    case 'O':
        case 'u':    case 'U':
            printf("\nVowel");
            break;
        default:
            printf("Consonant");
    }
    getch();
}
```

OUTPUT:



```
Enter a character: a
Vowel
```

Que 28: WAP to input two numbers and display calculation according to user choice as sum, subtraction, divide, reminder, multiply.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a,b,c;
    clrscr();
    printf("\n1. Press 1 for sum\n2. Press 2 for subtraction\n3. Press 3 for multiply\n4. Press 4 for divide\n5. Press 5 for reminder\n");
    printf("Enter your choice: ");
    scanf("%d",&c);
    printf("Enter first value= ");
    scanf("%d",&a);
    printf("Enter second value= ");
    scanf("%d",&b);
    switch (c)
    {
        case 1:
```

```

        printf("\nSum is= %d",a+b);
    break;
    case 2:
        printf("\nSub is= %d",a-b);
    break;
    case 3:
        printf("\nMul is= %d",a*b);
    break;
    case 4:
        printf("\nDivide is= %d",a/b);
    break;
    case 5:
        printf("\nReaminder is= %d",a%b);
    break;
    default:
        printf("Invalid Choice !!");
}
getch();
}

```

OUTPUT:

```

1. Press 1 for sum
2. Press 2 for subtraction
3. Press 3 for multiply
4. Press 4 for divide
5. Press 5 for reminder
Enter your choice: 3
Enter first value=4
Enter second value=5
Mul is= 20

```

2) Looping/Iterative Statements

There may be a situation, when we need to execute a block of code several numbers of times. In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on. Programming languages provide various control structures that allow for more complicated execution paths.

A loop statement allows us to execute a statement or group of statements multiple times till the given condition remaining true. Every loop can be expressed in three parts –

- Initialization
- Condition
- Increment/Decrement

Initialization: Initialization is beginning of the loop. It is the point where the loop starts.

Condition: Condition is the responsible for terminating of a loop. The statements written inside the loop body execute till the given condition is true.

Increment/Decrement: This is difference between the executed statements of loop body. If loop executed in ascending order than increment statement works otherwise if loop executed in descending order than decrement statement works.

There are 3 types of loop are available in 'C' language –

- 1) **while** Loop
- 2) **do-while** Loop
- 3) **for** loop

1) **while** Loop

In this type of looping structure first initialization is done before loop starts and the condition is given inside the parenthesis "()" of *while* loop. If the given condition is true than the statement inside loop body will be execute otherwise not.

CHECKPOINT: *while loop is also known as entry control loop because the condition is checked in starting of loop.*

Syntax:

```
<Initialization>;
while(<condition>)
{
    statement(s);
    <increment/decrement>;
}
```

For example

Que 29: Print your name 5 times using while loop.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int n;
    clrscr();
    n=1;
    while(n<=5)
    {
        printf("Naresh\n");
        n++;
    }
    getch();
}
```

OUTPUT:

```
Naresh
Naresh
Naresh
Naresh
Naresh
```


Que 30: WAP to print 10 to 1 (in decreasing order) using while loop.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int n;
    clrscr();
    n=10;
    while(n>=1)
    {
        printf("Naresh\t");
        n- -;
    }
    getch();
}
```

OUTPUT:

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

Que 31: WAP to print 1,3,5,7.....23 series using while loop.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int n;
    clrscr();
    n=1;
    while(n<=23)
    {
        printf("%d\t",n);
        n+2;
    }
    getch();
}
```

OUTPUT:

```
1  3  5  7  9  11  13  15  17  19  21  23
```

Que 32: WAP to print 2,4,6,8.....20 series using while loop.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int n;
    clrscr();
    n=2;
    while(n<=20)
    {
        printf("%d\t",n);
        n+=2;
    }
    getch();
}
```

OUTPUT:

2	4	6	8	10	12	14	16	18	20
---	---	---	---	----	----	----	----	----	----

Que 33: WAP to print A,B,C,D.....Z using while loop.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int c;
    clrscr();
    c=65;
    while(c<=90)
    {
        printf("%c\t",c);
        c++;
    }
    getch();
}
```

OUTPUT:

A	B	C	D	E	F	G	H	I	J	K	L
M	N	O	P	Q	R	S	T	U	V	Z	X
Y	Z										

Que 34: WAP to print 1.1,1.2,1.3.....2 series using while loop.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    float n;
    clrscr();
    n=1.1;
    while(n<=2)
    {
        printf("%f\t",n);
        n=n+0.1;
    }
    getch();
}
```

OUTPUT:

```
1.1  1.2  1.3  1.4  1.5  1.6  1.7  1.8  1.9  2.0
```

2) do while Loop

The **do-while** loop in C programming language checks its condition at the bottom of the loop.

A **do-while** loop is similar to a while loop, except that a **do-while** loop is guaranteed to execute at least one time.

CHECKPOINT: *do-while loop is also called exit control loop because the condition in this loop check in last of loop.*

Syntax:

```
<initialization>;
do
{
    Statement(s);
    <increment/decrement>;
} while(<condition>);
```

For example

Que 35: Print 10 to 1 series using do-while loop.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int n;
    clrscr();
    n=10;
```

```

do
{
    printf("Naresh\t");
    n--;
}while(n>=1);
getch();
}

```

OUTPUT:

```

10  9  8  7  6  5  4  3  2  1

```

Que 36: WAP to print 1.1,1.2,1.3,.....2 series using while loop.

```

#include<stdio.h>
#include<conio.h>
void main()
{
    float n;
    clrscr();
    n=1.1;
    do
    {
        printf("%f\t",n);
        n=n+0.1;
    }while(n<=2);
    getch();
}

```

OUTPUT:

```

1.1  1.2  1.3  1.4  1.5  1.6  1.7  1.8  1.9  2.0

```

Que 37: WAP to print a,b,c,d.....z using while loop.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int c;
    clrscr();
    c=97;
    do
    {
        printf("%c\t",c);
        c++;
    } while(c<=122);
    getch();
}
```

OUTPUT:

A	b	c	d	e	f	g	h	i	j	K	l
m	n	o	p	q	r	s	t	u	v	w	x
y	z										

Que 38: WAP to print 1,8,27,64.....1000 series using while loop.

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

OUTPUT:

1	8	27	64	125	216	343	512	729	1000
---	---	----	----	-----	-----	-----	-----	-----	------

Que 39: WAP to print 1,4,9,16,.....100 series using while loop.

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

OUTPUT:

```
1    4    9    16   25   36   49   64   81   100
```

3) for Loop

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

Syntax:

```
for(<initialization>;<condition>;<increment/decrement>)
{
    Statement(s);
}
```

Here is the flow of control in a for loop:

- 1) The initialization step is executed first, and only once. This step allows you to declare and initialize any loop control variables. You are not required to put a statement here.
- 2) Next, the condition is evaluated. If it is true, the body of the loop is executed. If it is false, the body of the loop does not execute and flow of control jumps to the next statement just after the **for** loop.
- 3) After the body of the **for** loop executes, the flow of control jumps back up to the increment statement. This statement allows you to update any loop control variables. This statement can be left blank, as long as a semicolon appears after the condition.
- 4) The condition is now evaluated again. If it is true, the loop executes and the process repeats itself (body of loop, then increment step, and then again condition). After the condition becomes false, the **for** loop terminates.

For example

Que 40: Print 1 to 10 numbers using for loop.

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

OUTPUT:

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

Que 41: WAP to print 1,3,5,7.....n times using for loop.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int n,m;
    clrscr();
    printf("Enter a range for loop: ");
    scanf("%d",&m);
    for(n=1;n<=m;n+=2)
    {
        printf("%d\t",n);
    }
    getch();
}
```

OUTPUT:

```
Enter a range for loop: 16
1    3    5    7    9    11    13    15
```

Que 42: WAP to print table of a number input by user, using for loop.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int n,t;
    clrscr();
    printf("Enter a number to print its table: ");
    scanf("%d",&t);
    for(n=1;n<=10;n++)
    {
        printf("%d\t",n*t);
    }
    getch();
}
```

OUTPUT:

```
Enter a number to print its table: 2
2    4    6    8    10   12   14   16   18   20
```

Que 43: WAP to print cube of 'n' numbers using for loop.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int n,c;
    clrscr();
    printf("Enter a range for print cubes: ");
    scanf("%d",&c);
    for(n=1;n<=c;n++)
    {
        printf("%d\t",n*n*n);
    }
    getch();
}
```

OUTPUT:

```
Enter a range for print cubes: 8
1    8    27   64   125  216  343  512
```


Nested Looping

C programming language allows us to use one loop inside another loop. Following section shows few examples to understand the concept better.

Syntax:

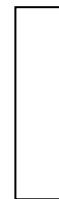
```
for(<initialization>;<condition>;<increment/decrement>)  
{  
    for(<initialization>;<condition>;<increment/decrement>)  
    {  
        statement(s);  
    }  
    statement(s);  
}
```

In nested loops there are two loops are used as we can see in the syntax and in following example, first loop can be called as outside loop and the another can call as inside loop because the second loop is inside the first loop. The outside loop considers as responsible for give output in a single horizontal line called *row* and the inside loop can consider as *column* loop because it gives output in columns. Let see following diagram –

What is a row? **a horizontal record is called row.**

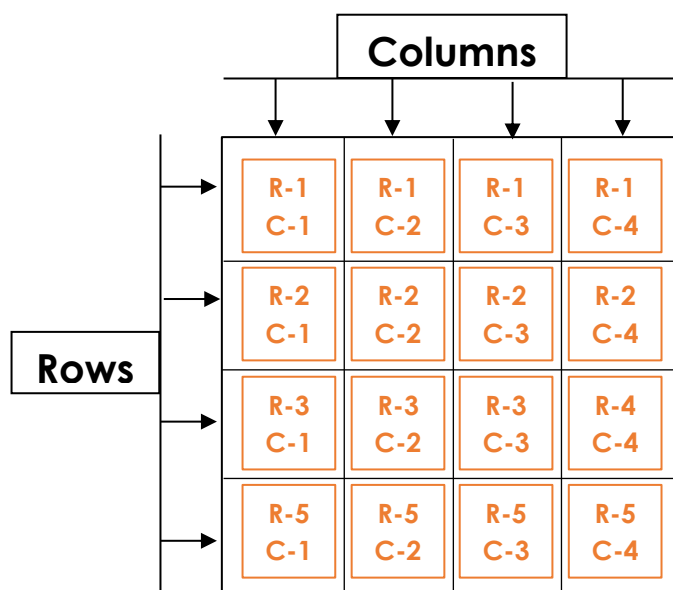


What is a column? **a vertical attribute is called column.**



In nested looping every row may have multiple columns as required, so we can say that for every 1 loop of row the column loop will execute maximum limit of given condition (in other words we can say that for every 1 loop of row the column loop will executes till the condition remains true).

There is following a matrix is shown in figure that indicates 5 rows and 5 columns also. We separate every row by using "\n" character. For every 1 one row there are 5 columns as shown in next figure –



For example:

Que 44: Print the following pattern using nested loops.

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

```
#include<stdio.h>  
#include<conio.h>  
void main()  
{  
    int r,c;  
    clrscr();  
    for(r=1;r<=5;r++)  
    {  
        for(c=1;c<=5;c++)  
        {  
            printf("*");  
        }  
        printf("\n");  
    }  
    getch();  
}
```

OUTPUT:

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

Que 45: Print the following pattern using nested loops.

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

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int r,c;
    clrscr();
    for(r=0;r<=4;r++)
    {
        for(c=65;c<=69;c++)
        {
            printf("%c",c);
        }
        printf("\n");
    }
    getch();
}
```

OUTPUT:

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

Que 46: Print the following pattern using nested loops.

```
F F F F F
D D D D D
C C C C C
B B B B B
A A A A A
```

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int r,c;
    clrscr();
    for(r=69;r>=65;r--)
    {
        for(c=0;c<=4;c++)
        {
            printf("%c",r);
        }
        printf("\n");
    }
    getch();
}
```

OUTPUT:

```
F F F F F
D D D D D
C C C C C
B B B B B
A A A A A
```

Que 47: Print the following pattern using nested loops.

1 2 3 4 5

1 2 3 4 5

1 2 3 4 5

1 2 3 4 5

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int r,c;
    clrscr();
    for(r=0;r<=4;r++)
    {
        for(c=1;c<=5;c++)
        {
            printf("%d",c);
        }
        printf("\n");
    }
    getch();
}
```

OUTPUT:

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

Que 48: Print the following pattern using nested loops.

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

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int r,c;
    clrscr();
    for(r=1;r<=5;r++)
    {
        for(c=1;c<=5;c++)
        {
            printf("%d",r);
        }
        printf("\n");
    }
    getch();
}
```

OUTPUT:

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

Que 49: Print the following pattern using nested loops.

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

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int r,c;
    clrscr();
    for(r=5;r<=1;r--)
    {
        for(c=1;c<=5;c++)
        {
            printf("%d",r);
        }
        printf("\n");
    }
    getch();
}
```

OUTPUT:

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

Que 50: Print the following pattern using nested loops.

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

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int r,c;
    clrscr();
    for(r=1;r<=5;r++)
    {
        for(c=5;c>=1;c--)
        {
            printf("%d",c);
        }
        printf("\n");
    }
    getch();
}
```

OUTPUT:

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


Que 51: Print the following pattern using nested loops.

1 2 3 4 5

6 7 8 9 10

11 12 13 14 15

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

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

```
void main()
```

```
{
```

```
    int r,c,x=1;
```

```
    clrscr();
```

```
    for(r=1;r<=3;r++)
```

```
    {
```

```
        for(c=1;c<=5;c++)
```

```
        {
```

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

```
            x++;
```

```
        }
```

```
        printf("\n");
```

```
    }
```

```
    getch();
```

```
}
```

OUTPUT:

```
1 2 3 4 5
```

```
6 7 8 9 10
```

```
11 12 13 14 15
```

Que 52: Print the following pattern using nested loops.

```
A B C D E
F G H I J
K L M N O
P Q R S T
U V W X Y
```

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int r,c,a;
    clrscr();
    a=65;
    for(r=1;r<=5;r++)
    {
        for(c=1;c<=5;c++)
        {
            printf("%c",a);
            a++;
        }
        printf("\n");
    }
    getch();
}
```

OUTPUT:

```
A B C D E
F G H I J
K L M N O
P Q R S T
U V W X Y
```

Que 53: Print the following pattern using nested loops.

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

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int r,c,a;
    clrscr();
    for(r=65;r<=69;r++)
    {
        for(c=1;c<=5;c++)
        {
            printf("%c",r);
        }
        printf("\n");
    }
    getch();
}
```

OUTPUT:

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

Que 54: Print the following pattern using nested loops.

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

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int r,c;
    clrscr();
    for(r=1;r<=5;r++)
    {
        for(c=69;c>=65;c--)
        {
            printf("%c",c);
        }
        printf("\n");
    }
    getch();
}
```

OUTPUT:

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

Que 55: Print following pattern using nested loops.

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

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int r,c;
    clrscr();
    for(r=65;r<=69;r++)
    {
        for(c=65;c<=r;c++)
        {
            printf("%c",c);
        }
        printf("\n");
    }
    getch();
}
```

OUTPUT:

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

Que 56: Print following pattern using nested loops.

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

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int r,c;
    clrscr();
    for(r=65;r<=69;r++)
    {
        for(c=65;c<=r;c++)
        {
            printf("%c",r);
        }
        printf("\n");
    }
    getch();
}
```

OUTPUT:

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

Que 57: Print following pattern using nested loops.

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

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int r,c,s,h=4;
    clrscr();
    for(r=65;r<=69;r++)
    {
        for(s=1;s<=h;s++)
        {
            printf(" ");
        }
        for(c=65;c<=r;c++)
        {
            printf("%c",r);
        }
        h--;
        printf("\n");
    }
    getch();
}
```

OUTPUT:

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

Que 58: Print following pattern using nested loops.

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

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int r,c,s,h=4;
    clrscr();
    for(r=65;r<=69;r++)
    {
        for(s=1;s<=h;s++)
        {
            printf(" ");
        }
        for(c=65;c<=r;c++)
        {
            printf("%c",c);
            printf(" ");
        }
        h--;
        printf("\n");
    }
    getch();
}
```

OUTPUT:

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


Que 59: Print following pattern using nested loops.

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

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int r,c,s,h=0;
    clrscr();
    for(r=69;r>=65;r--)
    {
        for(s=1;s<=h;s++)
        {
            printf(" ");
        }
        for(c=65;c<=r;c++)
        {
            printf("%c",c);
        }
        h++;
        printf("\n");
    }
    getch();
}
```

OUTPUT:

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

Que 60: Print following pattern using nested loops.

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

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int r,c;
    clrscr();
    for(r=5;r>=1;r--)
    {
        for(c=1;c<=r;c++)
        {
            printf("%d",r);

        }
        printf("\n");
    }
    getch();
}
```

OUTPUT:

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

Que 61: Print following pattern using nested loops.

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

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int r,c,s,h=4;
    clrscr();
    for(r=5;r>=1;r--)
    {
        for(s=5;s>=r;s--)
        {
            printf(" ");
        }
        for(c=1;c<=r;c++)
        {
            printf("%d",r);
        }
        h--;
        printf("\n");
    }
    getch();
}
```

OUTPUT:

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

Que 62: Print following pattern using nested loops.

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

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int r,c;
    clrscr();
    for(r=1;r<=5;r++)
    {
        for(c=5;c>=r;c--)
        {
            printf("%d",c);

        }
        printf("\n");
    }
    getch();
}
```

OUTPUT:

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

Que 63: Print following pattern using nested loops.

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

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int r,c,s,h=4;
    clrscr();
    for(r=1;r<=5;r++)
    {
        for(s=1;s<=h;s++)
        {
            printf(" ");
        }
        for(c=1;c<=r;c++)
        {
            printf("%d",r);
        }
        h--;
        printf("\n");
    }
    getch();
}
```

OUTPUT:

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

Que 64: Print following pattern using nested loops.

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

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int r,c,s,h=0;
    clrscr();
    for(r=5;r>=1;r--)
    {
        for(s=1;s<=h;s++)
        {
            printf(" ");
        }
        for(c=1;c<=r;c++)
        {
            printf("%d",c);
        }
        h++;
        printf("\n");
    }
    getch();
}
```

OUTPUT:

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

Que 65: Print following pattern using nested loops.

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

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int r,c,s,h=4;
    clrscr();
    for(r=1;r<=5;r++)
    {
        for(s=1;s<=h;s++)
        {
            printf(" ");
        }
        for(c=r;c>=1;c--)
        {
            printf("%d",c);
        }
        h--;
        printf("\n");
    }
    getch();
}
```

OUTPUT:

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

Que 66: Print following pattern using nested loops.

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

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int r,c,s,h=4;
    clrscr();
    for(r=65;r<=69;r++)
    {
        for(s=1;s<=h;s++)
        {
            printf(" ");
        }
        for(c=r;c>=65;c--)
        {
            printf("%c",c);
        }
        for(c=66;c<=r;c++)
        {
            printf("%c",c);
        }
        h--;
        printf("\n");
    }
    getch();
}
```

OUTPUT:

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


Que 67: Print following pattern using nested loops.

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

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int r,c,s,h=8;
    clrscr();
    for(r=65;r<=69;r++)
    {
        for(c=65;c<=r;c++)
        {
            printf("%c",c);
        }
        for(s=1;s<=h;s++)
        {
            printf(" ");
        }
        for(c=r;c>=65;c--)
        {
            printf("%c",c);
        }
        h=h-2;
        printf("\n");
    }
    getch();
}
```

OUTPUT:

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

Que 68: Print following pattern using nested loops.

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

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int r,c,s,h=0;
    clrscr();
    for(r=69;r>=65;r--)
    {
        for(s=1;s<=h;s++)
        {
            printf(" ");
        }
        for(c=65;c<=r;c++)
        {
            printf("%c",c);
            printf(" ");
        }
        h++;
        printf("\n");
    }
    getch();
}
```

OUTPUT:

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

Armstrong Number


Armstrong number is a number which is equal to sum of cube power of every digit in the number. To check whether a number is Armstrong or not we have to separate each digit of that number and then multiply the number by 3 time and so on for each digit. After multiply we have to add the cube of the number with next digit's cube. If the sum of the cube of every digit is same as the given number then this number will be treated as Armstrong otherwise not.

For example: 153 $1^3+5^3+3^3$ will be $1+125+27=153$

Que 69: WAP to input a number and check it is Armstrong or not.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int n,x,r,s=0;
    clrscr();
    printf("Enter a number: ");
    scanf("%d",&n);
    x=n;
    while(n>0)
    {
        r=n%10;
        s=s+(r*r*r);
        n=n/10;
    }
    If(s==x)
    {
        printf("\n%d is an Armstrong number",x);
    }
    else
    {
        printf("\n%d is not an Armstrong number",x);
    }
    getch();
}
```

OUTPUT:



```
Enter a number: 153
153 is an Armstrong number
```

In above example we have check 153 for Armstrong and as we shown before the sum of 1, 5, 3 will be same as 153, So this is an Armstrong number.

Palindrome Number

Those numbers which will be same if we written them reserves, are called Palindrome number. Not only numbers but any string that is looking same after reversing it, is also treated as palindrome.

For Example [Numbers]: 121,242,585,999,656 etc.

For Example [string]: WOW, MOM, POP, NON etc.

Que 70: WAP to input a number and check it is Palindrome or not.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int n,x,r,s=0;
    clrscr();
    printf("Enter a number: ");
    scanf("%d",&n);
    x=n;
    while(n>0)
    {
        r=n%10;
        s=r+(s*10);
        n=n/10;
    }
    If(s==x)
    {
        printf("\n%d is a Palindrome number",x);
    }
    else
    {
        printf("\n%d is not a Palindrome number",x);
    }
    getch();
}
```

OUTPUT:

```
Enter a number: 151
151 is a Palindrome number
```

Prime Number

Prime number is a number which is only divided by itself and by 1 are treated as prime number.

For Example: 1, 2, 3, 5, 7, 11, 13, 17, 19 etc.

Que 71: WAP to input a number and check it is Prime or not.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int n,x;
    clrscr();
    printf("Enter a number: ");
    scanf("%d",&n);
    x=2;
    while(x<=n-1)
    {
        if(n%x==0)
        {
            Break;
        }
        x++;
    }
    if(n==x || n==1)
    {
        printf("\n%d is a Prime number",n);
    }
    else
    {
        printf("%d is not a Prime number",n);
    }
    getch();
}
```

OUTPUT:

```
Enter a number: 7
7 is a Prime number
```

Fibonacci series

Que 72: WAP to print Fibonacci series till a range given by user.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int n,a=0,b=1,c;
    clrscr();
    printf("Enter a number: ");
    scanf("%d",&n);
    printf("%d\t%d\t",a,b);
    while(c<n)
    {
        c=a+b;
        printf("%d\t",c);
        a=b;
        b=c;
    }
    getch();
}
```

OUTPUT:

```
Enter a number: 15
0    1    1    2    3    5    8    13    21
```

Factorial

The factorial is the product of all integers less than or equal to **n** but greater than or equal to 1.

For Example: Assume we want to calculate the factorial of a number **4**, then-

$$4 \times 3 \times 2 \times 1 = 24$$

Que 73: WAP to input a number and display its factorial.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int n,x,f=1;
    clrscr();
    printf("Enter a number: ");
    scanf("%d",&n);
    for(x=1;x<=n;x++)
    {
        f=f*x;
    }
    printf("Factorial of %d is: %d",n,f);
    getch();
}
```

OUTPUT:

```
Enter a number: 5
Factorial of 5 is: 120
```

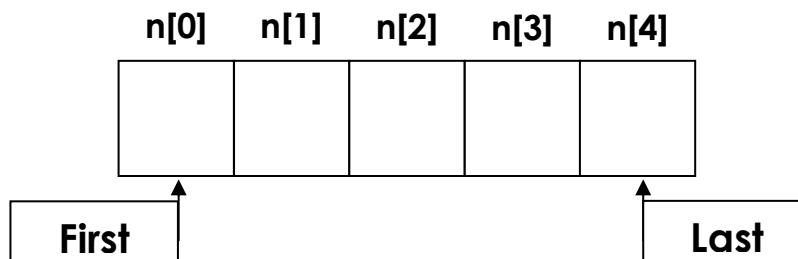
Array

C programming language provides a data structure called the **array**, which can store a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type. *Array is also called derived data type.*

CHECKPOINT: Array is also called sub-scripted variables.

Properties of an array:

- An array can store only similar type of information.
- First element of an array is start from 0(zero) index, so the last index is less than one of total size of an array.
- Elements are stored continuously in memory location.
- The address of first element is called base address of array.
- Instead of declaring individual variables, such as n0, n1, ..., and n20, you declare one array variable such as numbers and use n[0], n[1], and ..., n[20] to represent individual variables. A specific element in an array is accessed by an index.



Types of Array

- 1) 1-D Array (1-Dimensional array)
- 2) Multi-Dimensional Array

1) 1-D Array

Declaring an Array

To declare an array in C, a programmer specifies the type of the elements and the number of elements required by an array as follows:

Syntax: <type> <arrayName>[<arraySize>]; {1-D Array}

Initialize an Array

int n[5]={3,5,4,1,6}; this will assign one value to every index.

or

int n[]={5,3,2,1,5,6}; this will also assign one value to one index, but this is used when we don't know actual numbers of inserting element.

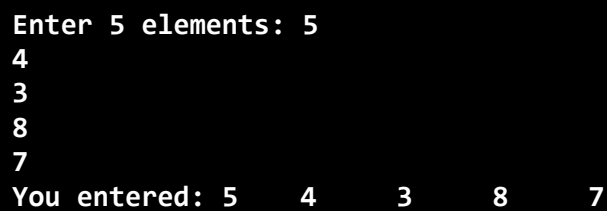
or

int n[2]=45; it will assign the value to specific index only.

Que 74: WAP to input an array of 5 elements and display them.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int n[5],x;
    clrscr();
    printf("Enter 5 elements: ");
    for(x=0;x<=4;x++)
    {
        scanf("%d",&n[x]);
    }
    printf("You entered: ");
    for(x=0;x<=4;x++)
    {
        printf("%d\t",&n[x]);
    }
    getch();
}
```

OUTPUT:

A screenshot of a terminal window with a black background and white text. The output shows the prompt 'Enter 5 elements: 5' followed by five lines of input: '4', '3', '8', '7'. Below these, the output line reads 'You entered: 5 4 3 8 7' with tabs between the numbers.

```
Enter 5 elements: 5
4
3
8
7
You entered: 5 4 3 8 7
```

Que 75: Input an array of 5 elements and display the greatest number of Array.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int n[5],x,max=0;
    clrscr();
    printf("Enter 5 elements: ");
    for(x=0;x<=4;x++)
    {
        scanf("%d",&n[x]);
    }
    max=n[0];
    for(x=0;x<=4;x++)
    {
        if(n[x]>max)
        {
            max=n[x];
        }
    }
}
```

```

        printf("Greatest number is: %d",max);
        getch();
    }

```

OUTPUT:

```

Enter 5 elements: 5
4
3
8
7
Greatest number is: 8

```

Que 76: Input an array of 5 elements and display sum of even and odd number separately.

```

#include<stdio.h>
#include<conio.h>
void main()
{
    int n[5],x,even=0,odd=0;
    clrscr();
    printf("Enter 5 elements: ");
    for(x=0;x<=4;x++)
    {
        scanf("%d",&n[x]);
    }
    max=n[0];
    for(x=0;x<=4;x++)
    {
        if(n[x]%2==0)
        {
            even=even+n[x];
        }
        else
        {
            odd=odd+n[x];
        }
    }
    printf("Sum of even number is :%d",even);
    printf("Sum of odd number is :%d",odd);
    getch();
}

```

OUTPUT:

```
Enter 5 elements: 5
4
3
8
7
Sum of even number is: 12
Sum of even number is: 15
```

2) 2-D Array or Multi-Dimensional Array

Declaring a 2-D Array

To declare an array in C, a programmer specifies the type of the elements and the number of elements required by an array as follows:

Syntax:

```
type arrayName[Size1][Size2].....[sizeN];
```

Initialize a 2-D Array

```
int n[3][3] = {
    {1,2,3} ,           // initializing for row indexed by 0
    {4,5,6} ,           // initializing for row indexed by 1
    {7,8,9}             // initializing for row indexed by 2
};                      // this will assign one value to every index.
```

or

```
int a[3][3] = {0,1,2,3,4,5,6,7,8};
```

For Example:


Que 77: WAP to input a 2-D array of 9 elements having 3 rows and 3 columns and display them in matrix form.

```
void main ()
{
    int a[3][3]={0,0,1},{4,1,2},{4,2,4}};
    int i,j;
    for(i=0;i<=2;i++)
    {
        for(j=0;j<=2;j++)
        {
            printf("%d",a[i][j]);
        }
        printf("\n");
    }
    getch();
}
```

Que 78: WAP to input a 2-D array of 9 elements having 3 rows and 3 columns and display them in matrix form.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a[3][3] ;
    int i, j;
    printf("Enter 9 elements: ");
    for(i=0;i<=2;i++)
    {
        for(j=0;j<=2;j++)
        {
            scanf("%d",&a[i][j]);
        }
    }
    printf("Matrix is :\n");
    for(i=0;i<=2;i++)
    {
        for(j=0;j<=2;j++)
        {
            printf("%d\t",a[i][j]);
        }
        printf("\n");
    }
    getch();
}
```

OUTPUT:



```
Enter 9 elements: 2
3
4
5
6
7
8
1
2
Matrix is:
2    3    4
5    6    7
8    1    2
```

Que 79: WAP to input a 2-D array of 9 (3x3) elements and display its transpose.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a[3][3] ;
    int i, j;
    printf("Enter 9 elements: ");
    for(i=0;i<=2;i++)
    {
        for(j=0;j<=2;j++)
        {
            scanf("%d",&a[i][j]);
        }
    }
    printf("Matrix is :");
    for(i=0;i<=2;i++)
    {
        for(j=0;j<=2;j++)
        {
            printf("%d ",a[i][j]);
        }
        printf("\n");
    }
    printf("Transpose of Matrix is :\n");
    for(i=0;i<=2;i++)
    {
        for(j=0;j<=2;j++)
        {
            printf("%d ",a[j][i]);
        }
        printf("\n");
    }
    getch();
}
```

OUTPUT:

```
Enter 9 elements: 2
3
4
5
6
7
8
1
2
Matrix is:
2 3 4
5 6 7
8 1 2
Transpose of matrix is:
2 5 8
3 6 1
4 7 2
```

Que 80: WAP to input a 2-D array of 9 elements and display sum of forward diagonal elements.

```
#include<stdio.h>
#include<conio.h>
void main ()
{
    int a[3][3] ;
    int i, j,sum;
    printf("Enter 9 elements: ");
    for(i=0;i<=2;i++)
    {
        for(j=0;j<=2;j++)
        {
            scanf("%d",&a[i][j]);
        }
    }
    printf("Matrix is :\n");
    for(i=0;i<=2;i++)
    {
        for(j=0;j<=2;j++)
        {
            printf("%d\t",a[i][j]);
        }
        printf("\n");
    }
    for(i=0,j=0;i<=2,j<=2;i++,j++)
    {
        sum=sum+a[i][j];
    }
    printf("Sum of forward diagonal elements is :%d",sum);
    getch();
}
```

OUTPUT:

```
Enter 9 elements: 2
3
4
5
6
7
8
1
2
Matrix is:
2 3 4
5 6 7
8 1 2
Sum of forward diagonal elements is :10
```

Que 81: WAP to input a 2-D array of 9 elements and display sum of forward diagonal elements.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a[3][3] ;
    int i, j,sum;
    printf("Enter 9 elements: ");
    for(i=0;i<=2;i++)
    {
        for(j=0;j<=2;j++)
        {
            scanf("%d",&a[i][j]);
        }
    }
    printf("Matrix is :\n");
    for(i=0;i<=2;i++)
    {
        for(j=0;j<=2;j++)
        {
            printf("%d\t",a[i][j]);
        }
        printf("\n");
    }
    for(i=0,j=2;i<=2,j>=0;i++,j--)
    {
        sum=sum+a[i][j];
    }
    printf("Sum of reverse diagonal elements is: %d",sum);
    getch();
}
```

OUTPUT:

```
Enter 9 elements: 2
3
4
5
6
7
8
1
2
Matrix is:
2 3 4
5 6 7
8 1 2
Sum of reverse diagonal elements is :18
```

Que 82: WAP to input a 2-D array of 9 elements and count total even and odd numbers in the matrix.

```
#include<stdio.h>
#include<conio.h>
void main ()
{
    int a[3][3] ;
    int i,j,ev=0,od=0;
    printf("Enter 9 elements: ");
    for(i=0;i<=2;i++)
    {
        for(j=0;j<=2;j++)
        {
            scanf("%d",&a[i][j]);
        }
    }
    printf("Matrix is :\n");
    for(i=0;i<=2;i++)
    {
        for(j=0;j<=2;j++)
        {
            printf("%d\t",a[i][j]);
            if(a[i][j]%2==0)
                ev=ev+1;
            else
                od=od+1;
        }
        printf("\n");
    }
    printf("Total even numbers are: %d\nTotal odd numbers are: %d",ev,od);
    getch();
}
```

OUTPUT:

```
Enter 9 elements: 2
3
4
5
6
7
8
1
2
Matrix is:
2 3 4
5 6 7
8 1 2
Total even numbers are: 5
Total odd numbers are: 4
```


Que 83: WAP to input a 2-D array of 9 elements and count total positive and negative numbers in the matrix.

```
#include<stdio.h>
#include<conio.h>
void main ()
{
    int a[3][3] ;
    int i,j,pos=0,neg=0;
    printf("Enter 9 elements: ");
    for(i=0;i<=2;i++)
    {
        for(j=0;j<=2;j++)
        {
            scanf("%d",&a[i][j]);
        }
    }
    printf("Matrix is:\n");
    for(i=0;i<=2;i++)
    {
        for(j=0;j<=2;j++)
        {
            printf("%d\t", a[i][j]);
            if(a[i][j]>0)
                pos=pos+1;
            else
                neg=neg+1;
        }
        printf("\n");
    }
    printf("Total positive numbers are: %d",pos);
    printf("\nTotal negative numbers are: %d",neg);
    getch();
}
```

OUTPUT:

```
Enter 9 elements: 2
3
4
5
-6
-7
8
1
-2
Matrix is:
2 3 4
5 -6 -7
8 1 -2
Total positive numbers are: 6
Total negative numbers are: 3
```

Que 84: WAP to input a 2-D array of 9 elements and display maximum and minimum number of matrixes.

```
#include<stdio.h>
#include<conio.h>
void main ()
{
    int a[3][3] ;
    int i,j,max,min;
    printf("Enter 9 elements: ");
    for(i=0;i<=2;i++)
    {
        for(j=0;j<=2;j++)
        {
            scanf("%d",&a[i][j]);
        }
    }
    printf("Matrix is :\n");
    max=a[0][0];
    min=a[0][0];
    for(i=0;i<=2;i++)
    {
        for(j=0;j<=2;j++)
        {
            printf("%d\t", a[i][j]);
            if(a[i][j]>max)
                max=a[i][j];
            if(a[i][j]<min)
                min=a[i][j];
        }
        printf("\n");
    }
    printf("Maximum number of the matrix is: %d",max);
    printf("\nMinimum number of the matrix is: %d",min);
    getch();
}
```

OUTPUT:

```
Enter 9 elements: 2
3
4
5
6
-7
8
1
12
Matrix is:
2 3 4
5 6 -7
8 1 12
Maximum number of the matrix is: 12
Minimum number of the matrix is: -7
```

Sorting

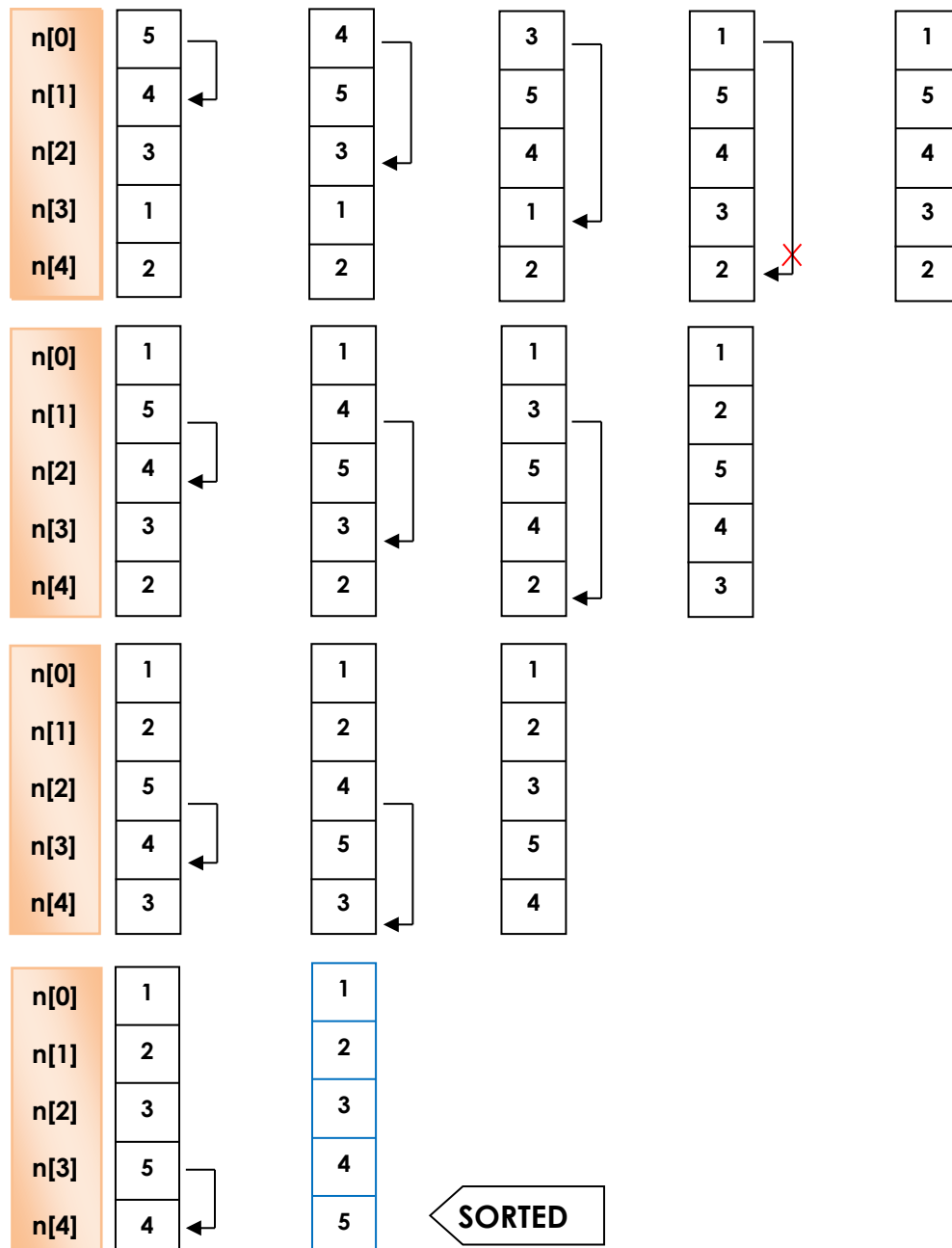
In C programming sorting is a technique to sort the data in ascending order or descending order. There are following sorting can be used to sort the data, these are –

- 1) Selection Sort
- 2) Bubble Sort

1) Selection sort

In this sorting, we compare first element with all other element, once base address has smallest element, we start sort for second element and then so on.

Let see following technique –



Que 85: WAP to input 5 element array and sort it using selection sort.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int n[5],i,j,temp;
    clrscr();
    printf("Enter five elements: ");
    for(i=0;i<=4;i++)
    {
        scanf("%d",&n[i]);
    }
    for(i=0;i<=4;i++)
    {
        for(j=i+1;j<=4;j++)
        {
            if(n[i]>n[j])
            {
                temp=n[i];
                n[i]=n[j];
                n[j]=temp;
            }
        }
    }
    printf("After sorting: \n");
    for(i=0;i<=4;i++)
    {
        printf("%d\t",n[i]);
    }
    getch();
}
```

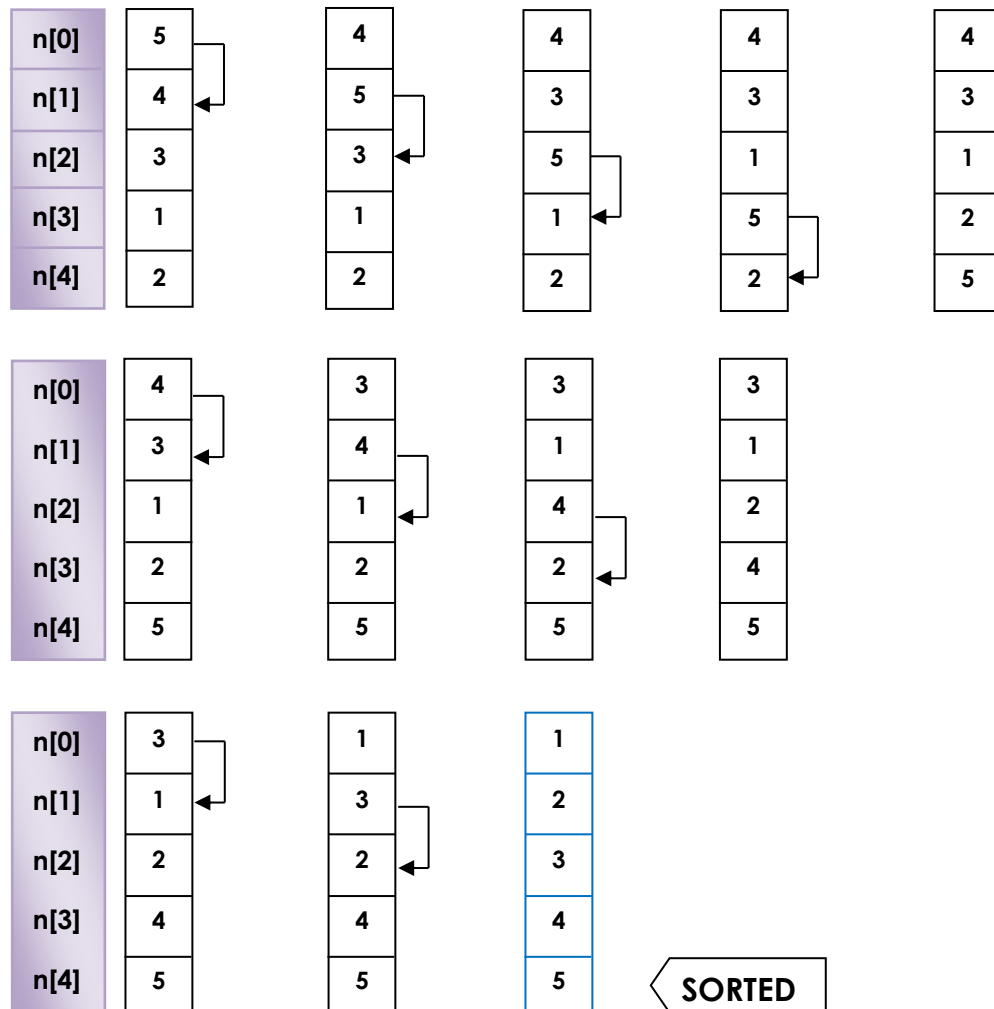
OUTPUT:

```
Enter 5 elements: 5
4
3
8
7
After sorting:
3    4    5    7    8
```

2) Bubble sort

In this type of sorting technique first the highest number of the array will be sort. In bubble sort, we take element of base address and then respectively check with next address's element if next element is smaller than they make interchange otherwise not. By doing this thing we get our array in sorted order.

Let see:



Que 86: WAP to input 5 element array and sort using Bubble sort.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int n[5],i,j,temp;
    clrscr();
    printf("Enter five elements: ");
    for(i=0;i<=4;i++)
    {
        scanf("%d",&n[i]);
    }
    for(i=0;i<=4;i++)
    {
        for(j=0;j<=3-i;j++)
        {
            if(n[j]>n[j+1])
            {
                temp=n[j];
                n[j]=n[j+1];
                n[j+1]=temp;
            }
        }
    }
    printf("After sorting: \n");
    for(i=0;i<=4;i++)
    {
        printf("%d\t",n[i]);
    }
    getch();
}
```

OUTPUT:

Enter 5 elements: 5

4

3

8

7

After sorting:

3

4

5

7

8

Searching

This is the process by which we can search an element from the group of elements. There are different methods of searching but let us deal two popular methods of searching and they are-

- 1) Linear Search
- 2) Binary Search

Linear Search

In this type of searching, each and every index of array is checked in a sequential manner until the desired element is found, once the element found the searching is break using break keyword. This technique is also known as **Sequential Searching** or **Simple Searching**.

Que 87: Input an array of 5 elements and search a specific element entered by user.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int n[5],x,s,f=0;
    clrscr();
    printf("Enter 5 elements: ");
    for(x=0;x<=4;x++)
    {
        scanf("%d",&n[x]);
    }
    printf("Enter an element to search: ");
    scanf("%d",&s);
    for(x=0;x<=4;x++)
    {
        if(s==n[x])
        {
            f=1;
            break;
        }
    }
    if(f==1)
    {
        printf("Value found at %d index",x+1);
    }
    else
    {
        printf("Not found !!");
    }
    getch();
}
```

OUTPUT:

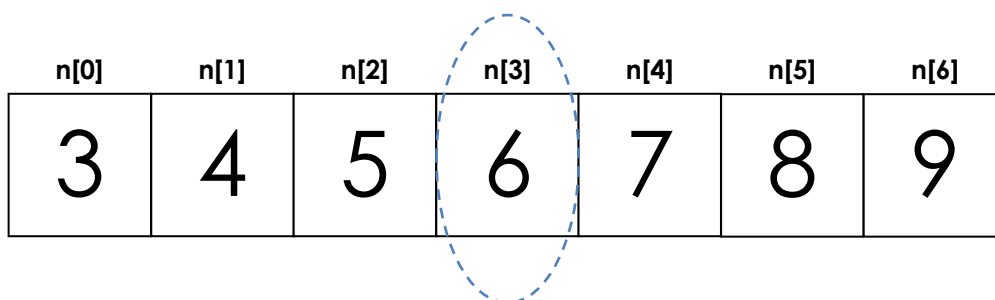
```
Enter 5 elements: 5
4
3
8
7
Enter an element to search: 4
Value found at 2 index
```

Binary Search

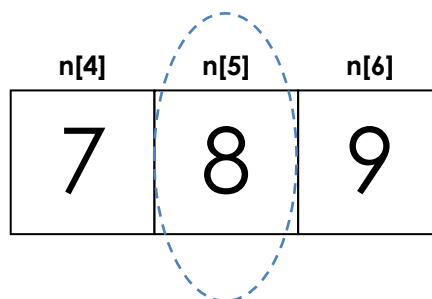
In Binary searching input array must be in sorted order or we have to write code for sort the array first than perform binary search. This technique is more effective than linear searching. This technique divides the array in two part and first check the searching item with the middle address's element if it is equal then searching successful otherwise it checks either searching item is greater than or less than middle element. If it is greater than middle element than it finds middle element of right side's element otherwise left side element.

Let consider we enter an array of 7 elements and user want search 8 from array, so binary search will perform following operation to search the desired element–

Step 1) First we have to find middle point and compare its value with searching value, if it isn't match than we've to check that searching item is greater or less than current middle element, in above case 8 is greater than 6 that is current middle value in array, so we've to find middle element for $n[4]$ to $n[6]$, then follow step 2.



Step 2) Now we have to find middle point for remaining 3 element and compare its value with searching value, we've to check that searching item is equal to current middle element, in above case 8 is equal to current middle value in array so the search is completed in just 2 steps [In above given case].



Que 88: Input an array of 5 elements and search a specific element entered by user using binary searching.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int n[5],x,y,temp,mid,frst,last,s,flag=0;
    clrscr();
    printf("Enter 5 elements: ");
    for(x=0;x<=4;x++)
    {
        scanf("%d",&n[x]);
    }
    for(x=0;x<=4;x++)
    {
        for(y=0;y<=3-x;y++)
        {
            if(n[y]>n[y+1])
            {
                temp=n[y];
                n[y]=n[y+1];
                n[y+1]=temp;
            }
        }
    }
    printf("Enter an element to search: ");
    scanf("%d",&s);
    frst=0;
    last=4;
    mid=0;
    while(frst<=last)
    {
        mid=(frst+last)/2;
        if(s==n[mid])
        {
            flag=1;
            printf("Value found at %d index",mid+1);
            break;
        }
        else
        {
            if(n[mid]>s)
            {
                last=mid-1;
            }
        }
    }
}
```

```

        else
        {
            frst=mid+1;
        }
    }
}
if(frst>last)
{
    printf("Not found !!");
}
getch();
}

```

OUTPUT:

```

Enter 5 elements: 5
4
3
8
7
Enter an element to search: 4
Value found at 2 index

```

Strings

The **string** in C programming language is a one-dimensional array of characters which is terminated by a null character '\0'. Thus, a null-terminated string contains the characters that comprise the string followed by a null as we learn in 1-D array of character.

The following declaration and initialization create a string consisting of the word "Hello". To hold the null character at the end of the array, the size of the character array containing the string is one more than the number of characters in the word "Hello".

One Dimensional Array of Characters: (Character Array or String)

String is a collection/group of characters, which is used to store text or string like name, address, any other text information.

Declaration: `char <arrayName>[<size>;`

Specifier: `%s`

Initialize an array of character: `char c[20]= {'H', 'e', 'l', 'l', 'o'};`

The memory representation is given following of above string –

n[0]	n[1]	n[2]	n[3]	n[4]	n[5]
H	e	l	l	o	\0

Actually, we do not place the null character at the end of a string. The C compiler automatically places the '\0' at the end of the string when it initializes the array. **Null character takes one byte in memory.** Let us try to print a string:

Input and Display a String

There are two methods –

1st Method: Using `%s` as **format specifier** in `scanf` function & **Display** string using a **for loop**.

Que 89: Input and display a string.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    char n[20];
    clrscr();
    printf("Enter a string: ");
    scanf("%s",n);
    for(x=0;n[x]!='\0';x++)
    {
        printf("You enter: \n%c",n[x]);
    }
    getch();
}
```

OUTPUT:

```
Enter a string: hello world
You enter: hello
```

2nst Method: Input and display a string using format specifier **%s** in **scanf** and **printf** function.

Que 90: Input and display a string.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    char n[20];
    int x;
    clrscr();
    printf("Enter a string: ");
    scanf("%s",n);
    printf("You enter: %s",n);
    getch();
}
```

OUTPUT:

```
Enter a string: hello world
You enter: hello
```

CHECKPOINT: In above both methods %s reads a string without space. If there is any space in string, it will treat the white-space as null character and terminate the string at null character.

To give or input a string with white space we must use –

%[^name of null character] instead of %s.

When user press the defined null character then string will be terminated. For Example:

Que 91: WAP to input and display a string.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    char n[20];
    int i;
    clrscr();
    printf("Enter a string: ");
    scanf("%[^\\n]",n);
    for(i=0;n[i]!='\\0';i++)
    {
        printf("You enter: %c",n);
    }
    getch();
}
```

OUTPUT:

```
Enter a string: hello world!!
You enter: Hello world!!
```

CHECKPOINT: In above program we take "\\n" as null character, so till we do not press Enter key from keyboard the string will be input continuously with white space also.

We can input and output a string with following functions too –

gets() - Read a string with space and terminate by Enter key.

puts() - Display or output a string.

Que 92: WAP to input a string using gets() function and display it using puts() function.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    char ch[20];
    clrscr();
    printf("Enter a string: ");
    gets(ch);
    printf("String is: ");
    puts(ch);
    getch();
}
```

OUTPUT:

```
Enter a string: Testing gets
String is: Testing gets
```

Que 93: WAP to input a string in lower case and display it in upper case.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    char ch[20];
    int a;
    clrscr();
    printf("Enter a string in lower case: ");
    gets(ch);
    for(a=0;ch[a]!='\0';a++)
    {
        if(ch[a]>=97 && ch[a]<=122)
        {
            ch[a]=ch[a]-32;
        }
    }
    printf("String in upper case :\n");
    puts(ch);
    getch();
}
```

OUTPUT:

```
Enter a string in lower case: my town sriganganagar
String in upper case: MY TOWN SRIGANGANAGAR
```

Que 94: WAP to input a string in upper case and display it in lower case.

```
#include<stdio.h>
#include<conio.h>
void main
{
    char ch[20];
    int a;
    clrscr();
    printf("Enter a string in upper case: ");
    gets(ch);
    for(a=0;ch[a]!='\0';a++)
    {
        if(ch[a]>=65 && ch[a]<=90)
        {
            ch[a]=ch[a]+32;
        }
    }
    printf("String in lower case :\n");
    puts(ch);
    getch();
}
```

OUTPUT:

```
Enter a string in upper case: MY TOWN SRIGANGANAGAR
String in lower case: my town sriganganagar
```

Que 95: WAP to input a string and count total number of characters with white space.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    char ch[20];
    int a,count;
    clrscr();
    printf("Enter a string: ");
    gets(ch);
    for(a=0;ch[a]!='\0';a++)
    {
        count=count+1;
    }
    printf("Total characters are: %d in - ",count);
    puts(ch);
    getch();
}
```

OUTPUT:

```
Enter a string: my town sriganganagar
Total characters are 21 in - my town sriganganagar
```

Que 96: WAP to input a string and count total number of vowels.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    char ch[20];
    int a,count;
    clrscr();
    printf("Enter a string: ");
    gets(ch);
    for(a=0;ch[a]!='\0';a++)
    {
        if(n[a]=='a' || n[a]=='A' || n[a]=='e' || n[a]=='E' || n[a]=='i' ||
           n[a]=='I' || n[a]=='o' || n[a]=='O' || n[a]=='u' || n[a]=='U')
        {
            count=count+1;
        }
    }
    printf("Total vowels are: %d in - ",count);
    puts(ch);
    getch();
}
```

OUTPUT:

```
Enter a string: my town sriganganagar
Total vowels are 6 in - my town sriganganagar
```

Que 97: WAP to input a string and copy it into another string.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    char ch[20],an[20];
    int a;
    clrscr();
    printf("Enter a string: ");
    gets(ch);
    for(a=0;ch[a]!='\0';a++)
    {
        an[a]=ch[a];
    }
}
```

```

        an[a]='\0';
        printf("You enter:\n");
        puts(ch);
        printf("Copied string:\n");
        puts(an);
        getch();
    }

```

OUTPUT:

```

Enter a string: hello world
You enter:
hello world
Copied string:
hello world

```

Que 98: WAP to input a string and count total words, total space, total lines and total characters.

```

#include<stdio.h>
#include<conio.h>
void main()
{
    char ch[200];
    int a,c=0,w=0,s=0,l=0;
    clrscr();
    printf("Press # to stop\nEnter a string: ");
    scanf("%[^#]",ch);
    for(a=0;ch[a]!='\0';a++)
    {
        if(ch[a]==' ')
        {
            s++;
        }
        if(ch[a]=='\n')
        {
            l++;
        }
        w=s+1;
    }
    printf("Total words are: %d",w);
    printf("Total characters are: %d",a-1);
    printf("Total space are: %d",s);
    printf("Total lines are: %d",l);
    getch();
}

```


OUTPUT:

```
Press # to stop
Enter a string: hello world
Total words are: 2
Total characters are: 11
Total spaces are: 1
Total lines are: 1
```

Library functions for Strings:

In C programming there some inbuilt/ library functions are available that can be used in manipulation of string. See the table given below to know about the string functions-

Function	Purpose
<code>strcpy(s1,s2)</code>	Copies string s2 into string s1.
<code>strcmp(s1,s2)</code>	Returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 if s1>s2.
<code>strcat(s1,s2)</code>	Concatenates string s2 onto the end of string s1.
<code>strlen(s1)</code>	Returns the length of string s1.

Table: Strings library functions

CHECKPOINT: if we want to try these functions in a 'C' program then we must include string.h header file at the header part of program.

Que 99: WAP to copy a string to another string using strcpy() function.

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

void main()
{
    char n[20],m[20];
    clrscr();
    printf("Enter a string: ");
    gets(n);
    strcpy(m,n);
    printf("You Enter: ");
    puts(n);
    printf("After copy another string is :");
    puts(m);
    getch();
}
```

OUTPUT:

```
Enter a string: hello students
You Enter: hello students
After copy anther string is: hello students
```

Que 100: WAP to count length of a string using strlen() function and display length.

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

void main()
{
    char n[20];
    int len=0;
    clrscr();
    printf("Enter a string: ");
    gets(n);
    len=strlen(n);
    printf("You Enter: ");
    puts(n);
    printf("Length of string (in Characters) is: %d",len);
    getch();
}
```

OUTPUT:

```
Enter a string: hello students
You Enter: hello students
Length of string (in Characters) is: 14
```

Que 101: WAP to add a string with another string using strcat() function.

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

void main()
{
    char n[20],m[20];
    clrscr();
    printf("Enter first string: ");
    gets(n);
    printf("Enter another string: ");
    gets(m);
    strcat(n,m);
    printf("After adding, first string is: ");
    puts(n);
    getch();
}
```

OUTPUT:

```
Enter a string: hello
Enter another string: World
After adding, first string is: helloWorld
```

Que 102: WAP to compare two strings using strcmp() function and display which is greater.

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

void main()
{
    char n[20],m[20];
    int c;
    clrscr();
    printf("Enter first string: ");
    gets(n);
    printf("Enter another string: ");
    gets(m);
    c=strcmp(n,m);
    if(c>0)
    {
        printf("First string is greater, string is- ");
        puts(n);
    }
    else if(c<0)
    {
        printf("Second string is greater, string is-");
        puts(m);
    }
    else
    {
        printf("Both strings are equal, strings are-");
        puts(n);
        puts(m);
    }
    getch();
}
```

OUTPUT:

```
Enter first string: hello
Enter another string: Hello World
Second string is greater, string is- Hello World
```

Pointer

In 'C' programming pointer is a variable whose value is the address of another variable. In simple words we can say that a pointer is a special type of variable that hold the address of another simple variable. We must declare a pointer before we use it to store any variable address.

Some C programming tasks are performed more easily with pointers, such as dynamic memory allocation, cannot be performed without using pointers. So, it becomes necessary to learn pointers to become a perfect C programmer.

As we know, every variable is a name memory location and every memory location have its address defined which can be accessed using ampersand (&) operator, which denotes an address in memory. Consider the following example, which will print the address of the variables defined:

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a;
    printf("Address of a variable: %u\n", &a);
    getch();
}
```

OUTPUT:

```
Address of a variable: 65524
```

Single Pointer:

A pointer variable takes 2 Bytes in memory, because the pointer variable holds the address of a variable and the address of memory location always in unsigned integer. An unsigned integer takes 2 Bytes; therefore, pointer occupies 2 Bytes.

Declaration:

```
data_type *variable_name;
```

For example:

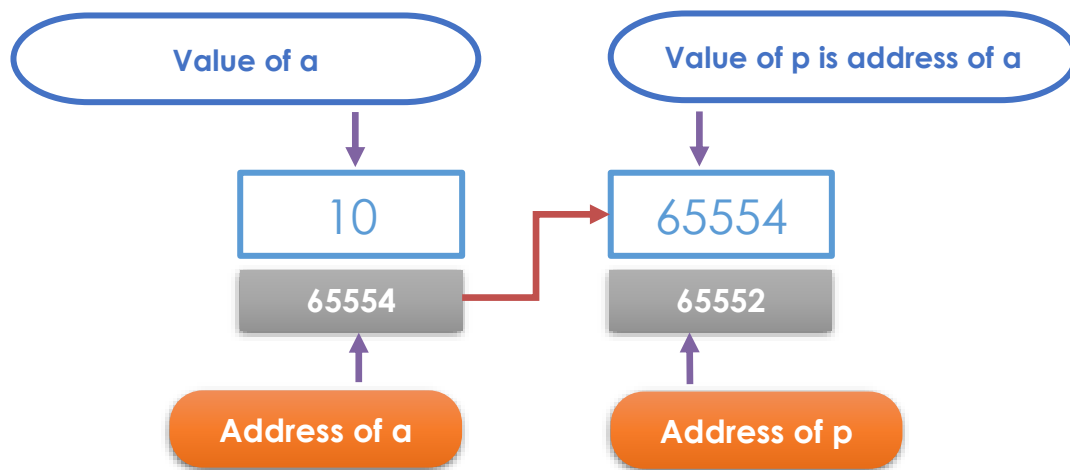
```
int *p, *q;
```

While learning about pointer we have to concentrate on two main things about pointer. These are –

&	-	address of <variablename>
*	-	value of <variablename>

For example: Consider there we have an integer variable and another is pointer variable. Let see following explanation –

```
int a=10;
int *p;
p=&a;
```



```

a      =    10
&a     =    65554
p      =    65554
&p     =    65552
*p     =    10

```

Que 103: WAP to describe pointers.

```

#include<stdio.h>
#include<conio.h>
void main()
{
    int a;
    int *p;
    clrscr();
    printf("Enter a number for variable a: ");
    scanf("%d",&a);
    p=&a;
    printf("The value of a is: %d",a);
    printf("\nThe address of a is: %u",&a);
    printf("\n The value of p(holds the address of a) is: %u",p);
    printf("\n The address of p is: %u",&p);
    printf("\n The value of *p(value on address that holds by p) is: %d",*p);
    getch();
}

```

OUTPUT:

```

Enter a number for variable a: 25
The value of a is: 25
The address of a is: 65524
The value of p (holds the address of a) is: 65524
The address of p is: 65522
The value of *p(value on address that holds by p) is: 25

```

Double Pointer:

In 'C' programming double pointer holds the address of single pointer.

Declaration:

data type ******variable name;

For example:

```
int **p, **q;
```

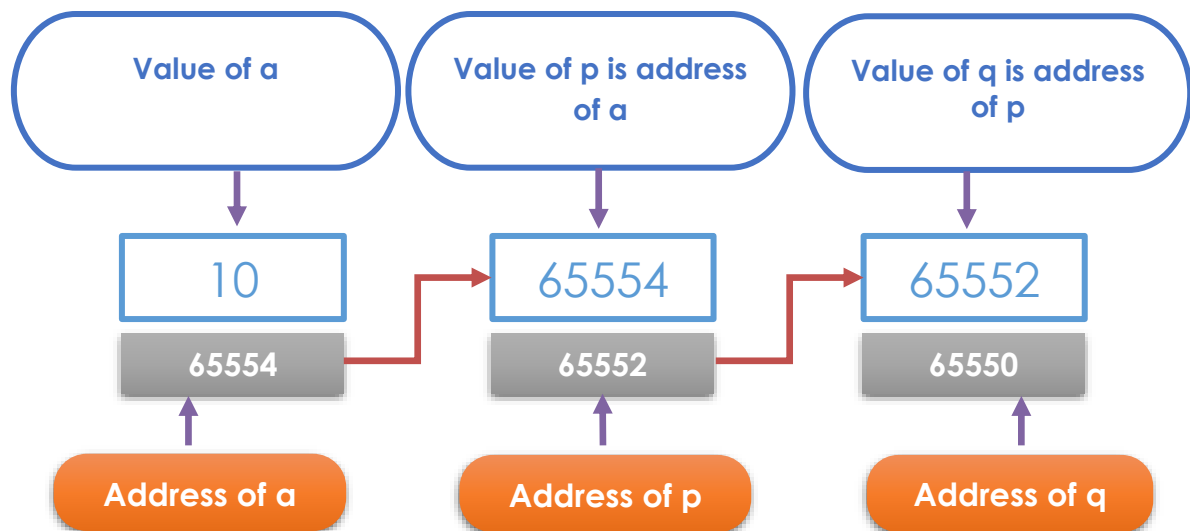
For example: Consider there we have an integer variable and another is pointer variable. Let see following explanation –

```
int a=10;
```

```
int *p, **q;
```

```
p=&a;
```

```
q=&p;
```



a	=	10
&a	=	65554
p	=	65554
&p	=	65552
*p	=	10
q	=	65552

Que 104: WAP to describe single pointers and double pointers.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a;
    int *p,**q;
    clrscr();
    printf("Enter a number for variable a: ");
    scanf("%d",&a);
    p=&a;
    q=&p;
    printf("The value of a is: %d",a);
    printf("\nThe address of a is: %u",&a);
    printf("\nThe value of p(holds the address of a) is: %u",p);
    printf("\nThe address of p is: %u",&p);
    printf("\nThe value of *p(value on address that holds by p) is: %d",*p);
    printf("\nThe value of q(holds the address of p) is: %u",q);
    printf("\nThe value of *q(value on address that holds by q) is: %d",*q);
    printf("\nValue of **q->address of p-> (value of address that p holds) is:
        %d",**q);
    getch();
}
```

OUTPUT:

```
Enter a number for variable a: 25
The value of a is: 25
The address of a is: 65524
The value of p(holds the address of a) is: 65524
The address of p is: 65522
The value of *p (value on address that holds by p) is: 25
The value of q (holds the address of p) is: 65522
The value of *q (value on address that holds by q) is: 65524
Value of **q->address of p-> (value of address that p holds) is: 25
```

Functions

In 'C' programming function is a group or block of statements that together perform a defined task. Every C program has at least one function, called **main()**, and we can define additional functions in a C program.

We can divide up our code into separate functions for performing specific work or operation by defined instruction of a function, so each function performs a specific task.

There are two type of functions in 'C' programming –

- 1) Library/In-built/pre-define functions
- 2) User defined functions

Library/In-built/pre-define functions

'C' programming provides some various type of in-built function so we can easily perform a task with an available function in C. Some in-built functions are - **clrscr()**, **getch()**, **printf()**, **scanf()**, **gets()**, **puts()**, **strcpy()**, **strcmp()**, **strlen()** etc. we were used earlier in our programs.

User defined function

Another types of functions in 'C' are user define function. 'C' provides facility to programmer to define their own function in a 'C' program to execute a block of statements to perform a specific task.

A user define function have following 3 main things we've to know –

- a) Declaration/Prototyping
- b) Definition
- c) Calling

a) Declaration of a function

A function **declaration** tells the compiler about a function's name, return type, and parameters. This term is also known as **prototyping** of a function.

Syntax:

```
<return_type> <function_name> (<list of arguments/parameters>);
```

For example:

```
void sum();      }
```

declaration of a function

b) Definition of a function

The body part that specifies the function working is called definition of a function. We can say that the block of statement that defining the task of a function is called definition of a function.

For example:

```
void sum()
{
    statement(s);
}
```

declaration of a function
Body part of a function.

Defining a Function

Defining a function means defining header of function and its body. Every function has four main parts in 'C' programming, these are –

Return Type

A function may return a value. The return type is the data type of the value that function returns. Some functions perform the desired operations without returning a value. In this case, the return type is the keyword `void`.

Function Name

This is the actual name of the function. The function name and the parameter list make a function complete.

Parameters

A parameter is like a placeholder. When a function is invoked, if we pass a value to the parameter. This value is referred to as actual parameter or argument. The parameter list refers to the type, order, and numbers of the parameters of a function that function passes at the time of invoking. Parameters are optional; a function may contain no parameters.

Function Body

The function body contains a collection of statements that defines what the function does.

c) Calling a function

While creating a 'C' function, we give a definition of what the function has to do. To use a function, we will have to call that function to perform the defined task. When a program calls a function, program control is transferred to the called function. A called function performs defined task. If a function has a return type then function will be return control back to the main program function return to main program with **return** keyword placed at end of the function statements and before end-braces.

To call a function, we simply need to pass the required parameters (**if function have parameter list**) along with function name, and if function returns a value, then you can store returned value in the main program.

For example:

```
sum();
```

Properties of Function:

- A function can call any number of times.
- By default, a function returns a value of type `int`.
- A function can calls any other function.
- A function is always defined outside of another function.
- A function can call itself and this process is called **recursion**.
- A function cannot define more than one time in a program.
- When function returns nothing then its return type must be `void`.

Types of Functions

In 'C' programming there are 4 types of user define function –

- i) No Argument and No Return
- ii) With Argument and No Return
- iii) No Argument and with Return
- iv) With Argument and with Return

i) No Argument and No Return:

A function that have `void` as return type, this indicates function does not return any value and the parameter list is empty or `void`, this indicates that function does not receive any arguments/parameters. This type of nature function is known as No Argument and No return.

Declaration:

```
void sum();           or      void sum(void);
```

Definition:

```
void sum()
{
    statement(s);
}
```

Calling:

```
sum();
```

For Example:

Que 105: Write a function avg(), that display the average of 3 numbers.

```
#include<stdio.h>
#include<conio.h>
void avg(void);
void avg(void)
{
    int a,b,c;
    printf("Enter three numbers respectively by pressing enter: ");
    scanf("%d%d%d",&a,&b,&c);
    printf("Average is :%d", (a+b+c)/3);
}

void main()
{
    clrscr();
    avg();
    getch();
}
```

NOTE: When a function invoked then control of program refers to function definition.

Que 106: Write a function factorial() having no arguments no return type to display factorial of a number entered by user.

```
#include<stdio.h>
#include<conio.h>
void factorial()
{
    int n,f=1,m;
    printf("Enter a number: ");
    scanf("%d",&n);
    m=n;
    while(n>0)
    {
        f=f*n;
        n--;
    }
}
```

```

        printf("Factorial of %d is: %d",m,f);
    }

void main()
{
    clrscr();
    factorial();
    getch();
}

```

OUTPUT:

```

Enter a number: 5
Factorial of 5 is: 120

```

ii) With argument and No Return

A function that have `void` as return type, this indicates function does not return any value and having parameter list, this indicates that function will receive arguments/parameters that sends at the time of calling. This type of nature function is known as With Argument and No return.

Declaration:

```
void sum(int a, int b.....n);
```

Definition:

```

void sum(int a, int b)
{
    statement(s);
}

```

Calling:

```

sum(5,6);
or

int a=5, int b=4;
sum(a,b);

```

For Example:

Que 107: Write a function avg(), that display the average of 3 numbers.

```

#include<stdio.h>
#include<conio.h>
void avg(int , int , int );
void avg(int a, int b, int c)
{
    printf("Average is :%d", (a+b+c)/3);
}

void main()
{
    int a=4,b=5,c=3;
    clrscr();
    avg(a,b,c);
    getch();
}

```

STEP-4

STEP-4.1

STEP-1

STEP-2

STEP-3

STEP-5

OUTPUT:

```
Average is: 4
```

Que 108: Write a function swap() having arguments and no return type to display swapped number entered by user.

```
#include<stdio.h>
#include<conio.h>
void swap(int n, int m)
{
    int x;
    x=n;
    n=m;
    m=x;
    printf("After swapping first number is: %d",n);
    printf("\nAnd second number is: %d",m);
}

void main()
{
    int n,m;
    clrscr();
    printf("Enter first number: ");
    scanf("%d",&n);
    printf("Enter second number: ");
    scanf("%d",&m);
    swap(n,m);
    getch();
}
```

OUTPUT:

```
Enter first number: 4
Enter second number: 5
After swapping first number is: 5
And second number is: 4
```

iii) No argument and With Return

A function that has a return type, it means the function will return a value every time when it is invoked and having no parameter list, this indicates that function will not receive arguments/parameters. This type of nature function is known as No Argument and with return.

CHECKPOINT: If function has return type then return keyword with a value or variable must be placed before end-braces.

Declaration:

```
int sum();
```

Definition:

```
int sum()
{
    statement(s);
    return <constant or variable>;
}
```

Calling:

```
sum();
```

For Example:

Que 109: Write a function avg(), that display the average of 3 numbers.

```
#include<stdio.h>
#include<conio.h>
int avg();
STEP-4 int avg()
{
STEP-4.1 int a=2,b=3,c=4;
STEP-4.2 retrun (a+b+c)/3;
}
STEP-1 void main()
{
STEP-2 clrscr();
STEP-3,5 printf("Average is: %d",avg());
STEP-6 getch();
}
```

OUTPUT:

```
Average is: 3
```

Que 110: Write a function factorial() having no arguments and with return type to display factorial of a number entered by user.

```
#include<stdio.h>
#include<conio.h>
int factorial()
{
    int n, f=1,m;
    printf("Enter a number: ");
    scanf("%d",&n);
    m=n;
    while(n>0)
    {
        f=f*n;
        n--;
    }
    return f;
}

void main()
{
    int x;
    clrscr();
    x=factorial();
    printf("Factorial is: %d",x);
    getch();
}
```

OUTPUT:

```
Enter a number: 5
Factorial is: 120
```

iv) With argument and With Return

A function that has a return type, it means the function will return a value every time when it is invoked and also having parameter list, this indicates that function will receive arguments/parameters that will send/throw at the time of invoking. This type of nature function is known as With Argument and with return.

Declaration:

```
int sum(int a, int b, int c.....n);
```

Definition:

```
int sum(int a, int b)
{
    statement(s);
    return <constant or variable>;
}
```

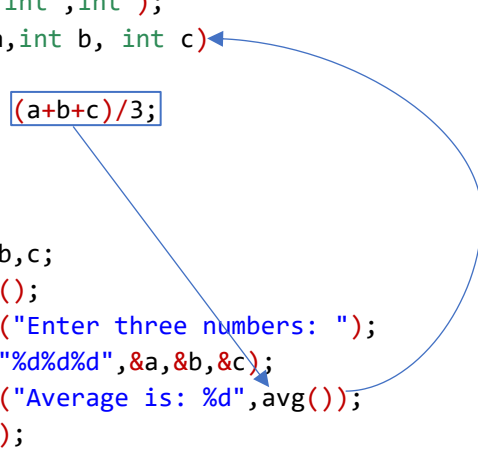
Calling:

```
printf("sum is: %d",sum(5,6));
```

For Example:

Que 111: Write a function avg(), that display the average of 3 numbers.

```
#include<stdio.h>
#include<conio.h>
int avg(int ,int ,int );
STEP-6 int avg(int a,int b, int c)
{
STEP-7.1     return (a+b+c)/3;
}
STEP-1 void main()
{
STEP-2     int a,b,c;
STEP-3     clrscr();
STEP-4     printf("Enter three numbers: ");
STEP-5, 7.2 scanf("%d%d%d",&a,&b,&c);
STEP-8     printf("Average is: %d",avg());
            getch();
}
```



Que 112: Write a function factorial() having arguments and with return to display factorial of a number entered by user.

```
#include<stdio.h>
#include<conio.h>
int factorial(int a)
{
    int f=1,m;
    m=n;
    while(n>0)
    {
        f=f*n;
        n--;
    }
    return f;
}
```

```

void main()
{
    int x,n;
    clrscr();
    printf("Enter a number: ");
    scanf("%d",&n);
    x= factorial(n);
    printf("Factorial is: %d",x);
    getch();
}

```

OUTPUT:

```

Enter a number: 5
Factorial is :120

```

Passing arguments to a Function

In above examples we have learned passing arguments to a function. Now we will learn 2 different ways to pass arguments to a function. These are-

- 1) Call by Value
- 2) Call by Reference

1) Call by Value

When arguments are passed using **call by value** then the **formal arguments** do not affect the **actual arguments** because **function passes the value of the arguments to its definition**. Let see the following example –

Que 113: WAP to make a function name swap and pass value by call by value.

```

#include<stdio.h>
#include<conio.h>
void swap(int a, int b)
{
    int c;
    c=a;
    a=b;
    b=c;
    printf("After swapping A is: %d\nB is: %d",a,b);
}
void main()
{
    int a,b;
    printf("Enter two numbers: ");
    scanf("%d%d",&a,&b);
    printf("Before swapping A is: %d\nB is: %d",a,b);

    swap(a,b);

    printf("[In main() after invoking swap] A is: %d\nB is: %d",a,b);
    getch();
}

```

Formal arguments

Actual arguments

OUTPUT:

```
Enter two number: 4
6
Before swapping A is: 4
B is: 6
After swapping A is: 6
B is: 4
[In main() after invoking swap] A is: 4
B is 6
```

As we can see in the output of above program the value of A and B interchanged just inside the swap() function and these changes never affects the actual argument that passes to the function, it is just because of passing values of the arguments. When we print the value of A and B after invoking swap function in main() then it will show the original value that we had entered.

2) Call by Reference

When arguments are passed using **call by reference**, the formal arguments affect the **actual arguments** because **function passes the address of the arguments** to its definition, and when we pass address of an argument then the calculation or manipulation of information performed on basis of address of the arguments. Let see the following example –

Que 114: WAP to make a function name swap and pass value by call by reference.

```
#include<stdio.h>
#include<conio.h>
void swap(int *a, int b)
{
    int c;
    c=*a;
    *a=*b;
    *b=c;
    printf("[In swap()]After swapping Values-\nA is: %d\nB is: %d",a,b);
}
void main()
{
    int a,b;
    printf("Enter two numbers: ");
    scanf("%d%d",&a,&b);
    printf("[In main()]Before swapping Values-\n A is: %d\nB is: %d",a,b);
    swap(&a,&b);
    printf("[In main() after invoking swap]\nA is: %d\nB is: %d",a,b);
    getch();
}
```

Formal arguments

Actual arguments

OUTPUT:

```
Enter two numbers: 4
6
[In main()]Before swapping Values-
A is: 4
B is: 6
[In swap()]After swapping Values-
A is: 6
B is: 4

[In main() after invoking swap]
A is: 6
B is: 4
```

As we can see in the output of above program the values of A and B interchanged by the swap() function and these changes affects the actual argument that passes to the function from main(), it is just because of passing addresses of the arguments. When we print the value of A and B after invoking swap function in main() then it will show the swapped values by the swap function because this function receive the address of both arguments and swap the address to one another.

Pointer and String

In C programming the memory is allocated in two methods –

- 1) **Compile Time (Static memory allocation)**
- 2) **Run Time (Dynamic memory allocation)**

1) Compile Time

Whenever we define an array of character actually, we are allocating static memory. In other words, we can say that by defining character array we reserved a limited memory in storage, we cannot change this size during execution of a program. For Example:

```
char n[20];
```

In above example we have define a character array of 20 elements having 20 Bytes in memory, we cannot change the size of array during execution.

2) Run Time

We know that the pointer is used for its nature of dynamic behavior. By using pointer when we input a string then the memory is not allocated at the time of compilation, because pointer allocate the memory at run time.

Let see following example for better figuring out the concept –

Que 115: WAP to input a string using pointer.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    char *n;
    char m[20];
    clrscr();
    printf("Enter a string for pointer: ");
    gets(n);
    printf("Enter a string for array: ");
    gets(m);
    printf("String is: %s\nSize of the pointer is: %d Bytes.",n,sizeof(n));
    printf("String is: %s\nSize of the array is: %d Bytes.",m,sizeof(m));
    getch();
}
```

OUTPUT:

```
Enter a string for pointer: Hello pointer
Enter a string for array: Hello array
Pointer string is: Hello pointer
Size of the pointer is: 2 Bytes.
Array string is: Hello array
Size of the pointer is: 20 Bytes.
```

Array to Function

Just like variables, array can also be passed to a function as an argument. Same as variable we can pass the array to a function by following 2 methods-

- 1) Call by value
- 2) Call by reference

1) Passing Array using Call by Value

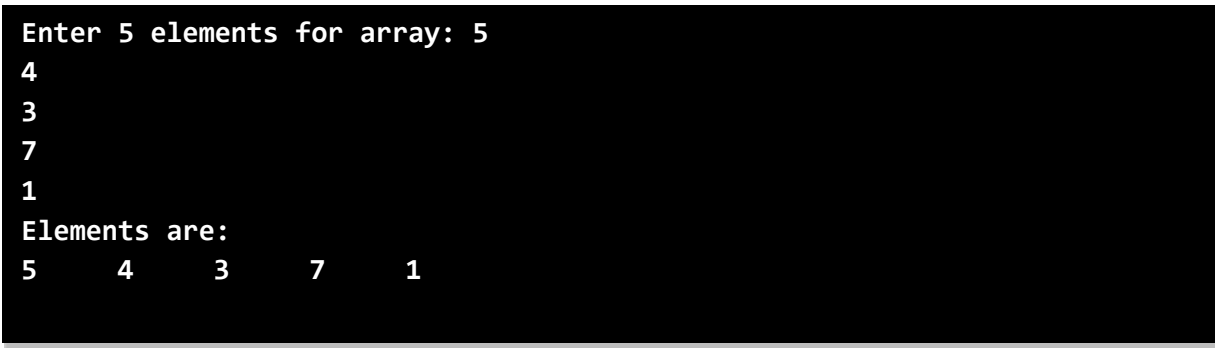
As we already know in this type of function call, the actual parameter is copied the values to the formal parameters.

For Example:

Que 116: Pass values of an array to a function and display them using an UDF.

```
#include<stdio.h>
#include<conio.h>
void disp(int n[])
{
    int i;
    printf("Elements are:\n");
    for(i=0;i<=4;i++)
    {
        printf("%d\t",n[i]);
    }
}
void main()
{
    int i,n[5];
    clrscr();
    printf("Enter 5 elements for array: ");
    for(i=0;i<=4;i++)
    {
        scanf("%d",&n[i]);
    }
    disp(n);
    getch();
}
```

OUTPUT:



```
Enter 5 elements for array: 5
4
3
7
1
5
Elements are:
5    4    3    7    1
```

2) Passing Array using Call by Reference

When we pass the address of an array while calling a function then this is called function call by reference. When we pass an address as an argument, the function declaration should have a pointer as a parameter to receive the passed address.

For Example:

Que 117: Pass values of an array to a function and display them using an UDF.

```
#include<stdio.h>
#include<conio.h>
void sortArray(int *v)
{
    int i,j,x;
    for(i=0;i<=4;i++)
    {
        for(j=0;j<=3-i;j++)
        {
            if(v[j]>v[j+1])
            {
                x=v[j];
                v[j]=v[j+1];
                v[j+1]=x;
            }
        }
    }
}
void main()
{
    int n[5],i;
    clrscr();
    printf("Enter 5 elements for array: ");
    for(i=0;i<=4;i++)
    {
        scanf("%d",&n[i]);
    }
    sortArray(n);
    printf("After sorting from the Function Array:\n");
    for(i=0;i<=4;i++)
    {
        printf("%d\t",n[i]);
    }
    getch();
}
```

OUTPUT:

```
Enter 5 elements for array: 4
6
2
7
1
After sorting from the Function Array:
1    2    4    6    7
```

CHECKPOINT: This is not mandatory to use '['] signs with an array variable in a function argument list while we using array as call by reference. Sometimes when we're using pointer array to a function then we will see the array notation as given following–

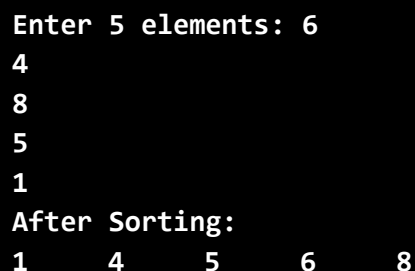
$n[x] \Rightarrow (n+x)$

The reason behind using this expression is that we cannot access the value of array using [].

Que 118: Write a function sort() that sort the 5 element array entered by user using pointers.

```
#include<stdio.h>
#include<conio.h>
void sort(int *n)
{
    int s,x,y;
    for(x=0;x<=4;x++)
    {
        for(y=x+1;y<=4;y++)
        {
            if(*(n+x)>*(n+y))
            {
                s=*(n+x);
                *(n+x)= *(n+y);
                *(n+y)=s;
            }
        }
    }
}
void main()
{
    int n[5],x,y;
    clrscr();
    printf("Enter 5 elements: ");
    for(x=0;x<=4;x++)
    {
        scanf("%d",&n[x]);
    }
    sort(n);
    printf("After Sorting:\n");
    for(x=0;x<=4;x++)
    {
        printf("%d\t",*(n+x));
    }
    getch();
}
```

OUTPUT:



```
Enter 5 elements: 6
4
8
5
1
After Sorting:
1    4    5    6    8
```

Que 119: Write a function max() which return the greatest elements from the pointer array.

```
#include<stdio.h>
#include<conio.h>
int max(int *n)
{
    int s,x;
    s=n[0];
    for(x=0;x<=4;x++)
    {
        if(*(n+x)>s)
        {
            s=*(n+x);
        }
    }
    return s;
}
void main()
{
    int n[5],x;
    clrscr();
    printf("Enter 5 elements: ");
    for(x=0;x<=4;x++)
    {
        scanf("%d",&n[x]);
    }
    printf("Greatest element is: %d",max(n));
    getch();
}
```

OUTPUT:

```
Enter 5 elements: 6
4
8
5
1
Greatest element is: 8
```

Structure

Structure is a group of different data items. In array we can store only similar type of data or information. We cannot store a complete record in an array.

In 'C' programming arrays allow us to define variables that can hold several data items of the same kind but **structure** is another **user defined data type** available in C programming, which allows us to combine data items of different kinds.

As we mention above Structures are used to represent a record. Suppose we want to keep track of our books in a library. For example –

- Title
- Author
- Subject
- Book ID

So, store these kinds of information we use structure.

Defining a Structure

To define a structure, we use the **struct** statement. The **struct** statement defines a new data type, with more than one member for your program. Each member definition is a normal variable definition, such as **int** i; or **float** f; or any other valid variable definition. The syntax of the **struct** statement is following –

Syntax:

```
Struct <structure tag name>
{
    member definition(s);
} <one or more structure Objects>;
```

defining an object at this scope
is treated as global declaration;

Defining a Structure Variable/Object

At the end of the structure's definition, **before the final semicolon, we can specify one or more structure variables but it is optional**. We can declare structure variable according to following methods –

- 1) As Global Scope
- 2) As Local Scope

1) As Global Scope

When we are defining variable of a structure as global scope then it is accessible to all functions (main() and other user-defined function) equally.

Syntax:

Outside of all functions	Before the final semicolon of structure –
<pre>struct <tag name> { members; }; struct <tag name> <variable name>;</pre>	<pre>struct <tag name> { members; }struct <tag name> <variable name>;</pre>

2) As Local Scope

If we define a structure variable as local scope or in a function body then it will not be accessible outside of that function.

Syntax:

```
struct <tag name>
{
    members;
}
void main()
{
    struct <tag name> <variable name>;
}
```

Accessing Structure Data Member

To access any member of a structure, we use the **member access operator (.)**. The member access operator is coded as a period between the structure variable name and the structure member that we wish to access. You would use **struct** keyword to define variables of structure type. Following is the example to explain usage of structure:

For Example:

Que 120: WAP to define structure for book information and input the information about book then display it.

```
#include<stdio.h>
#include<conio.h>
struct Books
{
    char t[50];
    char au[50];
    char s[100];
    int id;
};
void main()
{
    struct Books B;
    clrscr();
    printf("Enter book Title: ");
    gets(B.t);
    fflush(stdin);
    printf("Enter Author: ");
    scanf("%s",B.au);
    printf("Enter Subject: ");
    scanf("%s",B.s);
    printf("Enter Book ID: ");
    scanf("%d",&B.id);
    printf("Book Details are -");
    printf("Book Title: %s\n", B.t);
    printf("Book Author: %s\n", B.au);
    printf("Book Subject: %s\n", B.s);
    printf("Book ID: %d\n", B.id);
    getch();
}
```


OUTPUT:

```
Enter Book Title: C Language
Enter Author: Unknown
Enter Subject: C
Enter Book ID: 231

Book Details are following -
Book Title: C Language
Book Author: Unknown
Book Subject: C
Book ID: 231
```

Que 121: WAP to define structure for student information and input the information about student then displays it.

```
#include<stdio.h>
#include<conio.h>
struct stud
{
    char n[50];
    char s[1];
    int id;
};
struct stud S;
void main()
{
    clrscr();
    printf("Enter Name of student: ");
    gets(S.n);
    fflush(stdin);
    printf("Enter Section of student: ");
    gets(S.s);
    printf("Enter ID of student: ");
    gets(S.id);
    printf("Student Details are -");
    printf("Student Name: %s\n", S.n);
    printf("Section: %s\n", S.s);
    printf("Student ID: %s\n", S.id);
    getch();
}
```

OUTPUT:

```
Enter Name of student: Naresh
Enter Section of Student: B
Enter ID of Student: 454
Student Details are -
Student Name: Naresh
Section: B
Student ID: 454
```

Array of Structure

In C programming we can define **array of structure** as we have defined array of in-built data type.

For example

Que 122: WAP to define structure for employee's records and input following information, then display it –

Name	Age	ID	Salary	City

Naresh	22	458	10000	SGNR
Ravi	21	452	15000	Delhi
Arjun	23	428	15500	Mumbai

```
#include<stdio.h>
#include<conio.h>
struct emp
{
    char n[50],city[50];
    int age , id;
    long int sal;
}S[3];
void main( )
{
    int r;
    clrscr();
    for(r=0;r<=2;r++)
    {
        fflush(stdin);
        printf("Enter Name of Employee: ");
        gets(S[r].n);
        printf("Enter age of Employee: ");
        scanf("%d",&S[r].age);
        printf("Enter ID of Employee: ");
        scanf("%d",&S[r].id);
        printf("Enter Salary of Employee: ");
        scanf("%ld",&S[r].sal);
        fflush(stdin);
        printf("Enter City of Employee: ");
        gets(S[r].city);
    }
    printf("Name\tAge\tID\tSalary\tCity");
    printf("-----");
    for(r=0;r<=2;r++)
    {
        printf("\n%s\t", S[r].n);
        printf("%d\t", S[r].age);
        printf("%d\t", S[r].id);
        printf("%ld\t", S[r].sal);
        printf("%s\t", S[r].city);
    }
    getch();
}
```

OUTPUT:

```
Enter Name of Employee: Naresh
Enter age of Employee: 22
Enter ID of Employee: 458
Enter Salary of Employee: 10000
Enter City of Employee: SGNR
Enter Name of Employee: Ravi
Enter age of Employee: 21
Enter ID of Employee: 452
Enter Salary of Employee: 15000
Enter City of Employee: Delhi
Enter Name of Employee: Arjun
Enter age of Employee: 223
Enter ID of Employee: 428
Enter Salary of Employee: 15500
Enter City of Employee: Mumbai
```

Name	Age	ID	Salary	City
Naresh	22	458	10000	SGNR
Ravi	21	452	15000	Delhi
Arjun	23	428	15500	Mumbai

Que 123: WAP to define structure named staff and input following information 'n' times and display it– Name of Post, Shift, ID.

```
#include<stdio.h>
#include<conio.h>
struct staff
{
    char pn[50],sh[10];
    long int sal;
};
void main( )
{
    struct staff S[100];
    int r,n;
    clrscr();
    printf("How much records you want to store: ");
    scanf("%d",&n);
    for(r=0;r<=n;r++)
    {
        fflush(stdin);
        printf("Enter Name of Post: ");
        gets(S[r].pn);
        printf("Enter Shift (morning/Evening): ");
        gets(S[r].sh);
        printf("Enter Salary: ");
        scanf("%d",&S[r].sal);
    }
    printf("\nPost\tShift\t\tSalary");
    printf("-----");
```

```

    for(r=0;r<=n;r++)
    {
        printf("\n%s\t", S[r].pn);
        printf("%s\t", S[r].sh);
        printf("%d\t", S[r].sal);
    }
    getch();
}

```

OUTPUT:

```

How much records you want to store : 3
Enter Name of Post : Manager
Enter Shift(Morning/Evening) : morning
Enter salary : 50000
Enter Name of Post : Clerk
Enter Shift(Morning/Evening) : morning
Enter salary : 25000
Enter Name of Post : Cashier
Enter Shift(Morning/Evening) : morning
Enter salary : 28000

```

Post	Shift	Salary
Manager	morning	50000
Clerk	morning	25000
Cashier	morning	28000

Structure as Function Argument

We can pass a structure as a function argument in very similar way as we pass any other variable or pointer. We would access structure variables in the similar way as we have accessed in the above example:

Que 124: WAP a program to demonstrate passing structure as a function argument.

```

#include<stdio.h>
#include<conio.h>
struct staff
{
    char pn[50],sh[10];
    long int sal;
};
void show(struct staff st)
{
    printf("\nPost\tShift\tSalary");
    printf("-----");
    printf("\n%s\t", S.pn);
    printf("%s\t", S.sh);
    printf("%d\t", S.sal);
}
void main()
{
    struct staff S;
    clrscr();
}

```

```

        fflush(stdin);
        printf("Enter Name of Post: ");
        gets(S.pn);
        printf("Enter Shift (morning/Evening): ");
        gets(S.sh);
        printf("Enter Salary: ");
        scanf("%d",&S.sal);
        show(S);
        getch();
    }

```

OUTPUT:

```

Enter Name of Post: Manager
Enter Shift (Morning/Evening) : morning
Enter salary : 50000

```

Post	Shift	Salary

Manager	morning	50000

Que125: WAP to create a structure named staff and input n' times records and display using a function named show().

```

#include<stdio.h>
#include<conio.h>
struct staff
{
    char pn[50],sh[10];
    long int sal;
};
void show(struct staff st[], int n)
{
    printf("\nPost\tShift\t\tSalary");
    printf("-----");
    for(r=0;r<=n-1;r++)
    {
        printf("\n%s\t",S[r].pn);
        printf("%s\t",S[r].sh);
        printf("%d\t",S[r].sal);
    }
}
void main()
{
    struct staff S[100];
    int r,n;
    clrscr();
    printf("How much records you want to store: ");
    scanf("%d",&n);
    for(r=0;r<=n-1;r++)
    {
        fflush(stdin);
        printf("Enter Name of Post: ");
        gets(S[r].pn);
    }
}

```

```

        printf("Enter Shift (morning/Evening): ");
        gets(S[r].sh);
        printf("Enter Salary: ");
        scanf("%d",&S[r].sal);
    }
    show(S,n);
    getch();
}

```

OUTPUT:

```

How much records you want to store : 2
Enter Name of Post : Manager
Enter Shift(Morning/Evening) : morning
Enter salary : 50000
Enter Name of Post : Clerk
Enter Shift(Morning/Evening) : morning
Enter salary : 25000
Post      Shift      Salary
-----
Manager   morning   50000
Clerk     morning   25000

```

Pointer to Structure

We can define pointers to structures in very similar way as you define pointer to any other variable as follows:

```
struct emp *E;
```

Now, we can store the address of a structure variable in the above defined pointer variable. To find the address of a structure variable, place the `&` operator before the structure's name as follows:

```
E = &emp1;
```

To access the members of a structure using a pointer to that structure, we must use `->` operator as follows:

```
E -> name;
```

Que 126: WAP to create a structure having pointer type variable.

```

#include<stdio.h>
#include<conio.h>
struct stud
{
    int id;
    char n[20];
};
void main()
{
    struct stud *st;                //pointer variable must be declared inside
main()
    clrscr();
    printf("Enter ID of student: ");
    scanf("%d",&st->id);
    printf("Enter Name of student: ");
    gets(st->n);
}

```

```

printf("ID\tName\n-----");
printf("%d\t%s\n",st->id,st->n);
getch();
}

```

OUTPUT:

```

Enter ID of student: 123
Enter Name of student: Naresh
ID      Name
-----
123     Naresh

```

Que 127: WAP to define a structure and pass structure variable address as argument to a function named show() that will display the information about staff member.

```

#include<stdio.h>
#include<conio.h>
struct staff
{
    char pn[50],sh[10];
    long int sal;
};
void show(struct staff *st)
{
    printf("\nPost\tShift\t\tSalary");
    printf("-----");
    printf("\n%s\t", st->pn);
    printf("%s\t", st->sh);
    printf("%d", st->sal);
}
void main()
{
    struct staff S;
    clrscr();
    fflush(stdin);
    printf("Enter Name of Post: ");
    gets(S.pn);
    printf("Enter Shift (morning/Evening): ");
    gets(S.sh);
    fflush(stdin);
    printf("Enter Salary: ");
    scanf("%d",&S.sal);
    show(&S);
    getch();
}

```

OUTPUT:

```
Enter Name of Post: Manager
Enter Shift (Morning/Evening): morning
Enter salary: 50000
```

Post	Shift	Salary

Manager	morning	50000

Union

A union is a special data type available in C that enables us to store different data types in the same memory location. We can define a union with many and different members, but only one member can contain a value at any given time. Unions provide an efficient way of using the same memory location for multi-purpose.

Defining a Union

Define a union is very similar to defining a structure. The union statement defines a new data type, with more than one member for your program. The format of the union statement is following –

Syntax:

```
union [union tag]
{
    member definition;
    ....
    member definition;
} [one or more union variables];
```

The union tag is **optional**. In union every data member has normal variable definition as we earlier define in main function. At the end of the union's definition, before the final semicolon, we can specify one or more union variables but it is optional.

For Example

Que 128: WAP to define a Union having three different data members and display the total size of union.

```
#include<conio.h>
#include<stdio.h>
union DATA
{
    int n;
    float x;
    char c[20];
}D;
void main()
{
    clrscr();
    printf("Total size of Union is :%d Bytes",sizeof(D));
    getch();
}
```

OUTPUT:



Total size of Union is: 20 Bytes

In above example we can see total size of union is 20 Bytes, It is just because of the maximum size of data member of c[20]. Every character takes 1 Bytes in memory and we define 20 element arrays of character so the size of union is displays 20 Bytes.

CHECKPOINT: Every union occupies a size of its largest variable defined inside it.

Accessing Union Members

To access any member of a union, we use the member access operator (.) same as we used in structure. The member access operator is coded as a period between the union variable name and the union member that we wish to access. We use **union** keyword to define its variable (**Object**).

To access any member of union we must have to define its variable, after it we can able to access the member of union. Suppose we have a union named DATA and we define its variable as D, so we can access its member defined as integer as following –

```
D.n=10;
```

For Example:

Que 129: WAP to define a union and access its data member.

```
#include<conio.h>
#include<stdio.h>
union DATA
{
    int n;
    float x;
    char c[20];
}D;
void main()
{
    clrscr();
    printf("Enter a value: ");
    scanf("%d",&D.n);
    printf("Enter a floating value: ");
    scanf("%f",&D.x);
    printf("Enter a string: ");
    scanf("%s",D.c);
    printf("Integer value: %d",D.n);
    printf("Floating value: %f",D.x);
    printf("String is: %s",D.c);
    getch();
}
```

OUTPUT:

```
Total size of Union is : 20
Enter a value : 20
Enter a floating value : 12.25
Enter a String : naresh
Integer value : 24942
Floating value :2254784587542200000.000000
string is : naresh
```

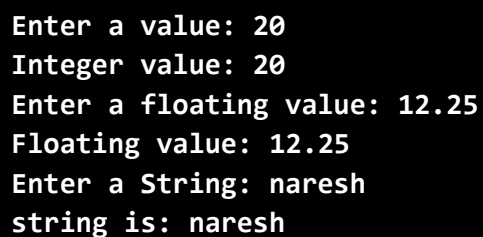
In above example we got a corrupted result in the output. This is just because of the final variable occupied the memory location. This is also causing the correct output for last variable (**str**).

So now it's clear that union locates memory for largest size variable and this causes of corrupting the output. to prevent these unwanted changes, we must separate every input and output statement. Let see following example to see the difference –

Que 130: WAP to define a union and display data as it inputted by user.

```
#include<conio.h>
#include<stdio.h>
union DATA
{
    int n;
    float x;
    char c[20];
}D;
void main()
{
    clrscr();
    printf("Enter a value: ");
    scanf("%d",&D.n);
    printf("Integer value: %d",D.n);
    printf("Enter a floating value: ");
    scanf("%f",&D.x);
    printf("Floating value: %f",D.x);
    printf("Enter a string: ");
    scanf("%s",D.c);
    printf("String is: %s",D.c);
    getch();
}
```

OUTPUT:

A screenshot of a terminal window with a black background and white text. It shows the output of the C program: 'Enter a value: 20', 'Integer value: 20', 'Enter a floating value: 12.25', 'Floating value: 12.25', 'Enter a String: naresh', and 'string is: naresh'. Each input is followed by its corresponding output on the same line.

```
Enter a value: 20
Integer value: 20
Enter a floating value: 12.25
Floating value: 12.25
Enter a String: naresh
string is: naresh
```

Now we got correct output as we input the information. We separate every input statement with its output statement. This causes correct output.

File Handling

In programming, we may require some specific input data to be generated several numbers of times. Sometimes, it is not enough to only display the data on the console. The data to be displayed may be very large, and only a limited amount of data can be displayed on the console, and since the memory is volatile, it is impossible to recover the programmatically generated data again and again. However, if we need to do so, we may store it onto the local file system which is volatile and can be accessed every time. Here, comes the need of file handling in C.

A file represents a sequence of bytes, regardless of it being a text file or a binary file. C programming language provides access on high level functions as well as low level (OS level) calls to handle file on your storage devices.

File handling in C enables us to create, update, read, and delete the files stored on the local file system through our C program. The following operations can be performed on a file.

- Naming or Creating a file.
- Opening a file.
- Write data to file.
- Read data from file.
- Closing a file.

To perform above operations there are numbers of various function available in C language. These are –

- 1) fopen()
- 2) fclose()
- 3) fcloseall()
- 4) fputc()
- 5) fgetc()
- 6) putw()
- 7) getw()
- 8) fprintf()
- 9) fscanf()
- 10) fwrite()
- 11) fread()
- 12) ftell()
- 13) rewind()
- 14) fseek()

Opening a File

We must open a file before it can be read, write, or update. All the task can be performed after opening a file.

fopen()

We can use the **fopen()** function to create a new file or to open an existing file. This call will initialize an object of the type **FILE**, which contains all the information necessary to control the stream. The prototype of this function call is as follows –

Syntax:

```
FILE *filepointer;  
filepointer=fopen("filename_with_extension","opening mode");
```

fopen() function create or open a file and store its address to the file pointer. If file doesn't exist than it stores NULL in the file pointer.

The **fopen()** function works in the following way.

- Firstly, it searches the file to be opened.
- Then, it loads the file from the disk and place it into the buffer. The buffer is used to provide efficiency for the read operations.
- It sets up a character pointer which points to the first character of the file.

Opening Modes

We can use one of the following modes in the **fopen()** function.

Mode	Description
r	opens a text file in read mode
w	opens a text file in write mode
a	opens a text file in append mode
r+	opens a text file in read and write mode
w+	opens a text file in read and write mode
a+	opens a text file in read and write mode
rb	opens a binary file in read mode
wb	opens a binary file in write mode
ab	opens a binary file in append mode
rb+	opens a binary file in read and write mode
wb+	opens a binary file in read and write mode
ab+	opens a binary file in read and write mode

Table: Opening Modes in File Handling

Closing a File

After we have done working with opened file then we must close the file.

fclose()

The **fclose()** function is used to close a file. The file must be closed after performing all the operations on it. The syntax of **fclose()** function is given below:

Syntax:

```
fclose(filepointer);
```

Writing a File

Putting some character into a file is termed as writing a file. To write a file we have following function available in C Language:

- 1) **fprintf()**
- 2) **fputc()**
- 3) **fputs()**

1) fprintf()

This Function is used to write a set of characters or a string into a file. It sends formatted output to a stream.

Que 131: WAP to input a program to write a file using *fprintf()* on HDD.

```
#include<conio.h>
#include<stdio.h>
void main()
{
    FILE *fp;
    clrscr();
    fp=fopen("demo.txt","w");           // opening file
    fprintf(fp,"It is written with fprintf"); // writing data into file
    fclose(fp);                         // closing file
    printf("File is successfully written!!");
    getch();
}
```

OUTPUT:

```
File is successfully written!!
```

In above example a file named demo will be created if it doesn't exist otherwise the file will be open in **write mode**. Then two strings will be stored in this text file that we are writing using two different functions.

2) fputc()

This Function is used to write a single character into a file. It outputs a character to a stream.

Que 132: WAP to input a program to write a file using *fputc()* on HDD.

```
#include<conio.h>
#include<stdio.h>
void main()
{
    FILE *fp;
    clrscr();
    fp=fopen("demo.txt","w");           // opening file
    fputc("H",fp);                     // writing data into file
    fclose(fp);                         // Closing file
    printf("File is successfully written!!");
    getch();
}
```

OUTPUT:

```
File is successfully written!!
```

3) fputs()

The **fputs()** function writes a line of characters into file. It outputs string to a stream.

Que 133: WAP to input a program to write a file using fputs() on HDD.

```
#include<conio.h>
#include<stdio.h>
void main()
{
    FILE *fp;
    clrscr();
    fp=fopen("demo.txt","w");           // opening file
    fputs("\nIt is written with fputs",fp); // writing data into file
    fclose(fp);                         // closing a file
    printf("File is successfully written!!");
    getch();
}
```

OUTPUT:

File is successfully written!!

Reading a File

Accessing a file from HDD is termed as reading file. To read a file from storage we can use following functions:

- 1) fscanf()
- 2) fgetc()
- 3) fgets()

1) fscanf()

The **fscanf()** function is used to read set of characters from file. It reads a word from the file and returns EOF at the end of file.

Que 134: WAP to input a program to read a file using fscanf() from HDD.

```
#include<conio.h>
#include<stdio.h>
void main()
{
    FILE *fp;
    char c[20];
    clrscr();
    fp=fopen("demo.txt","r");           // opening file
    fscanf(fp,"%s",c);                  // reading file from disk
    printf("%s",c);                     // printing data of file on console
    fclose(fp);
    getch();
}
```

OUTPUT: In above example the text stored in demo.txt file will be shown as result.

Hello this is file is written with fprintf

2) fgetc()

The **fgetc()** function returns a single character from the file. It gets a character from the stream. It returns EOF at the end of file.

Que 135: WAP to input a program to read a file using *fgetc()* from HDD.

```
#include<conio.h>
#include<stdio.h>
void main()
{
    FILE *fp;
    char c[20];
    clrscr();
    fp=fopen("demo.txt","r");           // opening file
    while((c=fgetc(fp))!=EOF)           // reading file from disk
    {
        printf("%c",c);                 // printing data of file on console
    }
    fclose(fp);
    getch();
}
```

OUTPUT: In above example the text stored in *demo.txt* file will be shown as result.

```
Hello this is file is written with fprintf
```

3) fgets()

The **fgets()** function reads a line of characters from file. It gets string from a stream.

Que 136: WAP to read a file using *fgetc()* from HDD.

```
#include<conio.h>
#include<stdio.h>
void main()
{
    FILE *fp;
    char c[200];
    clrscr();
    fp=fopen("demo.txt","r");           // opening file
    printf("%s",fgets(c,200,fp));       // reading and printing data.
    fclose(fp);
    getch();
}
```

OUTPUT: In above example the text stored in *demo.txt* file will be shown as result.

```
Hello
```


Checking weather, a File exists or not

To check a file is already on disk or not while we are opening a file in writing mode, we can put some logic for it to give error if file doesn't exist on the disk.

Que 137: WAP to check and open a file and print error message if file doesn't exist.

```
#include<conio.h>
#include<stdio.h>
void main()
{
    FILE *fp;
    char c[20];
    clrscr();
    fp=fopen("demo.txt","r");           // opening file
    if((fp==NULL)
    {
        printf("File not found!!");
        getch();
        exit();
    }
    while((c=fgetc(fp))!=EOF)           // reading file from disk
    {
        printf("%c",c);                 // printing data of file on console
    }
    fclose(fp);
    getch();
}
```

OUTPUT: if file is not existing on the disk.

File not found!!

Operator Precedence & Associativity

Operator	Description	Associativity
() [] . -> ++ --	Parentheses or function call Brackets or array subscript Dot or Member selection operator Arrow operator Postfix increment/decrement	left to right
++ -- + - ! ~ (type) * & sizeof	Prefix increment/decrement Unary plus and minus not operator and bitwise complement type cast Indirection or dereference operator Address of operator Determine size in bytes	right to left
* / %	Multiplication, division and modulus	left to right
+ -	Addition and subtraction	left to right
<< >>	Bitwise left shift and right shift	left to right
< <= > >=	relational less than/less than equal to relational greater than/greater than or equal to	left to right
== !=	Relational equal to and not equal to	left to right
&	Bitwise AND	left to right
^	Bitwise exclusive OR	left to right
	Bitwise inclusive OR	left to right
&&	Logical AND	left to right
	Logical OR	left to right
? :	Ternary operator	right to left
= += -= *= /= %= &= ^= = <<= >>=	Assignment operator Addition/subtraction assignment Multiplication/division assignment Modulus and bitwise assignment Bitwise exclusive/inclusive OR assignment	right to left
,	Comma operator	left to right

Operator Precedence and Associativity