C- Programming

* Problem Solving Using Computer

Computer is an electronic device that takes inputs, processes it and provides the output.

Software is a set of instructions that are programmed by using different programming language.

There are different steps or activities involved while solving problems using computer. They are:

1) Problem Analysis (Identification)
2) Algorithm Development (Procedure)
3) Flowcharting (Flow of Procedure)
4) Program coding (Deployment of steps)
5) Compilation and Execution (Implementation)
6) Testing and Debugging (Review and checking)
7) Documentation. (Result in Paper)

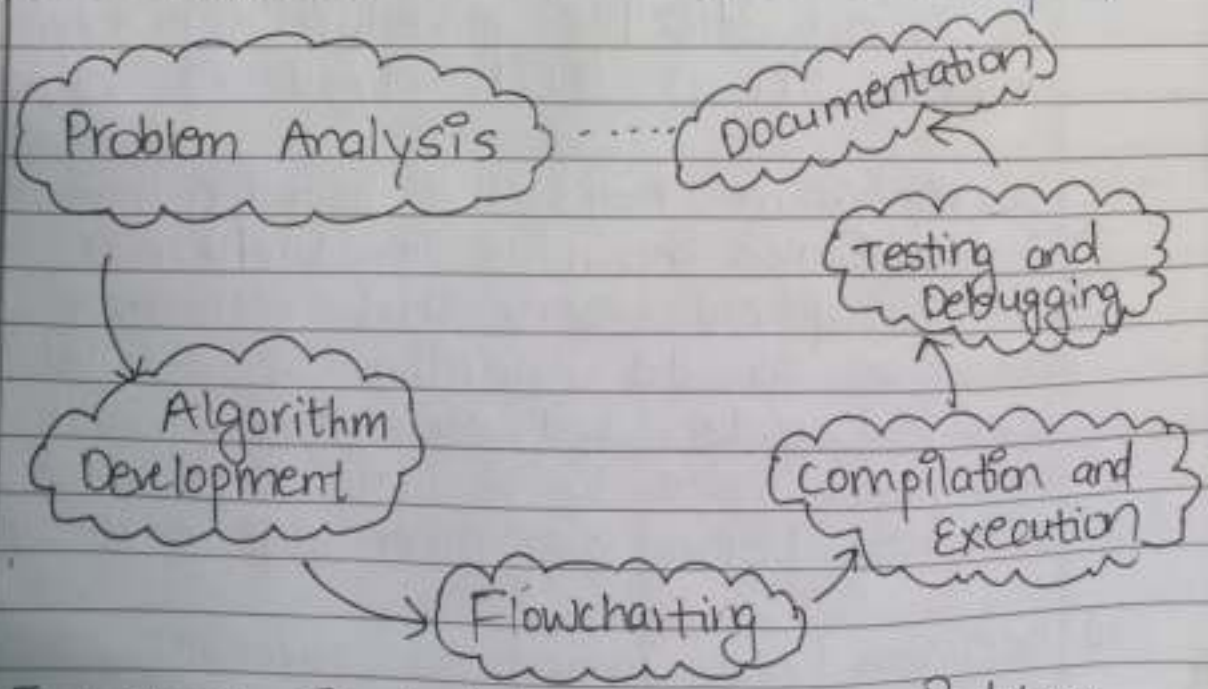


Fig: Steps Involved in Solving a Problem

Date: _____
Page: _____

1) Problem Analysis

Problem analysis is the first step of problem solving using computer. Before solving a problem it should be analyzed and the things like program's input, output, requirement, time period, different ways of solving a problem, etc should be understood.

Problem Analysis is involves decomposing whole parts of a system into smaller parts or modules. Then we have to identify possible inputs, process and output of the problem.

2) Algorithm Development

An algorithm is the step-by-step description of the procedure written in human language for solving given problem. It is an effective procedure for solving a problem in a finite number of steps.

Algorithm shows the following three features

a) Sequence (also known as Procedure/Process)

Sequence means that each steps or processes in the algorithm is executed in the specified order and each process must be in the proper place i.e. previous steps must be executed before any other ~~step~~ next steps.

b) Decision (also known as selection):

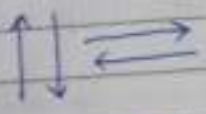
In some case, we have to make decis

If the outcome of the decision is true, then something is done otherwise another thing should happen. If it rains outside, then open umbrella is an example.

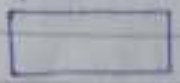
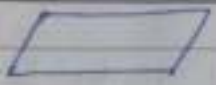
c) Repetition (also known as Iteration or Loop)
If the algorithm repeats again and again until some condition occur then it is called repetition. For example, fill water in tank until tank is full or overflow.

3) Flowcharting (Flowchart Development):
Flowchart is the graphical representation of an algorithm by using standard symbols. Flowchart is a pictorial representation of an algorithm that uses ~~boxes of~~ different shapes, connector, arrows, etc to show structure of a program flow.

Symbols used In Flowchart.

Symbol	Name	Description.
	Arrow	used to connect flow-chart symbol and direct the flow.
	Start/End (Rounded Rectangle)	used to represent start and end of the process.

Date :
Page :

Symbol	Name	Use
	Rectangle	used for arithmetic and data manipulation.
	Input/Output	Used for input and output.
	Diamond (Decision)	used for decision making and branching
	Circle (Connector)	used to connect different flow lines and remote parts of flowchart.
	Function call	used when function call from main function.
	For loop	used to indicate for loop.

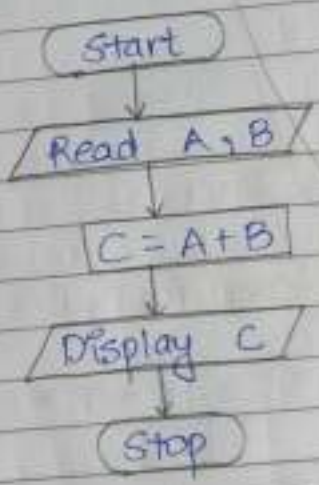
Ans.
Step 1:
Step 2:
Step 3:
Step 4:
Step 5:
Step 6:

Write an algorithm and flowchart for finding the sum of two numbers.

Ans Algorithm

Step 1: Start
Step 2: Display "Enter two numbers"
Step 3: Read A and B
Step 4: Store sum as C i.e. C=A+B
Step 5: Display "C as sum of two numbers"
Step 6: Stop

Flowchart



Write an algorithm and draw flowchart to calculate simple interest.

Ans Algorithm

Step 1: Start
Step 2: Accept principal as P
Step 3: Accept time as T
Step 4: Accept rate as R
Step 5: Multiply the value of P, T and R and ~~store it as~~ divide it by 100 and store the result as SI
Step 6: Display the value of SI
Step 7: Stop

Flowchart

Write an algorithm and draw a flowchart to determine wheather a number is positive or negative

Ans Algorithm

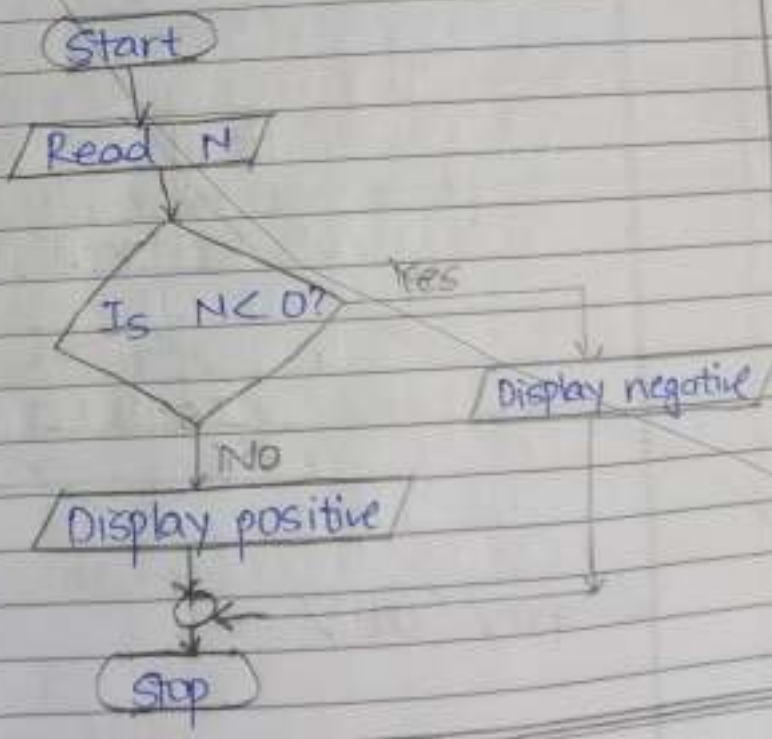
Step 1: Start

Step 2: Accept number as N

Step 3: If the value of N is less than zero display "~~pos~~ negative" otherwise display "positive"

Step 4: Stop

Flowchart



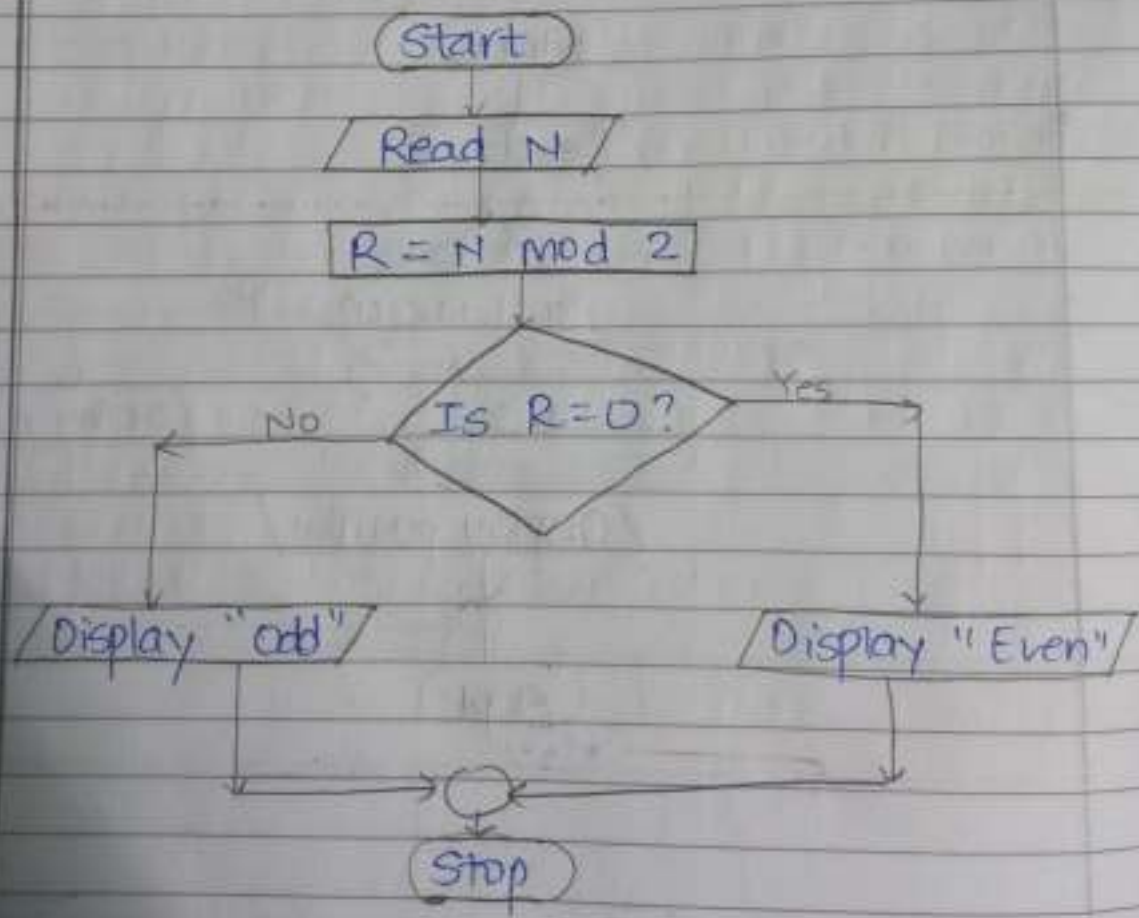
Date:
Page:

Write an algorithm and draw a flowchart to check wheather a number is odd or even.

Ans Algorithm

Step 1: start
Step 2: Accept number as N
Step 3: Divide N by 2 and store remainder as R
Step 4: If ~~the ve~~ R=0, then display "even" otherwise display "odd"
Step 5: Stop

Flowchart



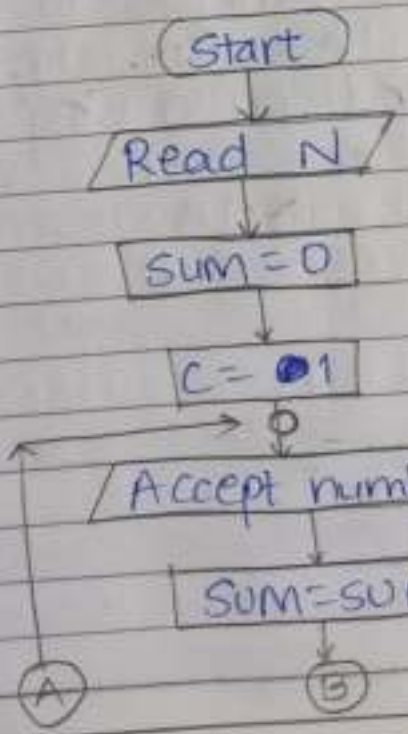
Ans
Step 1:
Step 2:
Step 3:
Step 4:
Step 5:
Step 6:
Step 7:
Step 8:
Step 9:

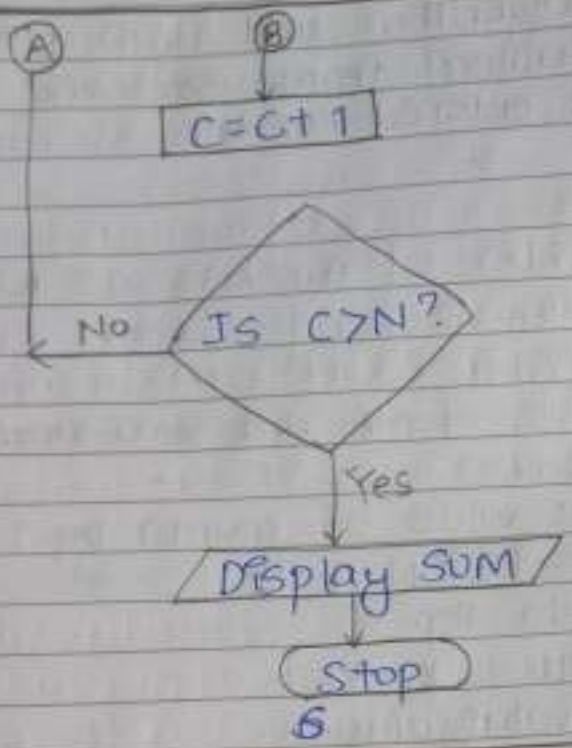
Write an algorithm and draw flowchart to read N numbers from user and display sum of all entered numbers.

Ans Algorithm

Step 1: Start
Step 2: Accept number of numbers as N
Step 3: Initialize sum=0 and counter=1
Step 4: Read a number as NA
Step 5: Add the value of NA and sum and store the value as sum
Step 6: Increase the value of counter by 1
Step 7: Is counter = N+1 ?
  Yes: Go to step 8
  NO: Go to step 4
Step 8: Display the value of sum
Step 9: Stop

Flowchart




A
B
C=C+1
No
IS C>N?
Yes
Display SUM
Stop

4. Program coding:

Coding is the process of transferring paper works like algorithm and flowchart into computer program by following well defined syntax using programming languages like C, C++, etc. The coding is the process of transforming the program logic design into a computer language format.

5. Compilation and Execution:

The process of changing high level language into low level language at once is called compilation done by a special software known as compiler. compilation process tests whether the program has syntax error or not. If there is syntax error, compiler cannot compile the code. once compilation is completed then the program is linked with ~~the~~ object, the other objects program needs for execution by resulting in a binary program which is loaded in the memory for execution. During execution the program may ask for input and generate output after processing input.

Here, is the sequence.

Program Code → Compilation (↑ Compiler) → Linking (↑ Linker) → Loading (↑ Loader) → Execution

Date:
Page:

6. Debugging and Testing:

Debugging is the process of finding errors and removing them from a program. Some error may remain in the program which appear during compilation/linking/execution of a program. When error occur, debugging is required. Testing is performed to verify ~~that~~ whether the program functions or works according to the expectation defined by the requirement.

7. Documentation:

Program documentation is the description of program and its logic written to support for understanding the program. Program documentation is the systematic writing of activities from problem analysis to debugging and testing phase including operational information and presenting it as a report.

Documentation can be

- Technical documentation or programmer documentation
- User documentation or user manual.

Date:
Page:

Basic structure of C-Programming

1	Documentation Section
2	Link section
3	Definition section
4	Global Declaration

5	main() function section
	{
	Declaration section
	Execution section
	}

6	Sub-program section Function 1(){..} Function 2(){} : Function n(){..} — user defined function.

1. Documentation section

This section is the part of program where the programmer gives the detail associated with the program. This section contains a set of comment lines giving name of program, the author, algorithm used or any other information that may require in the program.

Example:

```
/* File name :- Hello.c
Author name :- Manu Sharan Sah
Date :- 2080/09/12
Description: A program to display hello world*/
```

2. Link section (Preprocessor section)
This part of the code is used to ~~describe~~ declare all the header file. This section provide instruction to the compiler to link function with program from system libraries.
Example:
```
#include <stdio.h>
#include <conio.h>
```
Here, # include, includes the specific file as a part of function at the time of compilation
#include <stdio.h> → consist of the content of standard input/output file like printf and scanf.

#include <conio.h> is used to include the console ~~o~~ input/output library function like getch() function.

Header file is file having .h extension inclusion of the header file in the program is the pre-information to the compiler regarding the library function used and its relevant code to be executed. This type of pre-information is called compiler directive or preprocessor directive.

3. Definition section
In this section, we define different constant keyword.
```
#define PI 3.14
```

4. Global Declaration section
Global Declaration section contains global variable, function declaration and static variable.
For example: float num = 2.54

5. main() function section
Every C program needs main() ~~program~~ function which contain 2 parts i.e. declaration part and execution part. Declaration part is the part where all variables are declared that are used in execution part where execution of the program occurs. main() function starts with '{' and ends with '}'.
All the tasks are done within main() function. If there is any user-defined function or we need to call some other function then we call ~~in~~ from main() function.

6. Sub-program section (user-defined section)
This section contains all the user-defined function that are called from main function.

C Welcome Program

```
/* This is a program that welcomes in
   the C-world */
#include <stdio.h>
#include <conio.h>
void main(){
printf("Hello world");
printf("Welcome to C-programming");
getch();
}
```

WAP to add two numbers from user's input.

```
/* This program adds two numbers */
#include <stdio.h>
#include <conio.h>
void main()
{
  clrscr();
  printf("Enter two numbers:");
  int a,b;
  scanf("%d%d",&a,&b);
  int c=a+b;
  printf("The sum of %d and %d is %d",a,b,c);
  getch();
}
```

By Assigning

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

```
int a=5,b=6;
int c=a+b;
printf("The sum is %d",c);
getch();
}
```

WAP to calculate simple interest.

```
/* This program calculates simple interest*/
#include <stdio.h>
#include <conio.h>
void main()
{
  clrscr();
int p,t,r,si;
printf("Enter the principal");
scanf("%d",&p);
printf("Enter the time in years");
scanf("%d",&t);
printf("Enter the rate per annum");
scanf("%d",&r);
si=(p*t*r)/100;
printf("The simple interest is %d",si);
getch();
}
```

Date:
Page:

Character set

C has its own set of vocabulary and grammar.

The set of characters that are used to form words, numbers and expressions in C is called character set. The character set are grouped into four categories.

1. Letter or Alphabet:

   All english letter and alphabets are supported by C i.e. uppercase (A to z) and lowercase (a to z)

2. Digit:

   All digit that are used in numeric value or expression.

   0.....9

3. Special character

, → comma	; → semicolon
. → period	& → ampersand
^ → caret	* → asterik
: → colon	? → question mark
' → apostrope	" → quotation mark
! → exclamation mark	| → vertical bar
/ → forward slash	\ → backslash
~ → tilde	_ → underscore
$ → dollar	% → percent
- → minus	+ → plus
< → less than	> → greater than
(→ left paranthesis	) → right paranthesis
[→ left bracket	] → right bracked
# → number or hash	Etc.

4. White space
   1. blank space \b
   2. Horizontal tab \t
   3. vertical tab \v
   4. carriage return \r
   5. newline \n
   6. Form feed \f

Keywords.

Keywords are predefined words for C-programming language which is reserved word. There are 32 keywords in C. They are:

auto	default	for	return
break	do	goto	short
case	double	if	signed
char	else	int	sizeof
const	extern	long	static
continue	float	register	struct

switch, typeof, union, unsigned, void, volatile, while

Identifiers.

An identifier is a type of string of alphanumeric character that always begins with either an alphabet or underscore character.

Rules to define identifier

- We cannot use the already defined keyword present in C language.
- All identifier should have a unique name in same scope.
- Identifier cannot start with a digit but

Date:
Page:

should start with alphabet and combaine with digit.
- Special characters like '*', '#', '@', '$' are not allowed.
- It is case sensitive i.e. A and a is different.
- Length of an identifier should not exceed 31 character
- Any blank space or commas are not allowed within an identifier.

Valid Example	Invalid
add, Add, abc, long_integer_to_store_sum -abc, etc.	m n, m+n, int, 5size, etc.

Datatype

Data type are the types of data stored in a C-program. It is important for the compiler to understand the type of predefined data. Data type is an attribute that tells a computer how to interpret the value.

Types of data in C

Primary	Derived	user-defined
- Integer	- Function	- Class
- character	- Array	- Structure
- Floating point	- Pointer	- Union
- Double floating point	- Reference	- Enum
- Void		- Typedef

Primary data type / Fundamental data type / Basic Data Type

1) Integer: Data type that are used for whole number without decimal value.
   : 16 bit of storage
   : three class of integer
     - integer (int)
     - short integer (short int)
     - long integer (long int)

2) Floating point: used for fractional number i.e. numbers with decimal values.
   : keyword used is float
   : Reserved 32 bit (4 byte) with 6 digit of precision.

For example:
3.1424578 floating point contains only six digits i.e. 3.14245. The number will be expressed as $3.14245e^5$
(3.14245 → mantissa)

3) Double precision floating point:
   : used to store decimal numbers with double precision.
   : use 64 bits (8 bytes) giving a precision of 14 digits.
   : to extend the precision we use long double that use 80 bits (10 bytes) giving 18 digits of precision

4) Character: Store single 'character with size of 8 bits (1 byte) with keyword 'char'.

5) Void: used to specify that no value is present.

Derived Data Type

They are derived basically from derived from fundamental data types. Derived data type won't typically create a new data type but would add various new functions to the existing one.

Types of derived data type

1) Array: Basically refers to a sequence of finite number of data item from the same data type sharing one common name.

2) Function: Refers to self contained block of statement(s) which has its own specified name.

3) Pointer: Refers to some special form of variable that can be used for holding other variable address.

user-defined data type

Those data type which are defined by the user as per his/her will are called user-defined data type. For example: structure, union, class, enum and typedef.

Constant

→ Constant is a name memory location which holds one value throughout the program execution.

Creating constant

Two ways

1) Using 'const' keyword
2) using '#define' keyword.

1) Using 'const' keyword

→ To create a constant, we prefix the variable declaration with 'const' keyword.

Syntax: datatype

const ~~datatype~~ [constant name] = [value]

For example:

```
const int x
    or
const int x=10
```

2) using '#define' preprocessor

We can create constant using '#define' preprocessor directive.

Syntax:

#define [constantname] [value]

For example:

```
#define pi 3.14
```

Date:
Page

1.

Variable

A variable in C is a memory location with some name that helps to store some form of data and retrieves it when required.

Syntax

[data-type] [variable-name] = [value] (Assigning)
[data-type] variable1, variable2, ... (Declaring)
For eg; int a=5; int a,b,c;

Operator

An operator is a symbol that operates on single or multiple data item.

→ Operand: Data item that operator acts upon

→ Expression: Combination of variable, constant and operator written according to syntax of the language.

For example:
y=a+b

Types of operator

On the basis of utility or function of operator, they are classified as:

1. Arithmetic operator
2. Assignment operator
3. Relational operator
4. Logical operator
5. Bitwise operator.

1. Arithmetic operator

C provides all the basic arithmetic operation.

Operator	Meaning	Example
+	Addition	a+b
-	Subtraction	a-b
*	Multiplication	a*b
/	Division	a/b
%.	Modular Division	a%.b

Write a program that perform all the arithmetic operation

```
/* This program provides performs all the
 arithmetic operation*/
#include <stdio.h>
#include <conio.h>
void main()
{
  int a,b,sum,diff,prod,quo,rem;
  Sum= printf("Enter first number");
       scanf("%d",&a);
       printf("Enter second number");
       scanf("%d",&b);
  sum=a+b;
  diff=a-b;
  prod=a*b;
  quo=a/b;
  rem=a%.b;
printf("The sum is %d",sum);
printf("The difference is %d",diff);
printf("The product is %d",prod);
```

```
printf("The quotient is %d", quo);
printf("The remainder is %d", rem);
getch();
}
```

Points to remember

int + int = int
int + float = int
float + float = float
int * float = int
float / int = float
int / float = float

2. Assignment operator

Assignment operator are binary operators that are used to assign result of an expression to a ~~volatile~~ variable. Mostly used assignment operator is '='.

operator	Meaning	Example:
+=	Add and assign	a+=b ⇒ a=a+b
-=	Subtract and assign	a-=b ⇒ a=a-b
*=	Multiply and assign	a*=b ⇒ a=a*b
/=	Divide and assign	a/=b ⇒ a=a/b
%=	Remainder and assign	a%=b ⇒ a=a%b

3. Relational operator

Relational operators are binary operators that are used to compare two values or condition

Operator	Meaning	Example:
<	less than	a<b
>	greater than	a>b
<=	less than or equal to	a<=b
>=	greater than or equal to	a>=b
==	equal to	a==b
!=	not equal to	a!=b

↳ Equality operator

4. Logical operator

Logical operator are binary operator that are used to compare or relate logical and relational operation.

Logical operator are

&& ⇒ Logical AND

|| ⇒ Logical OR

! ⇒ Logical NOT

Truth Table

A	B	A&&B	A||B	!A	!B
T	T	T	T	F	F
T	F	F	T	F	T
F	T	F	T	T	F
F	F	F	F	T	T

In terms of binary numbers:

A	B	A&&B	A||B	!A	!B
1	1	1	1	0	0
1	0	0	1	0	1
0	1	0	1	1	0
0	0	0	0	1	1

5. Bitwise Operator

Bitwise operator are binary operator that are used for manipulating data at bit level.

Types of Bitwise operator.

1) Bitwise logical operator
2) Bitwise shift operator
3) ~~On's~~ One's complement operator.

1) Bitwise logical operator:

It performs logical test between integer type operand. Three types of bitwise logical operator.

a. Bitwise AND operator (&)

Bitwise AND performs operation between two operand and result value is 1 if both bit value is 1 otherwise 0.

b. Bitwise OR operator (|)

Bitwise OR performs operation between two operands and result value 1 if any of the bit value is 1.

c. Bitwise ~~NOT operator~~ Exclusive (XOR) operator (^)

The result of Exclusive ORing operator is 1 ~~only~~ if one of the bit have a value of 1 otherwise 0. If two bits have same value then result is 0.

Bitwise logical operators are often used in low-level programming and manipulation of

binary data.

Find the output of the given program.

```
void main()
{
  int n1=60, n2=15, AND, OR, XOR;
  AND = n1 & n2;
  OR = n1 | n2;
  XOR = n1 ^ n2;
  printf("AND => %d\n", AND);
  printf("OR => %d\n", OR);
  printf("XOR => %d\n", XOR);
  getch();
}
```

Ans AND => 12
OR => 63
XOR => 51

2) Bitwise shift operator

Bitwise shift operator are used to move bit pattern either to the left or right.
It is of two types:

1) Left shift (<<):

The left shift operation cause operand to be shifted to the left by same bit position. General form

operand << n

For example: n1<<3 means 3 times left shift

If $n_1 = 60$ and $n_2 = n_1 << 3$

n_1	0000	0000	0011	1100
1st	0000	0000	0111	1000
2nd	0000	0000	1111	0000
3rd	0000	0001	1110	0000

$\therefore n_2 = 480$

2) Right shift (>>):
The right-shift operation cause the operand to be shifted to right by same bit position. General form:

operand >> n

For example: $n_1 >> 3$

If $n_1 = 60$ and $n_2 = n_1 >> 3$

n_1	0000	0000	0011	1100
1st	0000	0000	0001	1110
2nd	0000	0000	0000	1111
3rd	0000	0000	0000	0111

→ These operations are used often for efficient multiplication or division by powers of 2.

→ These operators are useful in optimizing certain arithmetic operations and manipulating data at the bit level in programming.

Ans

Output of the program

```
void main()
{
    int n1 = 60, left, right;
    left = n1 << 3;
    right = n2 >> 3;
    printf("Left => %d\n", left);
    printf("Right => %d\n", right);
    getch();
}
```

Ans Left => 480
Right => 7

Bitwise one's complement

Bitwise one's complement operator is unary operator which convert all bits represented by its operand.

For example: $n_1 = 60$ and $n_2 = \sim n_1$

n_1	0000	0000	0011	1100
n_2	1111	1111	1100	0011

$n_2 = 65475$

$n_2 \Rightarrow -61$

Date:
Page

On the basis of the number of operand required for an operator, they are classified as:

1) Unary operator
2) Binary operator
3) Ternary operator.

1) Unary operator:

The operator which requires only one operand are known as unary operator. These operators are like -(unary minus), +(unary plus), ++ (increment), -- (decrement)

Increment and decrement operators are used to increase and decrease the value of operand by 1.

1. The increment and decrement operator can be used as postfix (a++) and prefix (++a) notation.

Program

```
void main()
{
  int a=10;
  printf("a=%d\n",a);
  printf("a=%d\n", ++a);
  printf("a=%d\n", a++);
  printf("a=%d\n", a);
  getch();
}
```

Output
→ a = 10
→ a = 11
→ a = 11
→ a = 12
→

The increment and decrement operator in the form of ++a and --a are prefix notation. In prefix notation, the value of variable is first increased or decreased and then the value is used. Similarly, increment and decrement operator in the form a++ and a-- are postfix notation. In postfix, first the previous value of variable is used and then the value of variable is increased or decreased.

2) Binary operator
The operator which require two operands are known as binary operators. for example: + (plus), - (minus), * (multiply), / (division), < (less than), > (greater than), etc.

3) Ternary operator (Also called conditional operator)
The operator that requires three operands ~~are~~ is known as ternary operator.
Syntax:
value = expression1 ? expression2 : expression3.
For example:
larger = $n_1 > n_2$? n_1 : n_2

At first expression1 is evaluated. If expression1 is true then expression 2 will be final value otherwise expression3 will be the final value.

WAP to read two numbers from user and determine the larger number using conditional operator.

```
#include <stdio.h>
#include <conio.h>
void main()
{
    int a, b, great;
    printf("Enter two numbers");
    scanf("%d %d", &a, &b);
    great = a>b ? a : b;
    printf("The larger number is %d", great);
getch();
}
```

Comma operator (,)

A command-linked list of expression are evaluated from left-right and value of rightmost expression is the value of combined expression.

$n_3 = (n_1 = 50, n_2 = 10, n_1 + n_2)$;

Sizeof() operator

Sizeof() operator is used with an operand to return the number of memory

bytes it occupies in primary memory. sizeof() operator may be constant, variable or data type.

Example:

```
void main()
{
    float n1;
    printf("Number of bytes ~~located~~ allocated
            for float variable n1 is %d", sizeof(n1));
    getch();
}
```

Operator Precedence and Associativity.

Precedence: It is an order of evaluation of operator existing in an expression. The operator at higher level of precedence ~~The operator at higher level~~ is evaluated first and then operator of next level is evaluated.

Associability: It is an order of evaluation of operator with same level of precedence in an expression. The operator of same precedence are evaluated either left or right or from right to left depending on the level of called associativity property of an operator.

Precedence Level	Operator	Description	Associativity
1	(), []	Parenthesis array index	Left to right
2	+, -, ++, --, !, ~, *, &, sizeof()		Right to left
3	*, /, %		Left to right
4	+, -		Left to right
5	<<, >>		Left to right
6	<, <=, >, >=		Left to right
7	==, !=		Left to right

```
int main()
{
  int a=10, b;
  b= a++ + ++a;
  printf("%d, %d, %d, %d", b, a++, a, ++a);
  return 0;
}
```

Output

→ 22, 12, 13, 14

Definition of computer Program and Programming Language

Computer program is a set of instruction given to the computer in order to solve a particular problem or perform a particular task. Computer programs are written in a special language known as programming language.

Thus, the language that can be used by the both programmers and computer in order that the computer can perform the task assigned by the programmer is called programming language simply, it is the language to write a computer problem. Java, Python, Javascript, C, etc are widely used programming language.

Types and Generations of Programming languages

On the basis of the type of instruction, programming language are classified into three types as:

1) Low-Level Language

Low-level language, also called machine level language is a computer programming language that makes the use of binary codes i.e 0 and 1) to write a computer program. It can be directly understood by the computer so there is no any requirement of any translator tool or language processor.

Date:
Page:

2) High-level language

High-level language is a computer programming language that makes the use of human-understable language to write a computer program. Depending upon the mode of execution, a high level language needs compiler or interpreter (ie. language processor) to convert the program into machine level language as it is the only language understood by the computers. Python, Java-script, etc are its examples.

3) Middle-level language

Middle-level language is a computer programming language that makes the use of both high-level language and low-level language to write a computer program. They donot use binary codes like LLL and exactly understood words like HLL. Like HLL, it also needs a language processor to be convert to machine language for the processing of the input by the computer. C is the best example of middle-level language.

language intermediate between

Generations of Programming language.

The advancement and development in the programming languages have been simplified in terms of the generations to which they belong. The programming languages can be categorized into five generations as:

1) First-Generation Language (1GL)
1GL is the programming language that uses machine level language to write a computer program. They are executed fastly than any other programming language but the use of binary codes makes them difficult to be used by the programmers.

2) Second-Generation Language (2GL)
2GL is the programming language that uses assembly language (uses human-readable notations that can be further converted into machine language using an assembler) to write a programming language. corrects and location of errors are easy than 1GL but this language is architecture/machine-dependent language

use of mnemonic codes

3) Third-Generation Language (3GL)
3GL is the generation of programming language using high-level language to write a computer program. POP and OOP are the programming paradigms used with this language. C and FORTRAN are few examples of 3GL.

4) Fourth-Generation Language (4GL)
4GL is the generation of programming language that too uses a HLL but with extra capability of users to access the database.

Date:
Page

For example: SQL, Foxpro, Focus, etc

5) Fifth Generation Language (5GL)
5GL is the generation of programming language that is based on the concept of Artificial Intelligence (A.I) and uses the technologies of parallel processing and superconductors for its execution to write a computer program. LISP and PROLOG are ~~few~~ popular examples of 5GL.

Full forms:
LISP: List Processing
PROLOG: Programming in Logic
SQL: Structured query language
LLL: low-level language
HLL: High-level language
MLL: Middle-level language
: Machine-level language

Input/output in C

Input means to provide the program with some data to be used in the program and output means to display data on-screen

There are mainly two types of I/O function in the C language:

(i) Unformatted I/O
(ii) Formatted I/O

The given chart shows different input and output functions used in C programming language

Unformatted I/O

→ The unformatted functions are not capable of controlling the format that is involved in writing and reading the available data

→ The supply of input or the display of output isn't allowed in the user format. Thus, we call these functions unformatted functions for input and output.

Date:
Page:

→ The unformatted I/O have further two categories.
i) The character function
ii) The string function

character I/O
We use the character input functions for reading only a single character from the input device (the keyboard)
on the other hand, we use the character output functions for writing just a single character on the output screen.

Unformatted Input functions ~~for~~ for character
i) getch(), getchar() and getc()
unformatted output functions for character
(i) putch() and putchar()

getchar() and putchar()
The getchar() function reads a character (single character) from a standard input devices.
Syntax: character_variable = getchar();

The getchar function makes wait until a key is pressed and then assigns this character to character-variable.

The putch() function displays a character to the standard output device.
Syntax: putchar(character_variable)

WAP to read a character from keyboard using getchar() function and display on the screen using putchar() function.

```
#include <stdio.h>
#include <conio.h>
void main()
{
   char gender;
   printf("Enter your gender as M or F");
   gender = getchar();
   printf("Your gender is:");
   putchar(gender);
   getch();
}
```

getche(), getch() and putch()

getch(): This is a function that gets a character from keyboard but does not echo to the screen. It allows a character to be entered without having to press the Enter key afterwards.
Syntax: character_variable = getch();

getche(): This is a function that gets a character from the keyboard and echoes to the screen. It allows characters to be entered without having to press the Enter key afterwards.
Syntax: character_variable = getche();

putch(): This function prints a character on the screen
Syntax: putch(character_variable)

Date:
Page

String I1O
In any programming language, the character array or string refers to the collection of various characters.
The gets() and puts() are the used unformatted input and output functions used in C programming language

gets() and puts() function
gets(): The gets() function is used to read string of text, containing white space, until a newline is encountered.
Syntax: gets(string-variable);

puts(): This function is used to display the string on the screen.
Syntax: puts(string-variable);

Formatted I1O
Formatted I1O are used to take various inputs from the user and display multiple outputs to the user.
These types of I1O functions can help to display the output to the user in different formats using format specifier
Scanf() and printf() are the formatted input and output functions respectively used in C programming language

scanf()

scanf() function is used as formatted input and its general form is:

scanf("control string", arg1, arg2, ... argn);

The general form of control string is:

→ [whitespace_character][ordinary_character]%[field_width][conversion_character]

→ Whitespace character [optional]

scanf() reads whitespace character but doesnot store till next non-whitespace character in the input.

→ Ordinary character [optional]

scanf() reads ordinary character like / in certain pattern.

→ Field width:

specifies the maximum number of characters to read.

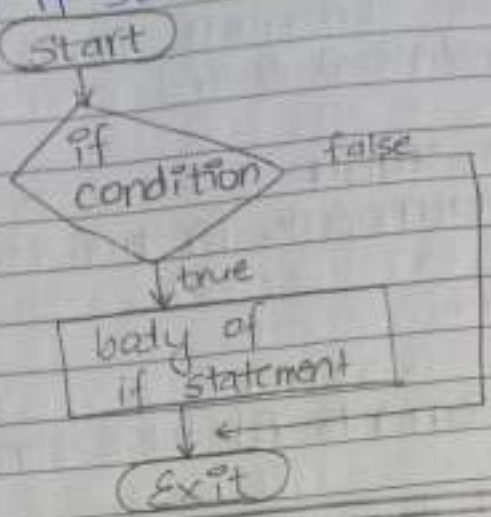
→ conversion character:

conversion character ~~on the basis~~ on the basis of the datatype to be showed ~~to t~~ i.e displayed on the output screen/ for a variable whose value is going to be displayed

printf()
printf() function is used to display data in a particular format. Its general form looks like:
printf("control string", arg1, arg2, ... ,argn);
The control string has the form:
%[flag][fieldwidth][.precision][conversion character]

Control Structures
⊛ Branching Statement
The statements that are used to make decisions based on the conditions are known as branching statements. They are also known as conditional statements. There are different types of branching statements such as
i) if statement.
It executes the block of code within it if the condition evaluated by it is true.
Syntax of if-statement.
```
if (condition)
{
  statements under if block;
}
```

Flowchart of if-statement



WAP to input the data from user and check whether the entered number is greater than 10 or not. If greater display the message "Your entered number is greater than 10"

Ans
```
#include <stdio.h>
#include <conio.h>
int main()
{
  int a;
  printf("Enter a number: ");
  scanf("%d", &a);
  if (a>10)
  {
    printf("Your entered number is greater than10");
  }
  return 0;
}
```

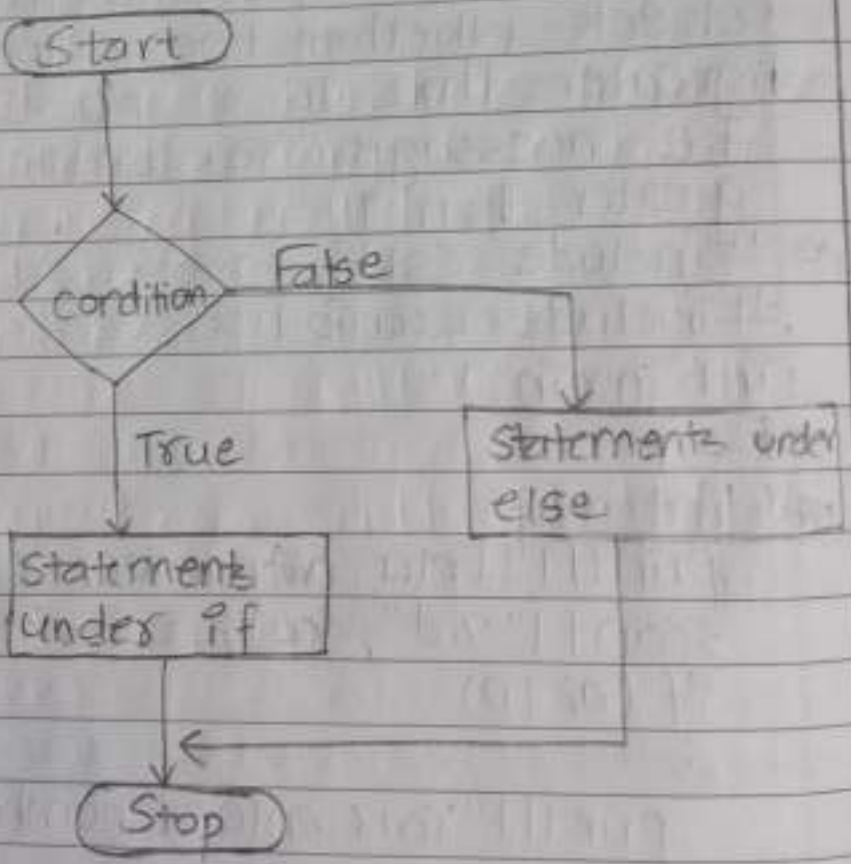
Date:
Page:

If-else Statement

Syntax:

```
if (condition)
{
    statements for true condition;
}
else
{
    statements of false condition;
}
```

Flowchart:



WAP to test whether a number is greater than 10 or not and display the message if it is smaller or greater than 10 or not

Ans

```
#include <stdio.h>
#include <conio.h>
void main()
{
  int num;
  printf("Enter a number: ");
  scanf("%d",&num);
  if (num>10)
  {
    printf("The number is greater than 10");
  }
  else
  {
    printf("The number is not greater than
           10");
  }
  getch();
}
```

Nested-If statement

An if statement under another if statement is called nested if statement.

Syntax:

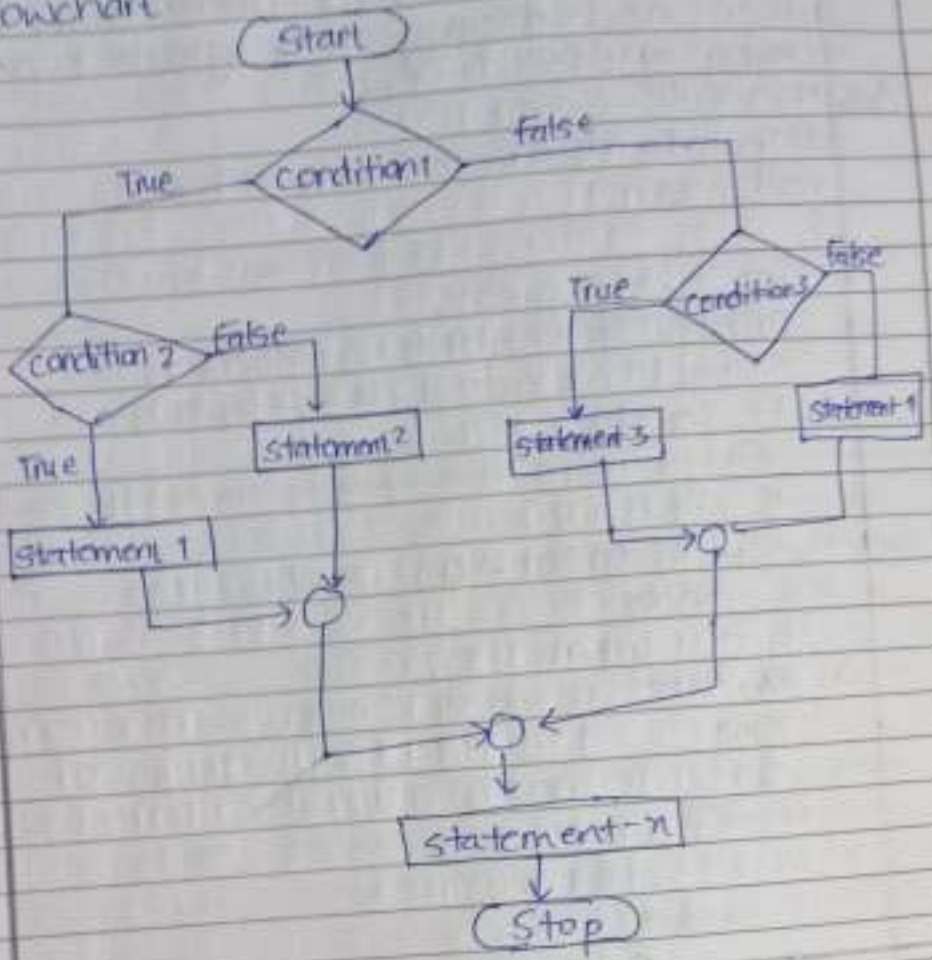
```
if (condition)
{
    if (condition-2)
    {
        statements;
    }
        :
        :
}
```

WAP to read three numbers from user and determine the largest number among them using nested if.

Ans

```
#include <stdio.h>
#include <conio.h>
void main()
{
  int x,y,z,great;
  printf("Enter three different numbers");
  scanf("%d%d%d",&x,&y,&z);
  if (x>y)
  {
     if (x>z)
      great = x
      else
       great=z
  }
  else
    great=y
  printf("The greatest number is %d",great);
}
```

Flowchart


Start
condition 1
True
False
condition 2
False
True
statement 2
statement 1
condition 3
True
False
statement 3
statement 4
statement-n
Stop

WAP to ask three numbers from the user and display the greatest number using a nested if.

Ans
```
#include <stdio.h>
#include <conio.h>
void main()
{
  int a,b,c,great;
  printf("Enter three numbers");
  scanf("%d%d%d",&a,&b,&c);
  if (a>b)
  {
     if (a>c)
        great=a;
     else
        great = c;
  }
  else
  {
     if (b>c);
        great=b;
     else
        great =c;
  }
  printf("The greatest number is %d",great);
  getch();
}
```

Else-if ladder

Syntax:

```
if (condition 1);
{
  statements for true condition1
};
else if (condition 2);
  {
   statements for ~~send~~ true condition 2 and
   false condition1
  };
```

WAP that reads total marks of a student in seven subjects. Then calculate percentage and determine the division use the following conditions.

1) Percentage greater than or equal to 80 → Distinction.
2) Percentage between 60 and 79 → First division
3) Percentage between 45 and 59 → ~~Third~~ Second division
4) Percentage between 32 and 44 → Third division
5) Percentage less than or equal to 31 → Fail.

Ans

```
#include <stdio.h>
#include <conio.h>
void main()
{
  int a,b,c,d,e,f,g,tot,per;
  char div[20];
  printf("Enter marks in seven subjects:");
  scanf("%d%d%d%d%d%d%d",&a,&b,&c,&d,&e,&f,&g);
  tot = a+b+c+d+e+f+g;
```

```
per = tot/7;
if (per>=80)
    grade = "Distinction";
else if (per>=60)
    grade = "First Division";
else if (per>=45)
    grade = "second Division");
else if (per>=32)
    grade = "Third Division";
else
    grade = "Fail";
printf(" Grade = %s", grade);
getch();
}
```

Switch statement

- A switch statement allows the user to choose a statement (or a group of statements) among several alternatives.

Syntax:

```
switch(variable name)
{
  case caseconstant1:
       statements;
       break;
  case caseconstant2:
       statements;
       break;
  :
  default:
       statements;
}
```

Example:

```
int choice;
printf("Enter 1 for File, 2 for Edit and 3
        for save");
scanf("%d", &choice);
switch (choice)
{
  case 1:
   printf("You have chosen File menu
           item");
   break;
   case 2:
    printf("You have chosen Edit menu
            item");
    break;
    case 3:
     printf("You have chosen save menu
             item");
     break;
    default:
      printf("Invalid option");
}
```

Page

WAP that displays
1. Addition
2. Subtraction
3. Multiplication
4. Division
and perform the operation by using switch statement.

Ans
```
#include <stdio.h>
#include <conio.h>
main()
{
    int num1,num2,choice,sum,diff,prod,div;
    printf(" Enter two numbers: ");
    scanf("%d%d",&num1,&num2);
    printf(" Choose the operation as follow\n");
    printf(" 1.Addition\n");
    printf("2 .Subtraction\n");
    printf(" 3.Multiplication\n");
    printf(" 4.Division\n");
    printf("Enter your choice: ");
    scanf("%d",&choice);
    switch(choice)
    {
    case 1:
        sum=num1+num2;
        printf(" The sum of %d and %d is
                %d",num1,num2,sum);
    break;

    case 2:
      diff=num1-num2;
```

```
printf("The difference of %d and %d is
        %d", num1, num2, diff);
break;

case 3:
   prod = num1*num2;
   printf("The product of %d and %d is
           %d", num1, num2, prod);
   break;

case 4:
   div = num1/num2;
   printf("The quotient of when %d is
          divided by %d is %d", num1, num2, div);
   break;

default:
   printf("Invalid option");
}
```

Go-to statement

A goto statement provides an unconditional jump from the 'goto' to a labeled statement in the same function.

Syntax:

```
goto label;
.....
.....
label: statements.
```

Example:

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
main()
{
   int num;
   printf("Enter a number:");
   scanf("%d",&num);
   if (n<0)
    {
      goto negative;
    }
   else
    {
      goto positive;
    }
   negative:
    printf("%d is negative",num);
    exit(0);
   positive:
    printf("%d is positive",num);
}
```

WAP that display "We are enjoying C-Programming" for 5 times

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

```
for(i=1; i<=5; i++)
{ printf("We are enjoying C-programming\n");
}
}
```

Looping statements.

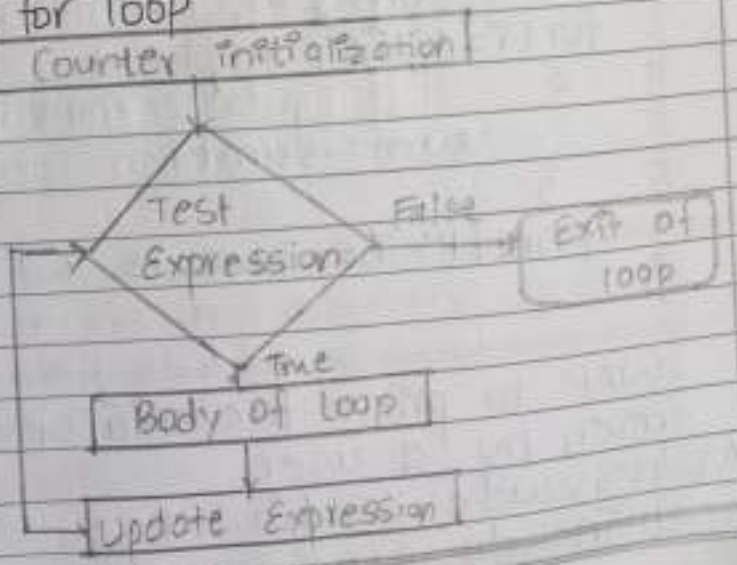
1) For loop

This loop is also known as definite loop or determinate loop. The for loop is useful to execute a statements for a finite number of times.

Syntax:

```
for(counter initialization; test condition;
        increment or decrement)
{
    body of loop
}
```

Flowchart of for loop



Date:
Page:

WAP to print natural numbers from 1 to 10.

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

WAP to print sum of the natural numbers from 1 to 10.

Ans
```
#include <stdio.h>
#include <conio.h>
main()
{
  int i,sum=0;
  for(i=1;i<=10;i++)
   {
      sum=sum+i;
   }
  printf("The sum is %d",sum);
}
```

WAP to print factorial ~~of~~ of a number given by a user.

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

```
main()
{  long int fact=1;
   int num,i;
   printf("Enter a number: ");
   scanf("%d",&num);
   for(i=1;i<=num;i++)
   {
      fact=fact*i;
   }
   printf("Factorial of %d is %d",num,float);
}
```

2) While Loop

The test condition is evaluated first and if the condition is true the body of the loop is executed. It is an entry-controlled loop.

Syntax:

```
while(test condition)
{
   body of loop;
   increment/decrement variable;
}
```

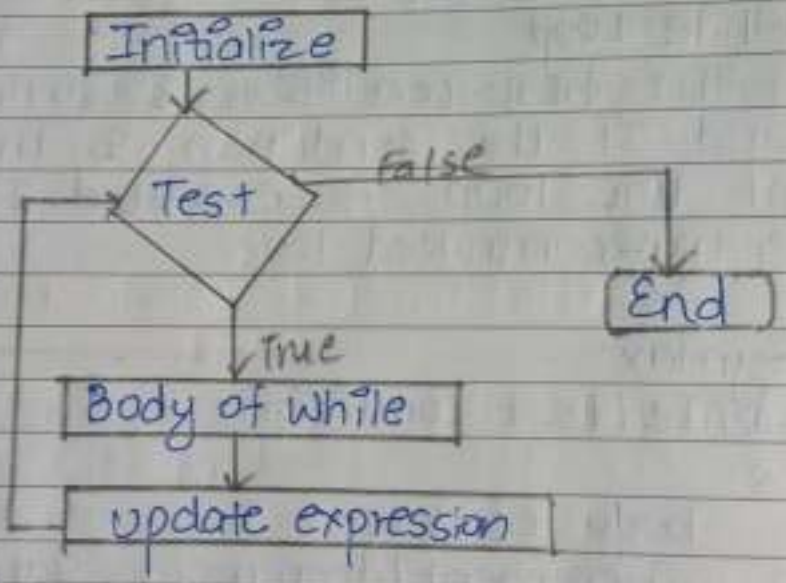
WAP to print natural numbers upto 10 by using while loop.

Ans

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

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

Flowchart of while-loop



WAP to enter 4 digit number and display sum of its digits by using while loop.

Ans
```
#include <stdio.h>
#include <conio.h>
int main()
{
  int num, rem, sum=0, rev; sum=0, rev;
  printf("Enter a 4 digit number: ");
  scanf("%d", &num);
  while(num!=0)
  {
    rem = num %. 10;
    sum = sum + rev;
    num = num / 10;
  }
  printf("The sum of digits is %d", sum);
  getch();
}
```

Do-While Statement

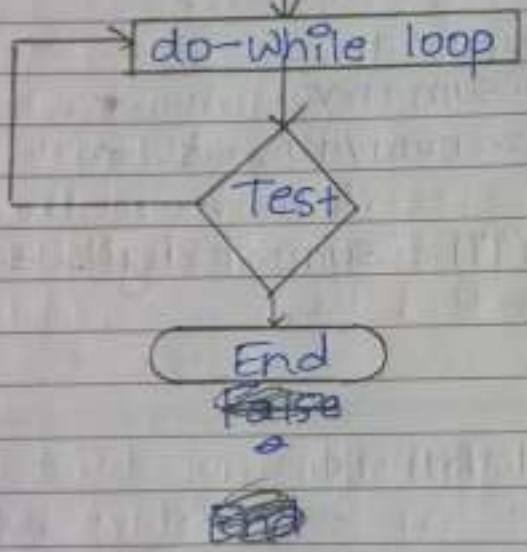
Do-while loop execute a body (i.e. block of statement) first without checking any condition and then check a test condition to determine whether the body of loop is to be executed for next time or not. Hence, do-while loop execute its body at least once even if the condition is fail at all. Do-while loop is exit controlled loop.

Date:
Page:

Syntax:

```
do
{
  //code to be executed at least once if
     the condition is t
}
while(condition);
```

Flowchart of do-while



WAP to read a number from keyboard until a zero or a negative number is keyed in. Finally calculate sum and average.

Ans

```
#include <stdio.h>
#include <conio.h>
int main()
{
    int num, count=0;
    float sum=0, avg;
    do
    {
      printf("\n Enter number ");
      scanf("%d", &num);
      sum = num + sum;
      count++;
    }while(num>0);
    sum=sum-num;
    avg= sum /(count-1);
    printf("The sum is %d", sum);
    printf("The avg is %f", avg);
    getch();
    return 0;
}
```

WAP to count number of digits in a number.

Ans Solution

```
#include <stdio.h>
#include <conio.h>
void main()
{
   int num, count=0;
   printf("Enter a number: ");
   scanf("%d",&num);
   do
   {
      num=num/10;
      count++;
   }while(num!=0);
   printf("The no.of digits is %d",count);
   getch();
}
```

WAP to find sum of digits of a number.

Ans

```
#include <stdio.h>
#include <conio.h>
void main()
{
   int num,rem,sum=0;
   printf("Enter a number: ");
   scanf("%d",&num);
   do
   {
      rem=n%10;
      sum=sum+rem;
```

```
    num = num / 10;
  } while(num != 0);
  printf("The sum of digits is %d", sum);
  getch();
}
```

Nested Loop

Syntax (Nested for loop)

```
for(initialization, condition, updation)
{
  for(initialization; condition, updation)
  { //statements of inner loop
  }
  //statements of outer loop
}
```

WAP to display the given output.

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

Ans

```
#include <stdio.h>
#include <conio.h>
void main()
{
  int i, j;
  for(i=0; i<=4; i++)
  {
    for(j=4; j>i; j--)
    {
      printf("*\t");
```

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

Syntax: (Nested while loop)

```
while(condition)
{
    while(condition)
    {
    //statements of inner loop
    }
    //statements of outer loop
}
```

Array and Pointer

Introduction to an Array

An array is a group of related data items that share a common name. The individual data items are called elements and all of them have same data types.

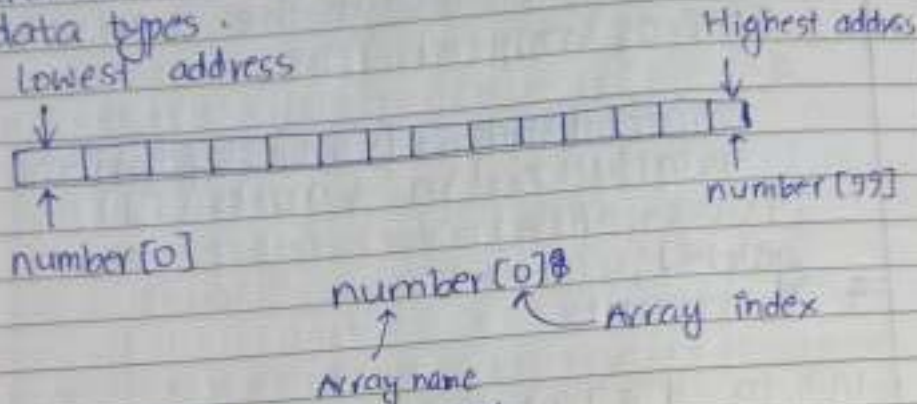


Fig: Array concept

One-Dimensional Array:

Syntax: data-type array-name[size];

For example: int a[5];

WAP to store marks of 5 subjects of a student.

Ans
```
#include <stdio.h>
#include <conio.h>
int main()
{
    int marks[5];
    int i;
```

```
for(i=0;i<5;i++)
{
    printf("Enter the marks of subject %d:", i+1);
    scanf("%d",&marks[i]);
}
printf("\n Displaying the marks:\n");
for(i=0;i<5;i++)
{
    printf("marks of subject %d:", i+1);
    printf("%d\n", marks[i]);
}
getch();
}
```

WAP to display the sum of all the number entered by user by using an array.

Ans
```
#include<stdio.h>
#include<conio.h>
int main()
{
    int num[100], sum=0, i, n;
    printf("Enter amount of numbers you want to enter");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        printf("Enter a number");
        scanf("%d",&a[i]);
        sum t=a[i];
    }
    printf("The sum is: %d", sum);
```

```
    getch();
}
```

WAP that reads marks in an examination of 10 student. calculate and display the average percentage and deviation percentage of each student.

```
#include <stdio.h>
#include <conio.h>
int main()
{
  float student[10], tot=0, avg, i;
  for(i=0; i<10; i++)
    {
      printf("Enter percentage of student %d: ", i+1);
      scanf("%d", &student[i]);
      tot += student[i];
    }
    avg = tot/10;
    printf("The average percentage is %f", avg);
  for(i=0; i<10; i++)
  {
    printf("Deviation of student %d is %f\n", i+1,
                                    student[i]-avg);

  }
  getch();
}
```

Two-Dimensional Array

The 2D array is organized as matrices which can be represented as the collection of rows and columns.

Syntax:

```
data_type array_name[rows][columns];
```

Example:

```
int twodimen[4][3];
```

WAP to read element of 2x2 matrix and display it as in matrix form.

Ans

```
#include <stdio.h>
#include <conio.h>
void main()
{
  int a[2][2], i, j;
  printf("E
  for(i=0; i<2; i++)
   {
     for(j=0; j<2; j++)
       {
         printf("Enter a number: ");
         scanf("%d", &a[i][j]);
       }
   }
  printf("Displaying in matrix form.\n");
  for(i=0; i<2; i++)
  {
    for(j=0; j<2; j++)
    {
      printf("%d\t", a[i][j]);
```

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

WAP to display sum of two matrices.

```
#include <stdio.h>
#include <conio.h>
int main()
{
   int a[2][2],b[2][2],c[2][2],i,j;
   for(i=0;i<2;i++)
    {
      for(j=0;j<2;j++)
       {
         printf("Enter an element: ");
         scanf("%d",&a[i][j]);
       }
    }
   printf("For second matrix\n");
   for(i=0;i<2;i++)
   {
     for(j=0;j<2;j++)
      {
        printf("Enter an element:");
        scanf("%d",&b[i][j]);
      }
   }
   for(i=0;i<2;i++)
   {
```

```
    for(j=0;j<2;j++)
    {
        c[i][j]=a[i][j]+b[i][j];
    }
}printf("The matrix addition is:\n");
for(i=0;i<2;i++)
{
    for(j=0;j<2;j++)
    {
        printf("%d\t",c[i][j]);
    }
    printf("\n");
}
getch();
}
```

WAP to determine whether a number is divisible by 5 or not & divisible by 7 but not by 11

num = 1.5 ↵

Date:
Page:

String.

String is an array of character (i.e characters are arranged one after another in memory) string is also known as character-array.

A string is always terminated by a null character (i.e \0). The terminating null character is important because it is only way for string handling function to know the end of string.

String declaration

char string_name[size]

For eg: char name[20];

C-String Initialization.

1. Assigning a string literal without size
   char str[] = "string";
2. Assigning a string literal with size
   char str[6] = "string";
3. Assigning character by character with size
   char str[6] = {"s","t","r","i","n","g"};
4. Assign character by character without size
   char str[] = {"s","t","r","i","n","g"}

The last index of string is always reserved for a null character. So, to ensure that all strings are taken, the size of string must be declared higher by 1.

WAP to take full name of a person and display the name.

Ans
```
#include <stdio.h>
#include <conio.h>
int main()
{
    char name[50];
    printf("Enter your name: ");
    gets(name);
    puts(name);
    return 0;
}
```

String Handling Functions.

1) strlen()

→ This function is used to find out the length of the string excluding null character.

→ Syntax:

integer_value = strlen(string1)

For eg: len = strlen(name1)

WAP to find the length of string without using strlen()

Ans
```
#include <stdio.h>
#include <conio.h>
#include <string.h>
void main()
{
    char string[50];
    int i=0;
    while(string[i]   int i=0;
```

Date: ______
Page: ______

```
printf("Enter a string: ");
scanf(" gets(string);
While(string[i] != '\0')
{
    i++;
}
printf("The st length of string is %d", i);
getch();
}
```

WAP to find the length of string by using strlen() function.

Ans.
```
#include <stdio.h>
#include <conio.h>
#include <string.h>
void main()
{ int length;
  char str[20];
  printf("Enter a string: ");
  gets(str);
  length = strlen(str);
  printf("The length of string is %d",
                                   length);

  getch();
}
```

2) strcpy()
→ This function copies one string to another
→ Syntax:
strcpy(destination_string, source_string)
For: strcpy(S_1, S_2) ⇒ This copies the contents of S_2 to S_1.

Mini-program:
```
int main()
{
  char name[60] = "Saroj Bhatta", s[15];
   strcpy(s, name); puts(s);
   return 0;
}
```

3. strcat() function.
→ This function concatenates two string (i.e. it appends one string to another).
→ Syntax:
strcat(string1, string 2)
This is equivalent to string1 = string1 + string2
Mini-program
```
int main()
{
char firstname[20] = "Ram", lastname[20] = "Bhatta";
strcat(firstname, lastname);
printf("%s", firstname);
}
```

WAP to concatenate two strings using without using strcat() function.

Ans
```
#include <stdio.h>
#include <conio.h>
#include <string.h>
int main()
{
  char str1[50], str2[50], str3[50];
  int len1, len2, i, j
  printf("Enter 1st string: ");
  scanf("%s", str1);
  printf("Enter 2nd string: ");
  scanf("%s", str2);
  len1 = strlen(str1);
  len2 = strlen(str2);
  for(i=0; i<len1; i++)
    {
      str[3]=str[i]; str3[i]=str1[i]
    }
  for(j=len1; j<(len1+len2); j++)
    {
      str[j]=str[j-len1];
    }
  str[j]='\0';
  for(i=0; i<strlen(str3); i++)
  printf("The concatenated string is: %s", str3);
  getch();
  return 0;
}
```

Strcmp()

→ This function compares two string to find out whether they are same or different. The function accepts two strings as parameter and return an integer whose value is

i) less than 0 (-1 in some compiler) if the first string is less than second.
ii) equal to 0 if both are same.
iii) greater than 0 (1 in some compiler) if the first string is greater than second.

→ Syntax:

integer-variable = strcmp(string1, string2)

Mini-program

```
void ~~int~~ main()
{
    int diff;
    char a[5]="abc";
    char b[5]="efg";
    diff=strcmp(a,b);
    if (diff>0)
    {
        printf("%s comes before %s", ~~a~~ b,a);
    }
    else
    {
    printf("%s comes before %s",a,b);
    }
    getch();
}
```

WAP to reverse the
strrev()
→ This function is used to reverse all the character at the end of string.
Syntax : strrev(string)

WAP to reverse the string with and without string handling function.

Ans //using strrev() function
```
#include <stdio.h>
#include <conio.h>
#include <string.h>
void main()
{
  char string[50];
  printf("Enter a string: ");
  gets(string);
  strrev(string)
  printf("The reverse is: %s", strrev());
  getch();
}
```

Ans //Without using strrev() function
```
#include <stdio.h>
#include <conio.h>
void main()
{
  int i,j,len=0,
  char str[50], rev[50];
  printf("Enter a string: ");
  gets(str);
  i=0;
```

```
while(str[i] != '\0')
{
    len++;
    i++;
}
for(i=0; i<len; i++)
{
    rev[i] = str[len-1-i];
}
rev[i] = '\0';
printf("The reverse is: %s", rev);
getch();
}
```

WAP to read a string and check whether it is palindrome or not using strrev().

Ans
```
#include <stdio.h>
#include <conio.h>
void main()
{
    char str[100];
    int len, i, pal=1;
    printf("Enter a string: ");
    gets(str);
    for(i=0; i<(len/2); i++)
    {
        if (st[i] != st[len-i-1])
            pal=0
    }
    if (pal==0)
        printf("Not palindrome.");
    else
        printf("Palindrome");
    getch();
}
```

strupr function

→ This is used to convert a string into uppercase format.

→ Syntax: ~~strupr~~()
strupr(string)

strlwr() function

→ This is used to convert a string into lowercase ~~forwa~~ format.

Pointer.

Pointers are special variable that are used to store address ~~variable~~ rather than value. A pointer is a variable pointing to the address of another variable. It is declared along with an asterisk symbol (*).

Syntax to declare a pointer

datatype *variablename

Syntax to assign the address of variable to a pointer

datatype variable1; *variable 2

variable 2 = &variable1

After declaring a pointer, pointer is initialized as

pointer_name = &variable.

For example:

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

Wrong Method

```
float b=1.1
int *p;
p=&b;
```

Mini program

```
#include <stdio.h>
int main()
{
 int a=10;
 int *p;
 p=&a;
printf("Address stored in a variable p is
        %x\n",p);
printf("Value stored in a variable p is
        %d\n",*p);
```

```
    return 0;
}
```

Mini-program

```
int main()
{
  int a=10, b=9, c;
  int *p, *q;
    p= &a;
    q= &b;
    c= *p;
printf("Value of a = %d", *p);
printf("Value of b = %d", *q);
printf("Address of a = %x\n", &a);
printf("Address of a = %x\n", p);
printf("c = %d", c);
  getch();
}
```

Type of pointer

Void pointer

A void pointer is a special type of pointer. It can point to a variable of any data type. Using void pointer of ~~any data type~~ the pointed data cannot be referenced directly. The type casting or assignment must be used to change the void pointer to a concrete data type to which we can refer.

For example:

```
int main()
{
  int a=10;
  double b=4.5;
  void *vptr;
  vptr=&a;
  printf("\n a=%d",*(int*)vptr);
  vptr=&b;
  printf("\n b=%f",*(double*)vptr);
  getch();
  return 0;
}
```

NULL Pointer

A NULL pointer is a special pointer value that points nowhere or nothing in memory address.

```
int *ptr=NULL
```

Double pointer (pointer-to-pointer)

The use of pointer that points to other pointer and these, in turn point to data.

For example:

```
int a=20;
int *p;
int **q;
p=&a;
q=&b;
printf("%d",*p);
```

printf("%d", **q);

Pointer Arithmetic

Pointer arithmetic is the set of valid arithmetic operations that can be performed on pointers.

These operations are:

- Increment / Decrement of a pointer.
- Addition of integer to a pointer.
- Subtraction of integer to a pointer.
- Subtracting two pointers of same type.
- Comparision of pointers.

Increment of a Pointer

It is a condition that also comes under addition. When a pointer is incremented, it actually increments by the number equal to the size of the data type for which it is a pointer.

Decrement of a Pointer

It is a condition that also comes under subtraction. When a pointer is decremented, it is actually decrements by the number equal to the size of the data type for which it is a pointer.

Addition of integer to a pointer

When a pointer is added with an integer value, the value is first multiplied by the size of the data type and then

added to the pointer.

Subtraction of an integer from a pointer.
When a pointer is subtracted with an integer value, the value is first multiplied by the size of the data type and subtracted from the pointer.

Subtracting two points of the same type
The subtraction of two pointers is possible only when they have the same data type. The result is generated by calculating the difference between the addresses of the two pointers and calculating how many bits of data it is according to the pointer data type. The subtraction of two pointers gives the increments between the two pointers.

Array of pointer
Syntax:
datatype *pointername[size];
For example:
int *p[20];

WAP to read marks of 10 student secured in a subject and store them in an array. Calculate average marks and display it.

Ans
```
#include <stdio.h>
#include <conio.h>
void main(){
```

```
int marks[10], sum=0, avg, i
for(i=0; i<10; i++)
{
  printf("Enter marks: ");
  scanf("%d", &marks[i]);
  sum = sum + marks[i];
}
avg = sum/10;
printf("The average marks is %d", avg);
getch();
}
```

WAP to read marks of 10 students served in a subject and store them in an array. Calculate average marks and display it using array pointer.

Ans
```
#include <stdio.h>
#include <conio.h>
void main()
{
  int *marks[10], sum=0, avg, i;
  for(i=0; i<10; i++)
  {
    printf("Enter marks: ");
    scanf("%d", &a+i);
    sum = sum + *(a+i);
  }
  avg = avg/10;
  printf("The average marks is %d", avg);
  getch();
}
```

WAP to sort 5 numbers in ascending order using array.

Ans
```
#include <stdio.h>
#include <conio.h>
int main()
{
   int num[5], i, j, temp;
   for (i=0; i<5; i++)
   {
     printf("Enter a number: ");
     scanf("%d", &num[i]);
   }
   for(i=1; i<5; i++)
   {
      for(j=0; j<i; j++)
        {
           if (a[i]<a[j])
           {
              temp=a[i];
              a[i]=a[j];
              a[j]=temp;
           }
        }
   }
   for(i=0; i<5; i++)
   {
     printf("%d\t", &num[i]);
   }
   getch();
   return 0;
}
```

WAP to read the string from user and count the number of character in that string without using any string handling function.

Ans
```
#include <stdio.h>
#include <conio.h>
void main()
{
   char str[100];  printf("Enter string: "); gets(str);
   int n=0; printf("Enter string: "); gets(str);
   while(str[n]!='\0')
   {
      n++;
   }
   printf("The number of characters is: %d",n);
   getch();
}
```

WAP to find the vowel characters in the string and count the number of vowel characters in it.

Ans
```
#include <stdio.h>
#include <conio.h>
#include <string.h>
void main()
{
   char str[100];
   int vow=0,n=0;
   while(str[n]!='\0'){
      if(str[n]=='a' || str[n]=='e' ||
      str[n]=='i' || str[n]=='o'
      || str[n]==
```

```
printf(" Enter string: ");
gets(str);
strlwr(str);
while(str[n]!='\0')
{
  if (str[n]=='a' || str[n]='e' || str[n]=='i'
        || str[n]=='o' || str[n]=='u')
    vow++;
  n++;
}
printf(" The no. of vowel character is: %d",vow);
getch();
}
```

Without using string handling function

```
#include <stdio.h>
#include <conio.h>
void main()
{
  char str[100];
  int n,=0,vow=0;
  printf(" Enter the string: ");
  gets(str);
  while (str[n]!='\0')
  {
   if (str[n]=='a' || str[n]=='e' || str[n]='i'
        || str[n]=='o' || str[n]=='u'
     || str[n]=='A' || str[n]='E' || str[n]=='I'
      || str[n]=='O' || str[n]=='U')
    vow++;
  n++;
```

```
}
printf("The number of vowel characters is: %d",vow);
getch();
}
```

Method 2:

```
#include <stdio.h>
#include <conio.h>
void main()
{
  int n=0,i,vow=0;
  Char str[100],check[]={'a','e','i','o','u','A',
  printf("Enter string: "); gets(str);
                          'E','I','O','U'}
  While[str[n]!='\0')
  {
     for(i=0;i<10;i++)
     {
        if(str[n]==check[i])
           vow++;
     }
     n++;
  }
  printf("The number of vowel characters is: %d",
           vow);
  getch();
}
```

WAP to convert uppercase into lowercase of all the characters of string without using string handling function.

Ans
```
#include <stdio.h>
#include <conio.h>
void main()
{
   char str[100]
   int n=0,p;
   printf("Enter a string: ");
   ~~scanf("%s",~~ gets(str);
   while(str[n]!='\0')
   {
      p=str[n];
      if(p<=90){
         p=p+32;
         str[n]=p;
      }
   n++;
   }
printf("The modified string is: %s", str);
getch();
}
```

WAP to ~~find~~ read ~~[illegible]~~ matrix and find out smallest element of matrix by using pointer.

Ans
```
#include <stdio.h>
#include <conio.h>
void main()
{
```

```
int *a[2];
int i,j,min;
for(i=0;i<2;i++)
{
    for(j=0;j<2;j++)
    {
        printf("Enter an element: ");
        scanf("%d",((a+i)+j));
        if(i==0 && j==0)
            min = *((a+i)+j);
        if(*((a+i)+j) < min)
            min = *((a+i)+j);
    }
}
printf("The minimum value is: %d", min);
getch();
}
```

Relation between Array and pointer.

1) Relation between 1-D Array and pointer.

1-D array is the arrangement of data of same-type in the adjacent memory whose address in the memory differ by the size of datatype that they hold. The notation &a[0] is same as a or (a+0) in case of array. which means the array name in case of 1-D array represents the base address of the array or the address in which the first element is stored.

For example:

If a[10] is a one-dimensional array, then a[9] is same as *(a+9). This is because a+9 represents the address of a[9] while the deferencing operator * is used to return the value stored in the address a+9.

Mini-Program

```
main()
{ int i;
  int a[10],*p;
  p=a;
  for(i=0;i<10;i++)
  {
    printf("Enter a number: ");
    scanf("%d",(p+i));
  }
}
```

This program stores element of an array by

2) Rela

the use of pointer.

2) Relation between 2-D Array and pointer.
2-D array is arrangement of 1-D array in a form of row. The base address of the array can be written as *array-name or *(array-name + 0). Similarly, the ith element in the jth 1-D array can be written as *(*(array-name+j-1)+i-1). This means that &a[i][j] is same as *(*(a+i)+j). It is to be noted that pointer can be used to access the elements of the array by assigning the base address of the array to the pointer and then by using appropriate pointer arithmetic, each and every element in 2-D array can be accessed.
For example:
This program shows the use of pointer to store elements in 2-D array.
Mini-program:

```
main()
{
  int *p,i,j,a[10][10];
  p=*a;
  for(i=0;i<10;i++)
  {
    for(j=0;j<10;j++)
     {
       scanf("%d",*(p+i)+j);
     }
}}
```

User Defined Functions.

Function:
A function is defined as a self-contained block of statements that performs a particular task.

Type of functions:
→ Library Functions
→ User-defined Functions.

Library function
→ Functions which are already defined in c Library.
→ Example: printf(), scanf(), strcmp(), strcat(), etc.
→ We just need to include appropriate header files to use these functions.

User-defined Functions.
→ Functions defined by the user. to perform a specific task.
→ It can be used anywhere in the program.

Advantages of Functions.
→ It utilizes abstract implementation of the code.
→ It enhanches code reusability and code modularization.
→ It improves code readability and code maintainability.
→ It helps to write cleaner, more efficient and more manageable code.

Date:
Page:

How function works in C programming?

```
#include <stdio.h>
void functionname()
{
    .....
    .....
}
int main()
{
    .....
    .....
    functionname();
    .....
}
```

Elements of a User-Defined Function

→ Function Definition
→ Function Prototype
→ Function Parameters.

Function Definition

A C function is generally defined and declared in a single step because the function definition always starts with the function declaration so we do not need to declare it explicitly.

Syntax:

```
return-type function-name(type variable-list,...)
{
    // body of function
}
```

Function Prototype/Function Declaration

→ A blueprint of a function that provides a compiler the following information:

- The name of the function.
- The type of the value returned by the function.
- The number and the type of the arguments that must be passed while calling of the function.

→ If a function is used before it is defined in the program, then function declaration is needed otherwise not.

Syntax:

return_type function-name(type1, type2, ...)

For example:

```
int sum(int a, int b);
```

- int (return type)
- sum (function name)
- a (Parameter Name)
- int (Parameter Type)
- b (Parameter Name)
-); (Ending statement Semicolon)

Date:
Page:

Function Parameters / Function Call

A function call is a statement that instructs the compiler to execute the function.

Syntax:

```
functionName(parameter1, parameter2, ..... );
{
  //code to be executed
}
```

WAP to read two numbers and display the sum of two numbers without using function

Ans

```
#include <stdio.h>
#include <conio.h>
int main()
{
  int a, b, sum;
  printf("Enter two numbers: ");
  scanf("%d %d", &a, &b);
  sum=a+b;
  printf("The sum is: %d", sum);
  getch();
  return 0;
}
```

By using function

```
#include <stdio.h>
#include <conio.h>
ret int sum(int x, int y)
{
  return x+y;
}
```

```
int main()
{ int a,b;
  printf("Enter two numbers: ");
  scanf("%d%d",&a,&b);
  printf("The sum is: %d", sum(a,b));
  getch();
  return 0;
}
```

WAP to check even or odd number using function.

Ans
```
#include <stdio.h>
#include <conio.h>
void eoo();
int main()
{
   eoo();
   getch();
   return 0;
}
void eoo()
{ int a;
  printf("Enter a number: ");scanf("%d",&a);
  if(a%2==0)
  {
    printf("Even");
  }
  else
  {
    printf("Odd");
  }
```

Date:
Page:

3

Actual Parameters	Formal Parameters.
→ Values or references that are passed into the function when it is called.	Default values, which are used if no value is passed in as an actual parameter when the function is called.
→ They are also called arguments or parameter values.	Formal parameters are also called function parameters or function arguments.
→ They provide input to the function.	They are the placeholders for the values or references that will be passed into the function.
→ They have a scope that is limited to the function call in which they are used.	They have a scope that is limited to the function in which they are defined.
→ Actual parameters can be expressed as the expressions that are evaluated before they are passed into the function.	They are used by the function to perform operations and computations on the value or references that are passed into the function as actual parameters.

void inverse(int x, int y) → Formal parameters
inverse(a,b) → Actual parameters.

WAP to find factorial of a number.

```
#include <stdio.h>
#include <conio.h>
void fact();
int main()
{
   fact();
   getch();
   return 0;
}
void fact()
{
  int num, i, fact=1;
  printf("Enter a number: ");
  scanf("%d",&num);
  for(i=1;i<=num;i++)
  {
    fact*=i;
  }
 printf("The factorial is: %d", fact);
}
```

With argument and with return type.

```
#include <stdio.h>
#include <conio.h>
int void sum(int,int);
void main()
{
   int a,b;
   printf("Enter two numbers: ");
   scanf("%d%d",&a,&b);
   printf("The sum is: %d",sum(a,b));
   getch();
}
int void sum(int x,int y)
{
   int sum=x+y;
   return sum; return sum;
}
```

With argument and without return type

```
#include <stdio.h>
#include <conio.h>
void sum(int,int);
void main()
{
  int a,b;
  printf("Enter two numbers: ");
  scanf("%d%d",&a,&b);
  sum(a,b);
  getch();
}
void sum(int x, int y)
{
```

```
    int sum=x+y;
    printf("The sum is: %d",sum);
}
```

Factorial of a Number

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

```
#include <stdio.h>
#include <conio.h>
int sum();
void main()
{
   int c=sum();
   printf("The sum is: %d",c);
   getch();
}
int sum()
{
   int a,b,c;
   printf("Enter two numbers: ");
   scanf("%d%d",&a,&b);
   c=a+b;
   return c;
}
```

```
#include <stdio.h>
#include <conio.h>
int sum(int,int);
void main()
{
   int a,b;
   printf("Enter two numbers: ");
   scanf("%d%d",&a,&b);
   printf("The sum is: %d ",sum(a,b));
   getch();
}
int sum(int x,int y)
{
   return x+y;
}
```

```
    scanf("%d%d", &a, &b);
    printf("The product is: %d", a*b);
    return a*b;
}
int div(int x, int y)
{
    int q = x/y;
    return q;
}

#include <stdio.h>
#include <conio.h>
void sum();
void sub(a,b);
void sub(int a, int b);
int mul();
int div(int a, int b);
void main()
{
    int a, b, c, m, d;
    printf("Enter your choice: ");
    scanf("%d", &c);
    switch(c)
    {
      case 1:
       { sum();
         break;
       }
      case 2:
       {
         printf("Enter two numbers: ");
```

```
scanf("%d%d", &a, &b);
sub(a,b);
break;
}
case 3:
{
    m=mul()
    printf("The product is %d", mul());
    break;
}
case 4:
{
    printf("Enter two numbers: ");
    scanf("%d%d", &a, &b);
    d=div(a,b);
    printf("The quotient is %d", d);
    break;
}
default:
{
    printf("Invalid choice.");
}
}
getch();
}
void sum()
{
  int a,b,s;
  printf("Enter two numbers: ");
  scanf("%d%d", &a, &b);
  s=a+b;
```

```
    printf("The sum is: %d",s);
}
void sub(int a,int b)
{
    int d=a-b;
    printf("The difference is: %d",d);
}
int mul()
{ int a,b,p;
    printf("Enter two numbers:");
    scanf("%d %d",&a,&b);
    p=a*b;
    return p;
}
int div(int a,int b)
{
    int q=a/b;
    return q;
}
```

Write a program that ~~return~~ follow the following condition.

1. sum() 2. ~~int~~ sub(a,b) 3. m=mul()
4. d = div(a,b);

By using switch case.

Ans
```
#include <stdio.h>
#include <conio.h>
void sum();
void diff(int,int);
int mul();
int div(a,b);
void main()
{
  int c,m,n;
  printf("Enter a choice(1-4): ");
  scanf("%d",&c);
  switch(c)
  {
   case 1:
    {
      sum();
      break;
    }
   case 2:
   {
     printf("Enter two numbers: ");
     scanf("%d%d",&m,&n);
     diff(m,n);
    break;
   }
   case 3:
```

Date:
Page

Recursive functions

→ Recursion is the technique of making a function call by itself.
→ Recursion is the process of a function calling itself repeatedly till the given condition is satisfied.
→ TO prevent infinite recursion, if.... else statement (or similar approach) can be used where one branch makes the recursive call, and other doesn't.

How does recursion work?

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

recursive call

function call

Basic structure of Recursive Functions

```
type function_name(args){
    //function statements
    // base condition
    // recursion case (recursive call) }
```

* The recursion continues until some condition is met to prevent it.
* A function that calls itself directly or indirectly is called a recursive function and such kind of functions calls are called recursive calls.

→ To solve the problem using recursive method the following condition must be satisfied.
- Problem could be written or defined in term of its previous result.
- Problem statement must include a stopping point or condition.

WAP to find sum of first n natural numbers using recursive function.

Ans
```
#include <stdio.h>
#include <conio.h>
int sum(int n);
int main()
{
   int num,r;
   printf("Enter a number: ");
   scanf("%d",num);
   r= sum(num);
   printf("The sum is: %d",r);
   getch();
   return 0;
}
int sum(int n)
{
```

```
if (n!=0)
   return n+sum(n-1);
else
   return n;
}
```

WAP to find the factorial number using recursive function.

Ans
```
#include <stdio.h>
#include <conio.h>
int fact(int n)
int main()
{
   int num;
   printf("Enter a number: ");
   scanf("%d", &num);
   printf("Factorial is: %d", fact(num));
   getch();
   return 0;
}
int fact(int n)
{
   if (n==1)
      return 1; else if (n==0) return 1;
   else
      return n*fact(n-1);
}
```

Function call by Values and Reference

The parameters of function can be passed in two ways:

- Call by value: A copy of the variable is passed to the function.
- Call by reference: An address of the variable is passed to the function

Call by value in C

- Calling a function by value will cause the program to copy the contents of an object passed into a function.

Write a program to swap values of two variables using call by value method.

Ans
```
#include <stdio.h>
#include <conio.h>
void swap(int, int);
int main()
{
    int a, b;
    printf("Enter two numbers: ");
    scanf("%d %d", &a, &b);
    printf("Before swapping:\n");
    printf("%d %d", a, b);
    swap(int, int);
    getch();
    return 0;
}
void swap(int x, int y)
{
```

```
int t=x;
x=y;
y=t;
printf("After swapping:\n");
printf("%d %d",x,y);
return void;
}
```

Call by reference in C

Calling a function by reference will give function parameter the address of original parameter due to which they will point to same memory location and any changes made in the function parameter will also reflect in original parameters.

Write a program to swap values of two variables by using call by reference method.

Ans
```
#Include <stdio.h>
#Include <conio.h>
void swap (int *, int *)
int main()
{
    int a,b;
    printf("Enter two numbers: ");
    scanf("%d %d",&a,&b);
    printf("Before swapping:\n");
    printf("%d %d",a,b);
    swap(&a,&b);
    printf("After swapping:\n");
    printf("%d %d",a,b);
```

```
    getch();
    return 0;
}
void swap(int *p, int *q)
{
    int temp = *p;
    *p = *q;
    *q = temp;
    return void;
}
```

Passing Array to a Function

Passing elements of an array:

Syntax:

functionname (arrayname[index], arrayname[index], ...)

For example:

```
#include <stdio.h>
#include <conio.h>
void display(int, int);
int main()
{
  int age = {1, 2, 3, 4};
  display(age[1], age[2]);
  getch();
  return 0;
}
void display(int x, int y)
{
  printf("%d %d", x, y);
  return 0; return void; }
```

Output:
2 3

WAP to add second and fourth element of array by using function where element are given by user.

Ans
```
#include <stdio.h>
#include <conio.h>
void sum(int,int);
int main()
{
   int a[5],i;
   for(i=0;i<5;i++)
    {
       printf("Enter a number: ");
       scanf("%d",&a[i]);
    }
   sum(a[1],a[3]);
  getch();
  return 0;
}
void sum(int x,int y)
{
    int sum1=x+y;
    printf("The sum is: %d ",sum1);
    return void;
}
```

Passing Entire Array to a Function.

To pass an entire array to a function, only the name of the array(1D) is passed as an argument.

Syntax for function-call

functionname(arrayname);

Syntax for function definition

return_type de functionname(data-type arrayname[])

Write a program to sum all the elements of array by using function.

Ans:
```
#include <stdio.h>
#include <conio.h>
void sum(int a[]);
int main()
{
   int a[10],i;
   for(i=0;i<10;i++)
   {
     printf("Enter a number : ");
     scanf("%d",&a[i]);
   }
   sum(a);
   getch();
   return 0;
}
void sum(int a[])
{
   int sum=0,i;
   for(i=0;i<10;i++)
```

```
{
  sum+=a[i];
}
printf("The sum is: %d", sum);
return void;
}
```

Write a program to read numbers and reorders them in ascending order by using function.

Ans
```
#include <stdio.h>
#include <conio.h>
void sort (int b[], int n)
int main()
{
   int n, a[100];
   int i;
   printf("Enter the no.of numbers: ");
   scanf("%d", &n);
   for(i=0; i<n; i++)
    {
       printf("Enter a number: ");
       scanf("%d", &a[i]);
    }
   sort(a, n);
   printf("The sorted list is:\n");
   for(i=0; i<n; i++)
     printf("%d ", a[i]);
   getch();
   return 0;
}
void sort (int b[], int n)
```

```
{
    int i, j, temp;
    for(i=1; i<n; i++)
    {
        for(j=0; j<n; j++)
        {
            if(b[i]<b[j])
            {
                temp=b[i];
                b[i]=b[j];
                b[j]=temp;
            }
        }
    }
}
```

Passing MultiDimensional Array to the Function.

To pass multidimensional array to the function is same as passing one-dimensional array to the function. But it is necessary to pass the non-leading dimensions to the function.

For example

```
void func(int a[][10]);
```

Function declaration to accept one dimensional string.

Syntax:

returnType functionName(char str[]);

For example:

void displaystring(char str[]);

Function declaration to accept two dimensional string.

In order to accept two dimensional string array the function declaration will look like the following.

returnType functionName(char [][N], int M)

For example:

void displayCities(char str[][50], int M)

Displaying list of strings using function pass.

```
#include <stdio.h>
#include <conio.h>
void display(char city[][50], int n)
int main()
{
   char str[][50]={"Manu","Gaurav","Saurav",
                   "Soni","Menka"}

   void display(city,5)
   getch();
   return 0;
}
disp void display(char city[][50], int n)
{
   int i,r;
```

```
for(r=0;r<5;r++)
{
    i=0;
    while(city[r][i]!='\0')
    {
      printf("%c",city[r][i]);
      i++;
    }
  printf("\n");
}
}
```

Write a program using function that displays the largest and smallest number from an array of number that is passed to the function.

Ans

```
#include <stdio.h>
#include <conio.h>
void smallandlarge(int b[], int n);
int main()
{
   int i, a[100], num;
   printf("Enter the no.of numbers: ");
   scanf("%d",&num);
   for(i=0;i<num;i++)
   {
     printf("Enter a number: ");
     scanf("%d",&a[i]);
   }
   smallandlarge(a,num);
   getch(); return 0;}
```

```
void smallandlarge(int b[], int n)
{
 int large=b[0], small=b[0]; int i;
 for(i=0; i<n; i++)
 {
   if (b[i]<small)
      small=b[i];
   if (b[i]>large)
      large=b[i];
 }
 printf("The largest number is: %d", large);
 printf("\n The smallest number is: %d", small);
}
```

2. Write a program Armstrong number between the range and display them in main function.

As
```
#include <stdio.h>
#include <conio.h>
int main()
{
  int a,b,n,s,m,r,j,i;
  printf("Enter the range: ");
  scanf("%d-%d",&a,&b);
  for(i=a;i<=b;i++)
  {
    n=i; m=0
    while(n!=0)
    { n=n/10; m++; }
    n=i; s=0;
    while(n!=0)
    { r=n%10;
      p=1
      for(j=1;j<=m;i++)
      {
        p=p*r;
      }
     s+=p;
     n=n/10;
    }
    if(s==i)
     printf("%d ",i);
  }
  getch(); return 0;
}
```

Date:
Page

Structures and Unions.

Arrays allow to define type of variable that can hold several data items of the same kind. Similarly, structure is another user defined type available in C that allows to combine variables (data items) of different kinds under a single name. The variables are called members of the structure. Structures are used to represent a record. suppose we want to keep track of books in a library. We might want to track the following attributes about each book:-

→ Book ID
→ Title
→ Author
→ Subject

Defining a structure

→ To define or create a structure, we use Struct keyword.
→ The struct keyword defines a new data type, with more than one member.
→ A structure definition consists of two parts:
- Structure declaration
- Structure Body

→ The

Syntax:

```
Struct structure-name
{
    data-type member_variable1;
    data-type member-variable2;
    data-type member-variable3;
    . . . . . .
    . . . . . .
    . . . . . .
};
```

For example:

```
Struct Books
{
    int book_id;
    char book-author[50];
};
```

Declaration of the structure

Structure variables can be declared in one of the two ways:

1. By using the struct keyword in the main() method.
2. By declaring a variable when the structure is defined.

By using the struct keyword in the main() keyword:

→ To declare the structure variable by struct keyword.

→ It should be declared within the main function.

→ This way of creating structure variable is preferred when multiple variables are required to be declared
→ The structure variables are declared outside the structure.

Syntax:
```
struct structName
{
    // structure definition
    data_type1  member_name1;
    data_type2  member_name2;
    data_type3  member_name3;
    . . . . . .
    . . . . . .
};
struct structName struct_var1, struct_var2;
```

For example:
```
struct bookstore
{
    char storeName[20];
    int totalBooks;
    char storeLicense[20];
};
int main()
{
    struct bookstore storeA, storeB;
}
```

Write a program using function that returns the largest and smallest number from an array of number that is passed to the function.

Ans
```
#include <stdio.h>
#include <conio.h>
int small(int [], int);
int large(int [], int);
void main()
{
   int a[100], n, i, s, l
   printf("Enter the no. of numbers: ");
   scanf("%d", &n);
   for(i=0; i<n; i++)
   {
      printf("Enter the number: ");
      scanf("%d", &a[i]);
   }
      s = small(a, n);
      l = large(a, n);
   printf("The smallest number is: %d", s);
   printf("The largest number is: %d", l);
   getch();
}
int small(int b[], int n)
{
   int s=b[0], i;
   for(i=0; i<n; i++)
   {
      if(b[i]<s)
         s = b[i];
```

```
    }
    return s;
}
int large(int b[], int n)
{
    int l = s[0], i;
    for(i=0; i<n; i++)
    {
        if (b[i]>l)
            l=b[i];
    }
    return l;
}
```

Write a program that displays Armstrong number between a specified range by calling function.

Ans
```
#include <stdio.h>
#include <conio.h>
int Arm(int);
int main()
{
    int a,b,i;
    printf("Enter the range: ");
    scanf("%d-%d",&a,&b);
    for(i=a;i<=b;i++)
    {
      if(Arm(i)==1)
        printf("%d\t",i);
    }
    getch();
    return 0;
}
int Arm(int n)
{
  int s=0;num=n;r,j;
int nn=0;
  while(num!=0)
  {
    nn++;
     num=num/10;
  }
  num=n;
  while(num!=0)
  {
     r=num%10;
```

```
p=1;
for(j=1;j<=r;j++)
    p=p*r;
 s+=p;
 num=num/10;
}
if(n==s)
 return 1;
else
    return 0;
}
```

Access structure variable

There are two ways to access structure variable.

a) By using . (dot) operator
b) By using → (this pointer) operator.

a) By using . (dot) operation

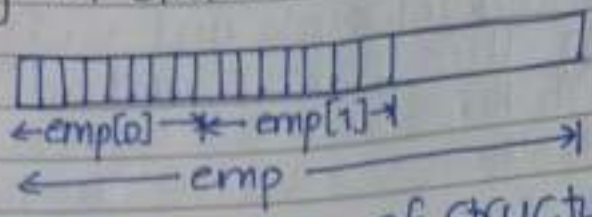
```
/* Program to display age and name of
the employee using structure. */
#include <stdio.h>
#include <conio.h>
struct employee
{
    int age;
    char name[100];
};
int main()
{
  struct employee e1;
  printf("Enter the age: ");
  scanf("%d", &e1.age);
  printf("Enter the name: ");
  gets(e1.name);
  printf("The details of the employee
          are: ");
  printf("\nAge: %d\nName: %s", e1.age,
                              e1.name);
  getch();
  return 0;
}
```

b) By

b) By this → (this pointer) operator
/* Program to store age and weight of a person using this pointer and display it */
```
#include <stdio.h>
#include <conio.h>
struct person
{
  int age;
  float weight;
};
int main()
{
  struct person *ptr, per1;
  ptr = &per1;
  printf("Enter the age: ");
  printf("%d", &ptr->age);
  scanf" printf("Enter the weight: ");
  scanf("%f", &ptr->weight);
  printf("The details of the person is:\n");
  printf("Age: %d\nWeight: %.2f", ptr->age,
                                  ptr->weight);

  getch();
  return 0;
}
```

Array of structure:



To declare array of structure we need to define/create a structure.

```
struct employee
{
   int empno;
   char name[10];
   float salary;
};
```

Now, we have to declare array of structure variable.

```
struct employee emp[10];
```

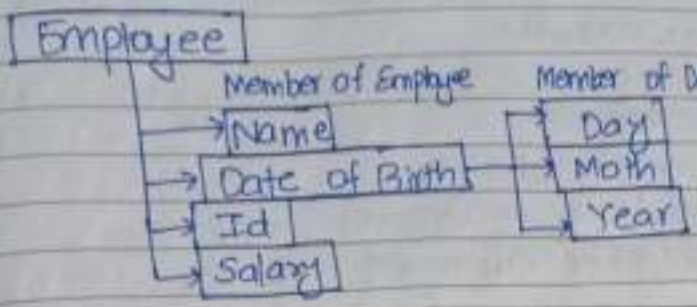
To access the structure variable

```
scanf("%d", &emp[i].empno);
```

Structure as a member of structure (Nested Structure)

C provides us the feature of nesting one structure by using which, complex data types are created. The outer structure is known as nesting structure and the inner structure is known as nested structure.

Let us take an example of it:



Syntax of nested structure.

```
Struct structure_name1
{
  data_type member_variable1;
  data_type member_variable2;
  .
  .
  .
  data_type member_variableN;
  Struct structure_name2
  {
    data_type member_variable1;
    data_type member_variable2;
    .
    .
    .
    data_type member_variable n;
  } structure_variable2;
} structure_variable1;
```

Accessing members of nested structure.

Syntax:

```
//accessing member of outer structure
structure_variable1.member_of_outer_structre;
```

// accessing member of inner structure.
structure_variable1. structure-variable2. member-of-inner variable.

For example:

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
struct employee
{
    int id;
    char name[20];
    struct Date
    {
        int dd;
        int mm;
        int yyyy;
    } dob;
} emp;
int main()
{
    //storing employee information
    emp.id = 101;
    strcpy(emp.name, "Manisha Lama");
    emp.dob.dd = 10;
    emp.dob.mm = 11;
    emp.dob.yyyy = 2014;
//printing first employee information
    printf("employee id: %d\n", emp.id);
    printf("employee name: %s\n", emp.name);
    printf("employee date of joining (dd/mm/yyyy):
            %d/%d/%d\n", emp.dob.dd, emp.dob.mm,
                emp.dob.yyyy);
```

→ outer structure.

→ Inner structure.

→ Inner structure variable

→ outer structure variable

```
getch();
return 0;
}
```

Function and structure

Ways

- Passing structure members to a function.
- Passing an entire structure to a function.
- Passing structure pointer to a function.
- Passing an array of structure to a function.

Passing structure member to a function

```
#include <stdio.h>
#include <conio.h>
struct employee
{
   int Id;
   char name[20];
   char designation[30];
   float salary;
}; void display(int, char [], char [], float)
int main()
{
  struct employee emp={234,"Donald Trump",
                       "Manager", 100000};
  display(emp.Id, emp.name, emp.designation,
          emp.salary);
  getch(); return 0;
}
```

```
void display(int Id, char name[20], char
             designation[30], float salary)
{
  printf("Details of Employee:\n");
  printf("Id: %d\n", Id);
  printf("Name: %s\n", name);
  printf("Designation: %s\n", designation);
  printf("salary: %f\n", salary);
}
```

Passing whole structure to a function.

→ It is also possible to pass an entire structure to a function as its argument.

→ An entire structure includes all the members of the structure.

```
#include <stdio.h>
#include <conio.h>
struct employee
{
    int Id;
    char name[20];
    char designation[30];
    float salary;
};
void display (struct employee);
int main()
{
  struct employee emp={234,"Donald Trump",
                       "Manager", 100000}
 display(emp);
 getch();
```

```
return 0;
}
void display (struct employee e)
{
printf("Employee details:\n");
printf("Id: %d\n", e.Id);
printf("Name: %s\n", e.name);
printf("Designation: %s\n", e.designation);
printf("Salary: %f\n", e.salary);
}
```

Passing an array of structure to a function.

```
#include <stdio.h>
#include <conio.h>
#define size 3
struct employee
{
    int Id;
    char name[20];
    char designation[30];
    float salary;
};
void display (struct employee e1[]);
void read (struct employee e[]);
int main()
{
  struct employee emp[size];
  read(emp);
  display(emp);
  getch();
  return 0;
```

```
}
void read(struct employee e[])
{
    int i;
    printf("Storing data of employees:\n");
    for(i=0;i<size;i++)
    {
      printf("Enter Id: ");
      scanf("%d",&e[i].Id);
      printf("Enter name: ");
      scanf("%s",e[i].name);
      printf("Enter designation: ");
      scanf("%s",e[i].designation);
      printf("Enter salary: ");
      scanf("%f",&e[i].salary);
    }
}
void display(struct employee e1[])
{
    int i;
    printf("Displaying data:\n");
    for(i=0;i<size;i++)
    {
      printf("Id: %d",e1[i].Id);
      printf("\nName: %s\n",e1[i].name);
      printf("Designation: %s\n",e1[i].designation);
      printf("Salary: %f\n",e1[i].salary);
    }
}
```

WAP to create structure named student that has structure member Roll_number, name and marks. Display the name of student whose marks is distinction by using function and array of structure.

```
#include <stdio.h>
#include <conio.h>
struct student
{
   int Roll_number;
   char name[10];
   float marks;
};
void read(struct student [], int);
void display(struct student [], int );
int main()
{
  int n;
  struct student s[50];
  printf("Enter no.of students: ");
  scanf("%d", &n);
  read(s, n);
  display(s, n);
  getch();
  return 0;
}
void read(struct student s1[], int n)
{
  int i;
  printf("storing data of students:\n");
  for(i=0; i<n; i++)
```

```
{
    printf("Enter Roll_Number: ");
    scanf("%d", s1[i].Roll_Number);
    printf("Enter Name: ");
    scanf("%s", s1[i].name);
    printf("Enter marks: ");
    scanf("%f", s1[i].marks);
}
}
void display(struct student s2[], int n)
{
    int i;
    for(i=0;i<n;i++)
    {
      if(s2[i].marks >= 80)
        printf("\n%s", s2[i].name);
    }
}
```

Passing structure pointer to a function.

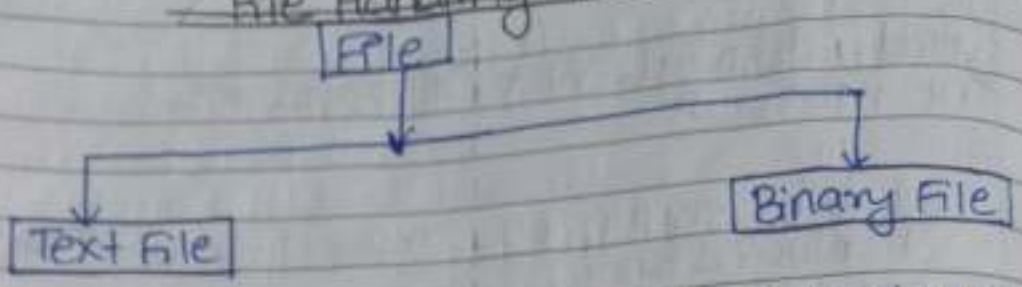
→ A structure pointer can be passed to a function.

→ When a pointer to a structure is passed as an argument to a function, the function is called by reference.

```
#include <stdio.h>
#include <conio.h>
struct employee
{
    int emp_id;
    char name[20];
```

```
    float sal;
};
void display(struct employee *e);
int main()
{
    struct employee emp={1,"Anu",1000};
    display(&emp);
    getch();
    return 0;
}
void display(struct employee *e)
{
    printf("Id: %d",e->emp_id);
    printf("Name: %s",e->name);
    e->sal+=1000;
    printf("Salary: %d",e->sal);
}
```

File Handling in C



- File extension is .txt
- Data are stored in plain text format

→ File extension is .bin
→ Data are stored in 0s and 1s.

File opening modes

Mode	Meaning of Mode	During Inexistence of file?
r	open for reading	If the file does not exist, fopen() returns NULL.
w	open for writing	If the file exists, its contents are overwritten If the file does not exist, will be created
a	Open for append (Data is added to the end of the file)	If the file does not exist, it will be created.
r+	Open for both reading and writing	If the file does not exist, fopen() returns NULL.

Mode	Meaning of Mode	During Inexistence of file
w+	open for both reading and writing	If the file exists, its content are overwritten. If the file doesnot exist, it will be created

If the suffix b is present in the mode like (rb, wb, ab, rb+, wb+, ab+), then same operation will be performed like above but it will perform operation on binary file.

Mode	Meaning of Mode	During Inexistence of file
a+	open for both reading and appending	If the file doesnot exist, it will be created.

Defining, opening and closing a file.

Opening a file

A program must open a file before to perform reading and writing to operation on the file. When working with files, we need to declare a pointer of type file as: FILE *fptr; Here, FILE is a special structure, declared in header file stdio.h.

A file is opened using the fopen() function defined in the stdio.h header file.

Syntax:

fptr = fopen ("filename", "file_mode");

Example:

```
fptr = fopen("E:\\cprogram\\newprogram.txt", "w");
fptr = fopen("E:\\cprogram\\newprogram.bin", "wb");
```

Closing a file

Closing a file is performed using the fclose function:

Syntax:

```
fclose(fptr);
```

Example:

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

File operations:

In C, we can perform FOUR major operations on files, either text or binary.

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

Library function for Reading/writing from/to a file:

1. String input/output
2. Character input/output
3. Formatted input/output
4. Record input/output.

Character input/output

a. fgetc():
- Used to read a single character from a file.
- Syntax:
char-variable = fgetc(file-ptr_variable);
Here, a character is read from a file representing file-ptr-variable.

b. fputc():
- Used to write a single character to a file.
- Syntax:
fputc(char-variable, file-ptr_variable);
Here, a character is written to a file represented by the file-ptr-variable.

WAP that creates a text file and write some lines of text into it until the user presses an enter key.

Ans:
```
#include <stdio.h>
#include <conio.h>
int main()
{
FILE *fp; char ch;
fp = fopen("data.txt", "w");
```

```
While((ch=getchar())!='\n'){
    fputc(ch,fptr);
}
fclose(fp); getch();
return 0;
}
```

WAP that read the content of the file previously written and display it in the console

Ans
```
#include <stdio.h>
#include <conio.h>
int main()
{
    FILE *fp; char ch; fp=fopen("data.txt","r")
    while((ch=fgetc(fp))!=EOF)
    {
        putchar(ch);
    }
    fclose(fp);
    getch();
    return 0;
}
```

WAP that copies one file to another file.

Ans
```
#include <stdio.h>
#include <conio.h>
int main()
{
  FILE *fp1,*fp2;
  char ch1;
```

```
fp1= fopen("read.txt","r");
fp2 =fopen("write.txt","w");
while((ch1=fgetc(fp1))!= EOF)
{
    fputc(ch1,fp2);
}
fclose(fp1);
fclose(fp2);
getch();
return 0;
}
```

WAP to create a file named "test.txt" and write "Welcome to my college" to this file.

Ans
```
#include <stdio.h>
#include <conio.h>
int main()
{
  FILE *fp;
  char str[]="Welcome to my college"
  fp=fopen("file.txt","w");
  fputs(str,fp);
  fclose(fp);
  getch();
  return 0;
}
```

String I/O (fgets(), fputs())

a. fgets()
- Used to read string from a file.
- Syntax:
  fgets (string-variable, int-value, file-ptr-variable)

b. fputs()
- Used to write string to a file
- Syntax:
  fputs (string-variable, file-ptr-variable)

WAP that reads a string from the file

Ans
```
#include <stdio.h>
#include <conio.h>
int main()
{
  FILE *fp;
  char str[100];
  fp=fopen("file.txt","r");
  fgets(str, 100, fp);
  printf("%s\n", str);
  fclose(fp);
  getch();
  return 0;
}
```

Formatted Input/Output.

a. fscanf():
- Formatted input function, used to read a block data from a file.
- Syntax:
fscanf (file-ptr-variable, "control string", &list-variables);

b. fprintf():
- Formatted output function, used to write a block data to a file.
- Syntax:
fprintf (file-ptr-variable, "control-string", list-variables);

WAP that asks the records of the students until the user wants and write them into a file and display the contents of the file in the console

Ans
```
#include <stdio.h>
#include <conio.h>
int main()
{
  FILE *fp;
  char name[10], address[10], yes;
  int roll; fp=fopen("file1.txt", "w+");
  do
  {
    printf("Enter name: ");
    scanf("%s", name);
    printf("Enter address: ");
```

```
scanf("%s",address);
printf("Enter roll: ");
scanf("%d",&roll);
fprintf(fp,"%s %s %d\n",name,address,
                        roll);

printf("Y for continue: ");
scanf("%c",&res);
} while(res=='Y' || res=='y');
rewind(fp);
printf("contents:\n");
while(!=EOF)
{
  fscanf(fp,"%s %s %d\n",name,address,&roll);
  printf("%s %s %d\n",name,address,roll);
}
fclose(fp);
getch();
return 0;
}
```