Unit-2:-

1) Operator Precedence determines the order in which operators are evaluated in expressions.

① (Highest) → [], (), ->, (Function call, array subscript)
② !, ~, ++, --, +, -, *, & → (Unary operators)
③ *, /, % → (Multiplicative)
④ +, - → (Additive)
⑤ <<, >> → (Bitwise shift) ⑥ <, <=, >, >= → (Relational)
⑦ & → (And) ⑧ ==, != → (Equality) ⑨ && → (logical AND)

2) Postfix :- The value is used first. (a++, a--)

Ex :- int a = 5;
int b = a++;

Result,
a=6, b=5

Prefix :- The value is incremented/decremented first,

Ex :- int a = 5;
int b = ++a;

Result :-
a=6, b=6,

3) ASCII values for Uppercase (A to z) → 65 to 90
" " " lower " (a to z) → 97 to 122

4) Explicit :- The programmer manually converts a variable from one type to another using a cast operator.

Ex :-
```
#include <stdio.h>
int main() {
    int a = 5, b = 2;
    float result = (float)a / b
    printf("Result: %.2f\n", result);
    return 0;
}
```

Dry run :-
a = 5, b = 2. int
float a = 5.0 . float
result = 2.5 float

Output :
Result: 2.50

Implicit :- C automatically Converts one data type to another when types are mixed in an expression.

Ex:
```
#include<stdio.h>
int main(){
    int a=5;
    float b=2.5;
    float c=a+b;
    Printf("Result : %f\n", c);
    return 0;
}
```

Dry run :-

a = 5 int
b = 2.5 float
result = 7.5 "

output :
Result : 7.5

5) Bitwise operators :-

" " Perform operation on the binary bits of integers. These are used for low level programming like Cryptography, Performance ~~optimization~~ optimization.

Types → &, |, ^, ~, <<, >>,

Ex:-
```
#include <stdio.h>
int main(){
    int a=5, b=3;
    int result = a&b;
    Printf("Result : %d\n", result);
    return 0;
}
```

Dry Run :-

	Binary
a = 5	0101
b = 3	0011
a & b	0001 = 1

output :
Result : 1

Unit 3 :- loops :-

A loop in C is used to execute a block of code repeatedly as long as a condition is true.

Types :- For loop, while loop, do – while loop
(i) (ii) (iii)

(i) Ex :
```
#include <stdio.h>
int main() {
  int i;
  for(int i=1 ; i<=5 ; i++)
    Printf ("%d", i);
  return 0;
}
```

Dry run :-

i = 1, 2, 3, 4, 5 i<=5 → Condition is true.

i = 6 i<=5 → Condition false.

loop ends

output :

1 2 3 4 5

(ii)
```
#include<stdio.h>
int main(){
  int i=1;
  while (i<=3)
    Printf ("%d", i++);
  return 0;
}
```

Dry run :-

i = 1, 2, 3 i<=3 → True

i = 4 → False.

loop ends

output :

1 2 3

(iii)
```
#include<stdio.h>
int main(){
  int i=1;
  do
    Printf("%d", i++);
  while (i<=3);
  return 0;
}
```

Dry run :-

i = 1, 2, 3 → True.

i++ = 2, 3, → True.

Out put :

1, 2, 3

Switch :- Switch is used to execute one block of code among many options, based on the value of an expression.

This is alternative to multiple if-else statements.

Ex:-
```
#include<stdio.h>
int main(){
    int day=2;
    Switch (day)
    Case 1;
    Printf ("Monday\n");
    break;
    Case 2;
    Printf ("Tuesday\n");
    break;
    Case 3;
    Printf ("wednesday\n");
    break;
    default;
    Printf ("Invalid day\n");
    return 0;
}
```

Dry run :-

day = 2.

Control enters Switch (day)

Case 1 : false → skip

Case 2 : true → execute then break.

output :

Tuesday

Ternary :- The ternary operator is a short hand way of writing an if-else condition.

if true → if true.

if false → if false.

Ex:
```
# include<stdio.h>
int main(){
int a=7, b=5;
int min =(a<b) ? a : b;
Printf ("%.d\n", min);
return 0;
}.
```

Dry run :-

a = 7

b = 5.

a<b → 7<5 → false

result b = 5.

min = 5

output :-

5.

2) Continue :- This statement skips the remaining code inside the loop for the current iteration and next iteration.

```
Ex:- #include<stdio.h>
     int main(){
       for(int i=1;i<=5;i++){
         if(i==3)
         continue;
         printf("%d ",i);
       }
       return 0;
     }
```

output : 1 2 4 5 .

Goto :- This statement jumps to a labeled part of the Progra

```
Ex:- #include<stdio.h>
     int main(){
       int x=1;
       if(x==1)
         goto skip;
       printf("This will be skipped \n");
     skip;
       printf("Jumped to here\n");
       return 0;
     }
```

output :
Jumped to here .

break :- This statement exist a loop or switch statement immediately, skipping any remaining iterations.

```
Ex: #include<stdio.h>
    int main(){
      for(int i=1; i<=5;i++){
        if(i==3)
          break;
        printf("%d\n",i);
      }
      return 0;
    }
```

output :
1
2

Difference :-

Continue	Goto	Break
1) Skip certain loop values	Stop loop when a Condition is met	Jumping over code or early exit
2) Structured and safe	Unstructured.	Structured and Safe
3) Goes to next iteration	Jumps to a specific label	Exits the loop or switch completely
4) loops only	Anywhere in the Program.	loops and switch
5) Skips rest of current loop iteration.	Exits loop or switch statement	Jumps to a labeled statement

3) Conditional operators :-

The Conditional operators is also known as the ternary operators (? :)

It provides a short-hand way to write an if-else Condition.

If the Condition is true, it returns value if true.

If the " " false, " " " if false

Ex:
```
#include <stdio.h>
int main() {
    int a = 10, b = 20;
    int max = (a > b) ? a : b;
    Printf("%d\n", max);
    return 0;
}
```

output :

20

↳ Fall through :- In C Programming fall through refers to the behavior in a switch statement.

Ex:- (no breaks is here.)

```
#include <stdio.h>
int main () {
  int num = 2;
  switch (num) {
  case 1;
    Printf ("Case 1\n");
  case 2;
    Printf ("Case 2\n");
  case 3;
    Printf ("Case 3\n");
    break;
  default:
    Printf ("Default Case\n");
  }
  return 0;
}
```

- Purpose of break in switch :—
  * Prevents fall through.
  * Exits the switch block after executing the matching case.

Ex:

```
#include <stdio.h>
int main () {
  int day = 3;
  switch (day) {
  case 1:
    Printf ("Monday\n");
  break;
  case 2:
    Printf ("Tuesday\n");
  break;
  case 3:
    Printf ("Wednesday\n");
  break;
  default:
    Printf ("Invalid day\n");
  }
  return 0;
}
```

• Default Key words :-

In a switch statement is used to define a block of code that runs when none of the case values match the s.c.

Purpose :-

* Acts like the 'else' in an if-else chain.
* Executes only when no case matches.

Ex:-

```
#include<stdio.h>
int main(){
    int choice = 5;
    switch (choice) {
        case 1:
            Printf("you chose 1\n");
            break;
        case 2:
            Printf("you chose 2\n");
            break;
        default;
            Printf("Default case\n");
    }.
    return 0;
}.
```

Output:

Default case.

Key Points :-

1) Default is optional in a switch block.
2) If it has no break, it may also fall through.

5) Ex:- Infinite loop → loops is never

while(1),

```
#include<stdio.h>
int main(){
while(1){
printf("This will print forever...\n");
}
return 0;
}
```

for,

```
#include<stdio.h>
int main(){
for(;;){
printf("Infinite for loop\n");
}return 0;
}
```

Do-while,

```
#include<stdio.h>
int main(){
do{
printf("Infinite loop\n");
}
while(1);
return 0;
}
```

Ex:- Null loop :- No operation inside its body.

```
#include<stdio.h>
int main(){
int i;
for(i=0; i<=100000; i++);
printf("Loop ended\n");
return 0;
}
```

```
#include<stdio.h>
int main(){
int flag=0;
while(flag==0);
printf("Flag is set!\n");
return 0;
}
```

Break	exit(0).
loop or switch only	Entire program
Header is not needed.	stdlib.h is Present
To next statement after loop.	Ends Program Completely.
Exiting loop	Ending program early

Ex:-
```
#include<stdio.h>
int main(){
  for(int i=0; i<=5; i++){
    if(i==3)
      break;
    Printf("%d", i);
  }
  Printf("Loop ended\n");
  return 0;
}
```
Output:
0 1 2 loop ended.

```
#include<stdio.h>
#include<stdlib.h>
  int main(){
    int age=17;
    if(age<18){
      Printf("Not eligible\n");
      exit(0);
    }
    Printf("Eligible\n");
  return 0;
}
```
output:
Not eligible.

7) Nested loop switch break :-

```
#include<stdio.h>
int main(){
  for(int i=1; i<=3; i++){
    for(int j=1; j<=3; j++){
      if(j==2)
        break;
      Printf("i=%d, j=%d\n", i, j);
    }
  }
  return 0;
}
```

output :-
i=1 j=1
i=2, j=1
i=3, j=1

use a flag.

```
#include<stdio.h>
    int main(){
        int done = 0;
for(int i=0; i<=3 && !done; i++){
for(int j=0; j<=3; j++){
  if(i+j==2){
      done = 1
    break;
}
printf("i=%d, j=%d\n", i, j);
}
}
```

use goto,

```
for(int i=0; i<3; i++){
 for(int j=0; j<3; j++){
    if(i+j==2)
     goto end;
   Printf("i=%d, j=%d\n", i, j);
 }
}
end:
```

Unit-4:— Size of Array :—

```
#include <stdio.h>
    int main(){
     int arr[5] = {1, 2, 3, 4, 5};
Printf("Total size of bytes: %lu\n", sizeof(arr));
"   ("Size of one element: %lu\n" " "(arr[0]));
"   ("Number of elements: %lu\n", sizeof(arr));
return 0;
}
```

output :-

20
4
5

realme 12 Pro+ 5G

⇒ Array :— Array are used to store multiple values under a single variable name.

```
#include <stdio.h>
    int main(){
int arr[5] = {10,20,30,40,50};
printf(" Array elements are: \n");
 for (int i=0; i<5 ; i++){
   Printf("arr[%d] = %d\n",i, arr[i]);
}
 return 0;
}.
```

2D Array :—

```
#include<stdio.h>
  int main(){
   int matrix[3][2] = {
        {1,2},
        {3,4};
        {5,6},
   };
   Printf("The 2D array is :\n");
    for (int i=0; i<3 ; i++) {
      for(int j=0; j<2 ; j++) {
         Printf("%d ", matrix[i][j]);
    }
     Printf("\n");
  }
  return 0;
}.
```

output :

```
12
34
56
```

3D :-

```
#include<stdio.h>
int main() {
    int arr[2][3][2] = {
        { {1,2},
          {3,4},
          {5,6},
        },
        {
          {7,8},
          {9,10},
          {11,12},
        }
    };
    for(int i=0; i<2; i++) {
      for(int j=0; j<3; j++) {
        for(int k=0; k<2; k++) {
printf("arr[%d][%d][%d] = %d\n", i, j, k, arr[i][j][k]);
      }
    }
  }
  return 0;
}
```

output :

arr[0][0][0] = 1
" [0][0][1] = 2
" [0][1][0] = 3
. . .
arr[1][2][1] = 12.

3) Functions in <string.h>

Function	Description
strlen(str)	Returns the length of the string
strcpy(dest, src)	Copies string from src to dest.
strncpy(dest, src, n)	Copies n characters from src to dest.
memset(ptr, val, n)	Fills memory with a constant byte value
memcpy(dest, src, n)	Copies n bytes from src to dest
strrchr(str, ch)	Finds last occurrence of character ch.

Unit-5

1)

Linear Search	Binary Search.
Unsorted and sorted array.	only sorted array.
Sequential	Divide and Conquer.
Time Complexity is O(n)	O(log n)
Implementation Simple	Slightly more Complex
Speed on large data (slow)	fast.
Use: Unsorted data/small	Sorted data / Large.

Time Complexity	Explanation
Best O(1)	Target found at the first element.
Average O(n/2) → O(n)	Target found around the middle.
Worst O(n)	Target is the last element or not present at all

Best O(1) → Target the middle element in the first Check.

Average O($\log_2 n$) → Target is somewhere in the middle divide and conquer.

Worst O($\log_2 n$) → Target is at the end or not present - all divisions done.

Algorithm:-

Start → Set i=0 → Repeat (steps 4-5 while i<n) → if arr[i] == key, then (element found) → Return i (element found, Increment i) → (End of array is reached and element not found) → if (Element not found) → Return -1 → End

Start → Set low=0, high=n-1 → Repeat (while low<=high) → find mid = (low+high)/2, if arr[mid] == key return mid (element found) → if key < arr[mid], set high = mid-1 → else, set low = mid+1 → if low > high, return -1 (element not found) → End.

2) Bubble Sort	Selection Sort	Insertion Sort
Time (best) $O(n)$	$O(n^2)$	$O(n)$
Time (worst) $O(n^2)$	$O(n^2)$	$O(n^2)$
Space $O(1)$	$O(1)$	$O(1)$
Stable → yes	No	yes
Adaptive → yes	No	yes
Suitable for small data	Small data	Nearly sorted data.
Algorithm: Start ↓ Repeat: for i=0, to n-2: ↓ Repeat: for j=0, to n-i-2: if arr[j]>arr[j+1] then Swap arr[j] and arr[j+1] ↓ End loop5 ↓ End.	Start ↓ Repeat for i=0 to n-2: Set min index = i Repeat for j=i+1 to n-1 if arr[i]<arr[min index] then. Set min index = i After inner loop ends swap arr[i] with arr[min index] ↓ End. ↓ Stop	Start ↓ Repeat for i=1 to n-1; Set key = arr[i] Set j=i-1; while j>=0 and arr[j]> move arr[j] to arr[j+1] Decrement j by 1 Set arr[j+1]=key ↓ End ↓ Stop

3) Quadratic Equation	Roots.
An algebraic expression of the form $ax^2+bx+c=0$	The solutions that make the equation equal to zero
Represent a problem to solve.	Represent the answers / soluti to that problem.
Quantity is → One equation	Always two roots.
That is a full Mathematical Expressio	That is can be real or Complex numbers.
Used to find the unknown values.	Used to understand the behavior of the eqn.

Unit-6 :-

1) Call by value :- A copy of the variable is passed to the function.

Ex:-
```
#include<stdio.h>
#include<iostream>
using namespace std;
void modifyvalue(int a){
    a = a+10;
}
int main(){
    int x=5;
    modifyvalue(x);
    Cout<<"value of x after function call:"<<x<<endl
    return 0;
}
```
output:
value ---- : 5

Call by Reference :-

The address of the variable is passed.

Ex:-
```
#include<stdio.h>
#include<iostream>
using namespace std;
void modifyvalue(int&a){
    a=a+10;
}
int main(){
    intx=5;
    modifyvalue(x);
    Cout<<"Value of x after function call:"<<x<<endl
    return 0;
}
```
Output :-

---- : 15.

2) Functions in <math.h>

Functions	Description	Example	output
sqrt(x)	Square root of x	sqrt(16)	4
pow(x,y)	x raised to the power y	pow(2,3)	8
abs(x)	Absolute value of x (int only)	abs(-5)	5
fabs(x)	absolute value of x (float/double)	fabs(-5.5)	5.5
ceil(x)	Smallest integer $\geq x$	ceil(3.2)	4
floor(x)	Largest int $\leq x$	floor(3.8)	3
log10(x)	logarithm base 10	log10(100)	2.

<Stdio.h>

Function	Description	Example
Printf()	Outputs formatted text to the screen	Printf("Hello,%d",10);
Scanf()	Reads formatted input from the keyboard.	Scanf("%d",&x);
Putchar()	outputs a single character	Putchar('A');
fgets()	Reads a string from a file.	fgets(str,100,fp);
fputc()	writes a character to a file	fputc('A',fp);
fPrintf()	writes formatted output to a file	fprintf(fp,"%d",x);
fScanf()	Reads formatted input from a file	fscanf(fp,"%d",&x);

3) - Actual parameters :- This are the real values or variables that are passed to a function when it is called.

Ex: greet("Sanjoy");

"Sanjoy" is the actual parameter.

These values are assigned to the formal parameters.

Ex:- greet(char name[]), 'name' is the " ".

5) Return keyword :—

The " " is used in C to exit a function and send a value back to the place where the function was called.

```
Ex: int add(int a, int b){
      return a+b;
    }
    int main(){
      int result = add(5,3);
      Printf("Result = %d\n", result);
      return 0;
    }.
```

Unit - 7 :—

1) Forward Tracing :- Starts with known facts and applies inference rules to extract more data until a goal is reached.

Backward Tracing :— Starts with a goal and works backward to see if there are facts or rules that support that goal.

F.T Recursion Trace.

A (Given facet)
↓
B (From Rule 1 : A→B)
↓
C (From Rule 2 : B→C)
↓
D (From Rule 3 : C→D)

Recursion type → Tail Recursion / Linear Recursion

B.T Recursion Trace.

D (Goal)
↓
Needs C (Rule 3 : C→D)
↓
needs B (Rule 2 : B→C)
↓
needs A (Rule 1 : A→B)
↓
A is known (Fact)

R.T is → Backtracking Recursion.

2) Recursion	Iteration
A function call itself	Repeats a block of code using loops
Uses more memory (call stack)	uses less memory
Generally slow speed	Generally faster.
Needs a base case to stop	stops with a loop condition
Simpler for complex problem	Simple for linear tasks

⇒ Definition of Recursion :-

Recursion is a programming technique where a function calls itself directly or indirectly to solve a problem by breaking it down into smaller sub problems.

Advantages :-

(i) Simpler Code : Makes code easier to write and understand for problems like tree traversal.

(ii) Natural Fit for some problem :- Works well with recursive structures like trees & graphs.

Disadvantages :-

(i) Slower Execution :- Recursion can be slower than iteration.

(ii) More Memory Usage :- Each recursive call uses stack memory.

Uses of Recursion :-

(i) Mathematical : Factorial, Fibonacci series.

(ii) Data Structures : Tree, graph traversal.

(iii) Algorithms : Merge Sort, quick sort, binary search.

⇒ Algorithm to Find sum

Iterative

Step 1: Start
↓
Initialize sum ← 0
↓
Read n
↓
For i ← 1 to n, do
sum ← sum + i
↓
Print sum
↓
Stop

Recursive

Start
↓
Define function sum(n)
if n == 1, return 1
else return n + sum(n-1)
↓
Read n
↓
Call sum(n) and store the result.
↓
Print result
↓
Stop.

Factorial :-

Iterative

Start
↓
Read number n
↓
Initialize factorial ← 1
↓
for i from 1 to n do
factorial ← factorial * i;
↓
Print factorial
↓
Stop

Recursive

Start
↓
Define function factorial(n)
if n == 0 or n == 1
return 1.
Else
return n * factorial(n-1)
↓
Read number n
↓
Call fac'(n) and store the result
↓
Print result
↓
Stop.

Fibonacci :-

Iterative

Start
↓
Read n (Position for Fibonacci series)
↓
Initialize
a ← 0
b ← 1
↓
if n == 0, Print a and stop
" n == 1, " b " "
for i from 2 to n do
c ← a + b
a ← b
b ← c
↓
Print b
↓
Stop

Recursive

Start
↓
Define function Fibonacci(n)
if n == 0, return 0
if n == 1, " 1
else return Fibonacci(n-1) + Fibonacci(n-2)
↓
Read n
↓
Call Fibonacci(n) and store result
↓
Print result
↓
Stop.

Quick Sort :-

Start
↓
Choose a pivot element from the array
↓
Partition the array into two parts:
- Left part with elements <= Pivot
Right " " " > Pivot
↓
Recursively apply quick sort on left part.
↓
" " " " " Right "
↓
Stop when the sub-array has 0 or 1 element.

Merge Sort :-

Start

↓

if the array has one element, it is already sorted, so return.

↓

Divide the array into two halves

↓

Recursively apply merge sort on the left half.

↓

" " " " " " "

Right half

↓

Merge the two sorted halves into a single sorted array

↓

Stop.

5) Base Case and Recursive Case

```
#include <stdio.h>
    int factorial(int n){
    // Base case
      if(n==0||n==1)
        return 1;
    // Recursive case
      return n*factorial(n-1);
}
    int main(){
        int num;
    Printf("Enter a number:\n");
      Scanf("%d", &num);
    int result = factorial(num)
    Printf("Factorial of %d is %d\n", num, result)
      return 0;
}
```

Unit-8 :— Structure is a user-defined data type that groups related variables of different type under one name.

Declaration :-

```
#include <stdio.h>
// Struct student{  //Structure definition
    int id;
    Char name[50];
    float marks;
};
int main(){
Struct student student1; //Structure variable declaration.
//Assign values to members
student1.id = 101;
Strcpy(student1.name, "Me");
student1.marks = 85.5;
Printf("ID : %d\n", student1.id);
" ("Name : %s\n", " 1.name);
" ("Marks : %.2f\n", " 1.marks);
return 0;
}
```

Size :-

```
Struct Example{
  Char a; // 1 byte
  int b; // 2 "
  Char c; // 1 "
};
```

Total without Padding = 6 bytes

Array :— It allows you to store multiple records of the same kind.

```
Struct Student {
   int id;
   Char name[50];
   float marks;
};
int main(){
   Struct Student student[3];
student[0].id = 101;
Strcpy(students[0].name, "Alice");
students[0].marks = 85.5;
same as
same as
for(int i = 0; i < 3; i++){
Printf("student %d :\n", i+1);
" ("ID : %d\n", students[i].id);
" (Name : %s\n", st" [i].name);
" ("Marks : %.2f\n\n", " [i].marks);
}
return 0;
}
```

Union :- That is a user-defined data type like a structure but all members share the same memory location.

Declaration :-

```
#include<stdio.h>
#  "  <string.h>
union Data {  // union Definition
    int i;
    float f;
    char str[20];
};
int main() {
    union Data data;
    data.i = 10;
    Printf("data.i = %d\n", data.i);
    data.f = 220.5;
    Printf("data.f = %.2f\n", data.f);
    Strcpy(data.str, "Hello");
    Printf("data.str = %s\n", data.str);
    return 0;
}
```

Size :-

```
union example {
    int i;      // 4 bytes
    float f;    // "  "
    char str[20]; // 20 bytes
};
union size = 20 bytes [2.M]
```

2)

Array	Structure
Collection of elements of same type stored contiguously	Collection of different types grouped under one name.
All elements have the same data type	Members can have different data types
Accessed by index	Accessed by member names
Used to store list of similar data	Used to represent a record with multiple related data fields.
Size depends on number of elements and element size.	Size depends on the sum of member sizes.

Unit-9:—

1) Pointer is a variable that stores the memory address of another variable.

Ex:

```
#include<stdio.h>
int main(){
    int a=10;
    int *p; //p is a pointer to an integer.
    p=&a;
Printf("%d\n",a);
Printf("%p\n",&a);
Printf("%p\n",p);
Printf("%d\n",p);
return 0;
}.
```

2) Size :-

Size of int pointer : 8 bytes
" " float " : "
" " char " : "
" " double " : "

Total = 64 bytes.

3) Self referential structure is a structure that contains a pointer to a structure of the same type.

It is commonly used to create linked data structures. like linked lists, trees.

Ex:-

```
#include<stdio.h>
Struct Node{
    int data;
Struct Node *next;
};
int main(){
Struct Node n1,n2;
n1.data=10;
n2.  "  =20;
n1.next=&n2;
n2.  "  = Null;
Printf("n1 data: %d\n",n1.data);
"  ("n2 through n1: %d\n",n1.next->data);
return 0;
}.
```

Output:
n1 data: 10
n2 data through n1: 20

Passing a function Pointer

```
#include <stdio.h>
int add(int a, int b) {
    return a+b;
}
void operate(int x, int y, int (*func_ptr)(int, int)) {
    int result = func_ptr(x, y);
    Printf("%d\n", result);
}
int main() {
    operate(5, 3, add);
    return 0;
}
```