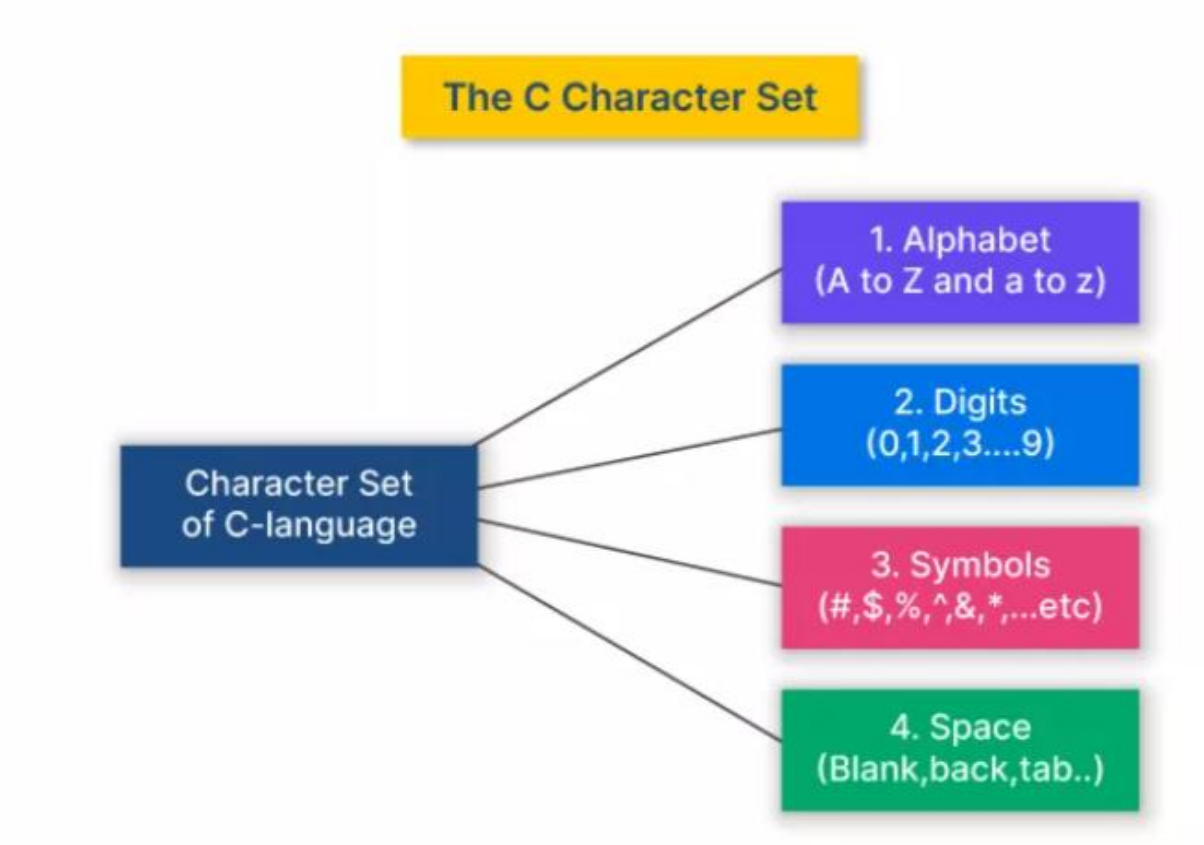
CSE 101:-- C programming notes

Unit-1

Basics and introduction to C : The C character set, Identifiers and keywords, Data types, Constants and variables, Expressions, Arithmetic operators, Unary, Relational, Logical, Assignment and conditional operators, Bitwise operators.

The C character set:--



Character Set in C

The character set in C consists of:

1. **Letters** – Includes uppercase (A-Z) and lowercase (a-z).
2. **Digits** – Numbers from 0 to 9.
3. **Special Symbols** – Includes characters like @, #, $, %, ^, &, *, (,), _, etc.
4. **White Spaces** – Space, tab (\t), newline (\n), and other formatting characters.
5. **Operators** – Includes arithmetic (+ - * / %), relational (== != < > <= >=), logical (&& || !), bitwise (& | ^ ~ << >>), assignment (= += -= *= /=), and ternary (?:) operators.

IDENTIFIER'S:-

Identifier's is a kind of varable's names, function-name and other user defined elements. They have also some set of rules.

1. Must start with a letter(upper-case or lower-case) or underscore(_) and cannot be a C keyword (for, while, int, float).
2. Cannot start with digit(0-9) or symbols or never include symbols in identifier's name.

3. Cannot include space (my variable is not a valid identifier my_variable is a valid identifier).
4. Include only under_score for connectivity naming of a variable between two words.

KEYWORDS:--

Keywords are reserved words and that have a predefined meaning .

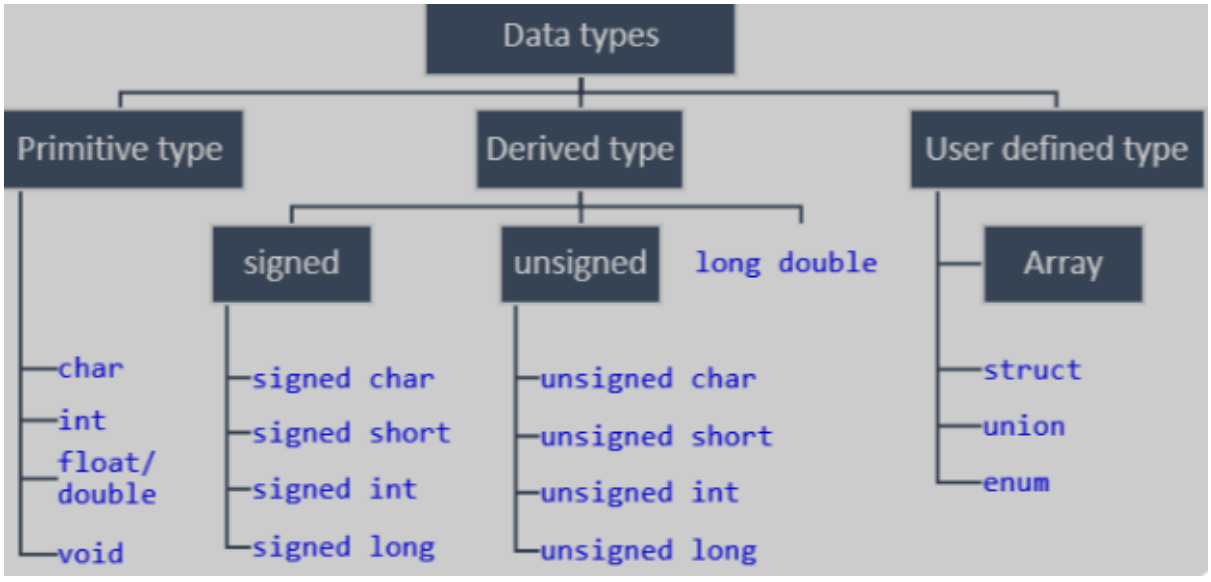
Keywords in C Language			
auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
continue	for	signed	void
do	if	static	while
default	goto	sizeof	volatile
const	float	short	unsigned

We cannot use these 32 keywords as an identifier’s

Categories of Keywords:

1. **Data Types** → int, float, char, double, void, _Bool, _Complex, _Imaginary
2. **Storage Classes** → auto, static, register, extern, typedef
3. **Control Flow** → if, else, switch, case, default
4. **Looping** → for, while, do, continue, break
5. **Memory Management** → const, volatile, restrict
6. **Functions & Return** → return, _Noreturn
7. **Others** → sizeof, goto, inline, _Alignas, _Atomic, _Thread_local, _Static_assert.

Data types:--



1. **Basic Types** → char, int, float, double, void

2. **Derived Types** → array, pointer, struct, union, enum
3. **Type Modifiers** → short, long, signed, unsigned

CONSTANTS & VARIABLES:--

A **constant** is a fixed value that **cannot be modified** once defined. We define constant using # define or const keyword.

```c
#define PI 3.14159
#define MAX 100
```

No = or ; used , #define used for macro constants.

```c
const int age = 18;
const float gravity = 9.8;
```

By using const the variable's value can't be change after once initialized using const

A **variable** is a named memory location that **stores values**, and its value **can be changed** during execution.

```
int num = 10;     // Integer variable
float rate = 5.5; // Float variable
char grade = 'A'; // Character variable
```

Naming rules of a variable's name is same as identifier's name.

Expressions and operator's:--

In C, an **expression** is a combination of variables, constants, and operators that evaluates to a value.

Ex:-- x + y = 10

5 + 5 = 10

Operators are symbols that perform operations on operands.

x + y = 10 here x , y, 10 are operands and +,= are operators.

Category	Operators	Example
Arithmetic	`+ - * / %`	`a + b`
Unary	`+ - ++ --`	`++a`
Relational	`== != > < >= <=`	`a > b`
Logical	`&&	
Assignment	`= += -= *= /= %=`	`a += 5`
Conditional	`? :`	`(a < b) ? a : b`
Bitwise	`&	`^ ~ << >>`

1. **Arithmetic Operators** – Used for mathematical operations like addition, subtraction, multiplication, and division.

- Example: +, -, *, /, %
- The **modulo (`%`) operator** is used to find the remainder after division.
- **Only for integer types (`int`, `long`, `short`, `char`).**
- Works with both **signed** and **unsigned** integers.

2. **Unary Operators** – Operators that operate on a **single operand**. Includes **increment (++)** and **decrement (--)**.

- Example: ++a(pre-increment), post-increment(a++), --b, -c

3. **Relational Operators** – Used to compare two values. These return either **true (1)** or **false (0)**.

- Example: ==, !=, >, <, >=, <=

4. **Logical Operators** – Used for **checking conditions** in expressions using **AND (&&), OR (||), and NOT (!)**.

- Example: (a > 5 && b < 10), !(x == 0)

5. **Assignment Operators** – Used to assign values to variables.

- Example: =, +=, -=, *=, /=, %=

6. **Conditional Operator (?:)**

Also known as the **ternary operator**, it is a shorthand for if-else statements.

```c
condition ? expression1 : expression2;
```

- If condition is **true** (nonzero), expression1 executes.
- If condition is **false** (zero), expression2 executes.

7.Bitwise Operators

Bitwise operators perform operations at the **binary level (bitwise operations)**.

How it operates: -- firstly if number is given in decimal form → converts it into binary form then form bitwise operations on these operands

Operator	Description	Example (A = 5, B = 3)	Result
`&` (AND)	Sets bit to 1 if both bits are 1	`A & B` (0101 & 0011)	0001 (1)
`	` (OR)	Sets bit to 1 if any bit is 1	`A
`^` (XOR)	Sets bit to 1 if bits are different	`A ^ B` (0101 ^ 0011)	0110 (6)
`~` (NOT)	Inverts bits (1→0, 0→1)	`~A` (~0101)	1010 (-6 in 2's complement)
`<<` (Left Shift)	Shifts bits left by n positions (multiplies by `2^n`)	`A << 1` (0101 << 1)	1010 (10)
`>>` (Right Shift)	Shifts bits right by n positions (divides by `2^n`)	`A >> 1` (0101 >> 1)	0010 (2)

Operator’s precedence in c:--

OPERATOR	TYPE	ASSOCIAVITY
( ) [ ] . ->		left-to-right
++ -- + - ! ~ (*type*) * & sizeof	Unary Operator	right-to-left
* / %	Arithmetic Operator	left-to-right
+ -	Arithmetic Operator	left-to-right
<< >>	Shift Operator	left-to-right
< <= > >=	Relational Operator	left-to-right
== !=	Relational Operator	left-to-right
&	Bitwise AND Operator	left-to-right
^	Bitwise EX-OR Operator	left-to-right
|	Bitwise OR Operator	left-to-right
&&	Logical AND Operator	left-to-right
||	Logical OR Operator	left-to-right
? :	Ternary Conditional Operator	right-to-left
= += -= *= /= %= &= ^= |= <<= >>=	Assignment Operator	right-to-left
,	Comma	left-to-right

Logical operators last me 0 ya 1 hi return krenge according to the truth table.

Logical and operators me if 1st condition hi false ho gyi 2nd condition pe move nhi krega.

Ex:-- *int a = -1, b = 4, c = 1, d*

d = ++a {&&} ++b {||} ++c;

----------Unit – 1(done)----------

Unit-2:--

Control structures and Input/Output functions : If, If else, Switch case statements, While, For, Do-while loops, Break and continue statements, Goto,Return, Type conversion and type modifiers, Designing structured programs in C, Formatted and unformatted Input/Output functions like printf(), Scanf(), Puts(), Gets() etc.

1. Conditional Statements

(a) If Statement

Executes a block of code **only if** the condition is true.

```
if (condition) {
    // Code to execute if condition is true
}
```

(b) If-Else Statement

Executes one block if the condition is true, otherwise executes another.

```
if (condition) {
    // Code if condition is true
} else {
    // Code if condition is false
}
```

Ex:-- which include if, else-if , else

```
int num = 0;
if (num > 0) {
    printf("Positive number\n");
} else if (num < 0) {
    printf("Negative number\n");
} else {
    printf("Zero\n");
}
```

(d) Switch Case

Used when multiple conditions depend on a single variable.

```
switch(expression) {
    case value1:
        // Code for value1
        break;
    case value2:
        // Code for value2
        break;
    default:
        // Code if no case matches
}
```

Here case 1: code is executed when 1 == expression (means case_value 1 == expression(inside bracket))

Case value is comparable with expression then the block of code of particular case is executable.

If not comparable then in this case default case will be executable automatically.

2. Loops in C

(a) While Loop

Executes **as long as** the condition is true and we don't know about how many times the loop iterate.

```
while (condition) {
    // Code to execute
}
```

```
int i = 1; // initialization
while (i <= 5) {  // condition
    printf("%d ", i);
    i++;  // increment / decrement
}
```

(b) For Loop

Used when the number of iterations is known.

```
for (initialization; condition; update) {
    // Code to execute
}
```

(c) Do-While Loop

Executes the loop **at least once**, even if the condition is false.

```c
do {
    // Code to execute
} while (condition);
```

```c
int i = 10;
do {
    printf("%d ", i);
    i++;
} while (i <= 5);
```

It prints –> 10

```c
int i = 10;
while (i <= 5) {
    printf("%d ", i);
    i++;
}
```

It prints nothing

3. Jump Statements

(a) Break

Exits the loop or switch case immediately.

```c
for (int i = 1; i <= 10; i++) {
    if (i == 5) {
        break;
    }
    printf("%d ", i);
}
// Output: 1 2 3 4
```

(b) Continue

Skips the current iteration and moves to the next.

```c
for (int i = 1; i <= 5; i++) {
    if (i == 3) {
        continue;
    }
    printf("%d ", i);
}
// Output: 1 2 4 5
```

(c) Goto

Transfers control to a labeled statement.

```c
int main() {
    int num;

    enter_number:
    printf("Enter a positive number: ");
    scanf("%d", &num);

    if (num < 0) {
        printf("Number is negative. Please enter again.\n");
        goto enter_number;
    }

    printf("You entered: %d\n", num);
    return 0;
}
```

(d) Return

Exits from a function and optionally returns a value.

```c
int add(int a, int b) {
    return a + b;
}
```

Type Conversion & Modifiers

Implicit Conversion (Type Promotion)

Happens automatically when a smaller type is converted to a larger type.

```c
int a = 10;
float b = a;  // a is automatically converted to float
```

Explicit Conversion (Type Casting)

Manually convert one type to another.

```c
float a = 5.5;
int b = (int)a;  // Converts 5.5 to 5
```

5. Input & Output Functions in C

Formatted I/O Functions

1. `printf()` → Displays output

```c
printf("Value of a: %d", a);
```

2. `scanf()` → Reads input

```c
scanf("%d", &a);
```

Unformatted I/O Functions

1. `puts()` → Prints a string with a newline

```c
puts("Hello");
```

2. `gets()` → Reads a string with spaces

```c
char name[50];
gets(name);
```

----------unit – 2----------

UNIT – 3:--

User defined functions and Storage classes : Function prototypes, Function definition, Function call including passing arguments by value and passing arguments by reference, Math library functions, Recursive functions, Scope rules (local and global scope), Storage classes in C namely auto, Extern, Register, Static storage classes.

1. Function Prototypes

A function prototype is a declaration of a function before its definition. It tells the compiler about the function's return type and parameters.

2. Function Definition

A function definition provides the actual implementation of the function.

3. Function Call

A function can be called from main() or other functions.

4. Passing Arguments

- **Pass by Value**: A copy of the argument is passed.

```c
#include <stdio.h>

int add(int, int); // Function prototype

int add(int a, int b) {
    return a + b; // Function definition
}

int main() {
    int sum = add(5, 3); // Function call
    printf("Sum: %d\n", sum);
    return 0;
}
```

Pass by Reference: The actual memory address is passed using pointers.

```c
void add(int, int, int*); // Function prototype

void add(int a, int b, int *result) {
    *result = a + b; // Function definition using pass by reference
}

int main() {
    int x = 5, y = 3, sum;
    add(x, y, &sum); // Function call with passing by reference
    printf("Sum: %d\n", sum);
    return 0;
}
```

5.Math Library Functions: C provides built-in math functions via the <math.h> library.

The math.h library in C provides many built-in functions, but the most commonly used ones are:

1. Trigonometric Functions

- **sin(x), cos(x), tan(x): Sine, Cosine, Tangent**
- **asin(x), acos(x), atan(x): Inverse Trigonometric Functions**
- **atan2(y, x): Arc tangent of two variables**

2. Exponential & Logarithmic Functions

- **exp(x): Exponential function (e^x)**
- **log(x): Natural logarithm (ln x)**
- **log10(x): Base-10 logarithm**

3. Power & Root Functions

- **pow(x, y): x raised to power y (x^y)**
- **sqrt(x): Square root of x**

4. Rounding & Remainder Functions

- **ceil(x): Round up to the nearest integer**
- **floor(x): Round down to the nearest integer**
- **fmod(x, y): Floating-point remainder of x/y**

5. Absolute & Comparison Functions

- **fabs(x): Absolute value of x**
- **fmax(x, y), fmin(x, y): Maximum & Minimum of two numbers**

6.Recursive function.

A function that calls itself to solve a problem.

```
int factorial(int n) {
    if (n == 0) return 1;
    return n * factorial(n - 1);
}
```

7.Scope rules(global and local):--

1. Local Scope (Local Variables)

- A **local variable** is declared **inside** a function or block.
- It is **accessible only within** the function or block where it is declared.
- It **cannot be used outside** its function.

- **Memory is allocated when the function is called and deallocated when the function ends.**

```c
#include <stdio.h>

void display() {
    int x = 10; // Local variable
    printf("Value of x: %d\n", x);
}

int main() {
    display();
    // printf("%d", x); // ❌ Error: x is not accessible here
    return 0;
}
```

2. Global Scope (Global Variables)

- A **global variable** is declared **outside** any function (usually at the top).
- It is **accessible in all functions** throughout the program.
- It **remains in memory** throughout the program execution.

```c
#include <stdio.h>

int x = 100; // Global variable

void display() {
    printf("Value of x in display(): %d\n", x);
}

int main() {
    printf("Value of x in main(): %d\n", x);
    display();
    return 0;
}
```

Feature	Local Variable	Global Variable
Declared in	Inside a function/block	Outside any function
Scope	Only within the function/block	Entire program
Memory Lifetime	Created when function is called, destroyed when it ends	Exists throughout the program execution
Default Value	Garbage value (if uninitialized)	0 (if uninitialized)
Accessibility	Cannot be accessed outside the function	Can be accessed by all functions

8. Storage Classes in C

In **C programming**, a **storage class** defines the **scope, lifetime, and visibility** of a variable.

1. Auto Storage Class (Default)

📌 **Auto variables are created when a function is called and destroyed when it ends.**

```c
#include <stdio.h>

void function() {
    auto int x = 10; // Auto variable (default storage class)
    printf("Value of x inside function: %d\n", x);
}

int main() {
    function(); // Calling function
    // x is not accessible here because it is local to function()
    return 0;
}
```

2.Register storage class:-- The **register** keyword in C is used to suggest that the variable be stored in the **CPU register** instead of RAM. This makes access **faster**, especially for frequently used variables.

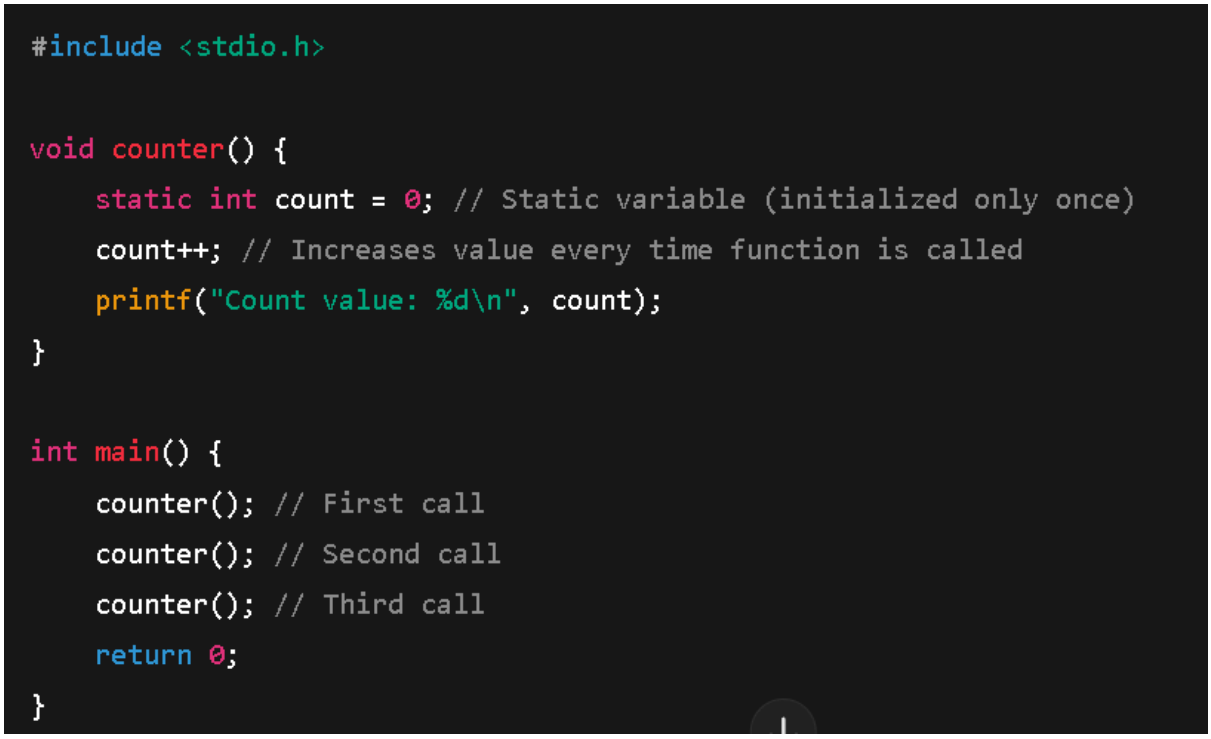
```c
#include <stdio.h>

void function() {
    register int x = 5; // Stored in CPU register (if possible)
    printf("Register variable x: %d\n", x);
}

int main() {
    function(); // Calling function
    return 0;
}
```

3. Static Storage Class

📌 Static variables retain their value between function calls.



```c
#include <stdio.h>

void counter() {
    static int count = 0; // Static variable (initialized only once)
    count++; // Increases value every time function is called
    printf("Count value: %d\n", count);
}

int main() {
    counter(); // First call
    counter(); // Second call
    counter(); // Third call
    return 0;
}
```

Difference between auto and static

```c
#include <stdio.h>

void testAuto() {
    auto int x = 0;  // Auto variable (reinitialized every time)
    x++;
    printf("Auto: %d\n", x);
}

void testStatic() {
    static int y = 0;  // Static variable (retains value)
    y++;
    printf("Static: %d\n", y);
}

int main() {
    printf("First Call:\n");
    testAuto();
    testStatic();

    printf("Second Call:\n");
    testAuto();
    testStatic();

    printf("Third Call:\n");
    testAuto();
    testStatic();

    return 0;
}
```

```
First Call:
Auto: 1
Static: 1

Second Call:
Auto: 1  // Reset
Static: 2  // Retained

Third Call:
Auto: 1  // Reset
Static: 3  // Retained
```

4.Extern Storage Class

- Stored in **Data Segment** (like static).
- **Scope: Global** (accessible across multiple files).
- **Lifetime:** Entire program execution.
- **Default Value:** 0 (if uninitialized).
- **Speciality:** Declares a global variable defined elsewhere.

Copy Edit

```c
#include <stdio.h>

// Extern storage class: Declared outside all functions
int globalVar = 100;

void testStorageClasses() {
    auto int autoVar = 10;  // Auto variable (reinitialized on each function call)
    register int regVar = 20;  // Register variable (stored in CPU register if possible)
    static int staticVar = 30;  // Static variable (retains value between function calls

    // Modify values
    autoVar++;
    regVar++;
    staticVar++;
    globalVar++;

    // Print values
    printf("Auto: %d\n", autoVar);
    printf("Register: %d\n", regVar);
    printf("Static: %d\n", staticVar);
    printf("Extern (Global): %d\n", globalVar);
}

int main() {
    printf("First Call:\n");
    testStorageClasses();

    printf("\nSecond Call:\n");
    testStorageClasses();

    printf("\nThird Call:\n");
    testStorageClasses();

    return 0;
}
```

```
First Call:
Auto: 11
Register: 21
Static: 31
Extern (Global): 101

Second Call:
Auto: 11  // Reset (local to function)
Register: 21  // Reset (like auto, but stored in a register)
Static: 32  // Retained
Extern (Global): 102  // Retained

Third Call:
Auto: 11  // Reset again
Register: 21  // Reset again
Static: 33  // Retained
Extern (Global): 103  // Retained
```

Storage Class	Storage Location	Scope	Lifetime	Default Value
`auto` (Default)	Stack	Local (function/block)	Created & destroyed in function call	Garbage (uninitialized)
`register`	CPU Register (if available)	Local (function/block)	Created & destroyed in function call	Garbage (uninitialized)
`static`	Data Segment (Fixed Memory)	Local (function/block)	Entire program execution	`0` (if uninitialized)
`extern`	Data Segment	Global (accessible from multiple files)	Entire program execution	`0` (if uninitialized)

auto & register: Behave similarly, **reset on function calls**. (register is faster if stored in CPU registers).

static: **Remembers its value** between function calls.

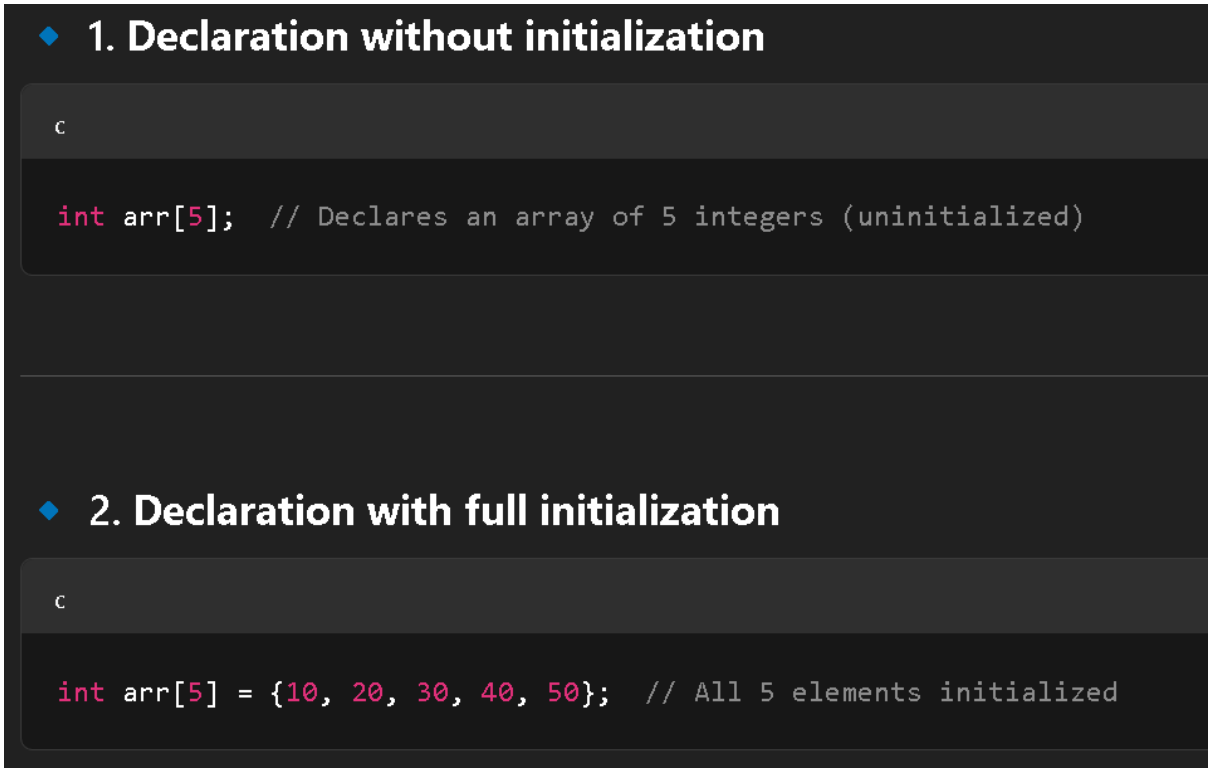
extern: **Global, accessible anywhere in the program**.

----------(unit- 3)----------

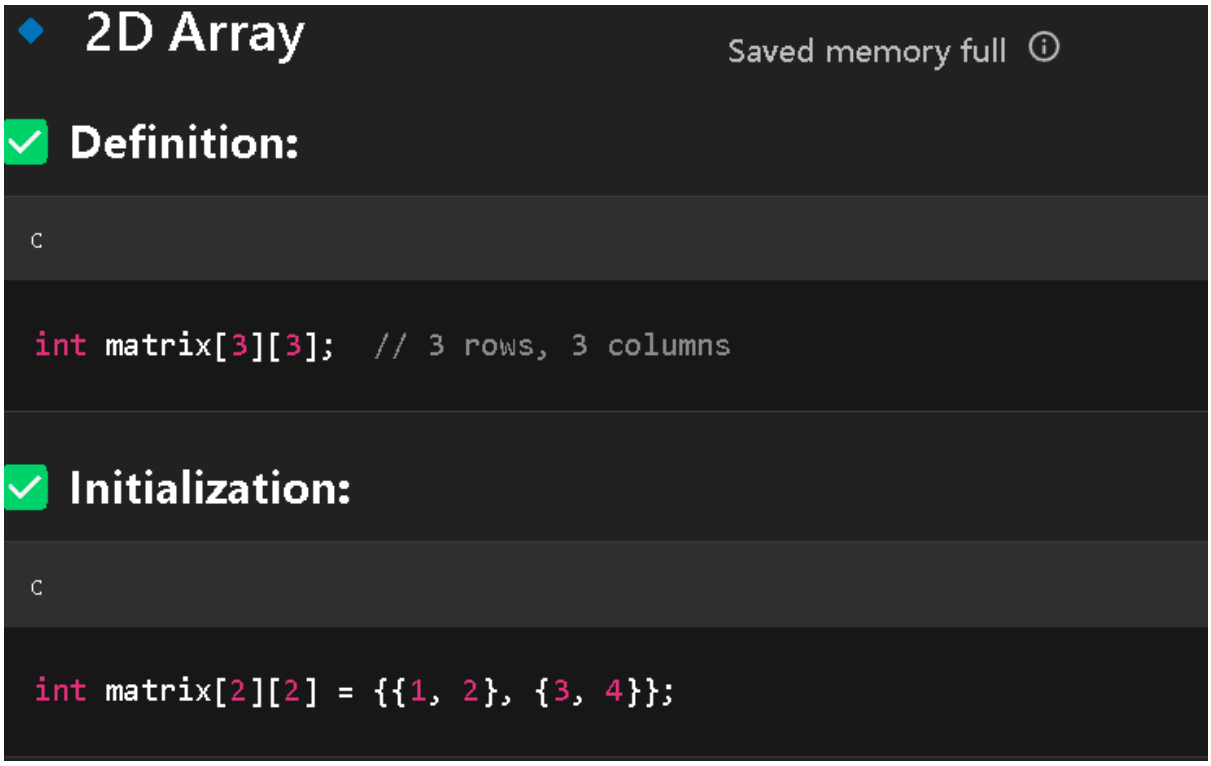
Unit-4:--

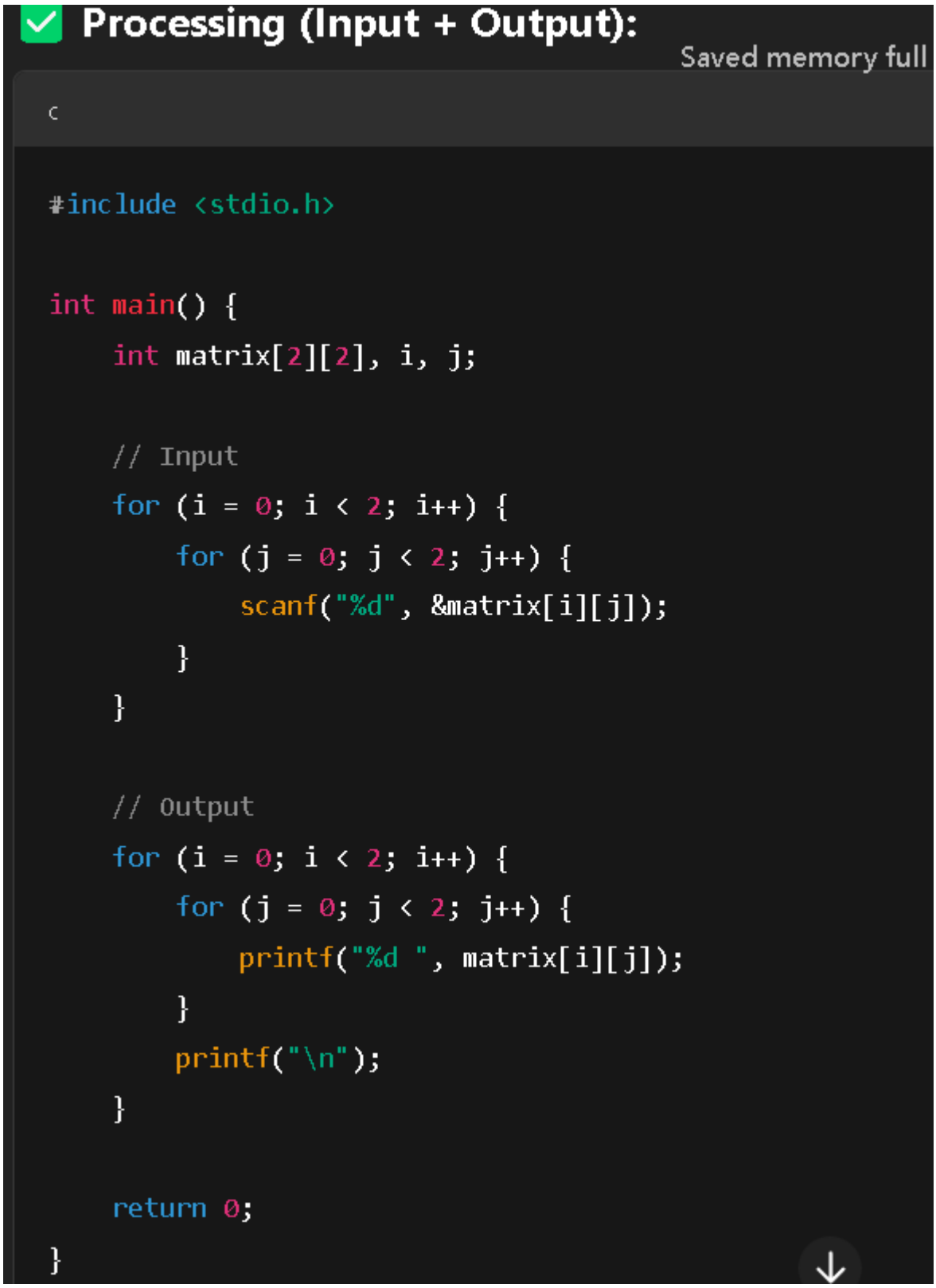
Arrays in C : Declaring and initializing arrays in C, Defining and processing 1D and 2D arrays, Array applications, Passing arrays to functions, inserting and deleting elements of an array, Searching including linear and binary search methods, Sorting of array using bubble sort.

Declaring and initializing arrays in c, declare 1D arrays



Defining and processing 2D arrays:--





Passing arrays to functions:--

Situation	Need Prototype?
Function defined **before** `main()`	❌ No
Function defined **after** `main()`	✅ Yes

```c
#include <stdio.h>

int displayAndSum(int[], int);  // Function prototype

int main() {
    int n, i;
    printf("Enter number of elements: ");
    scanf("%d", &n);
    int a[n];

    printf("Enter elements: \n");
    for (i = 0; i < n; i++) {
        scanf("%d", &a[i]);
    }

    int total = displayAndSum(a, n);
    printf("\nSum = %d", total);
    return 0;
}

// Function definition
int displayAndSum(int arr[], int n) {
    int sum = 0;
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
        sum += arr[i];
    }
    return sum;
}
```

```c
#include <stdio.h>

void display(int[], int);  // Function prototype

int main() {
    int n, i;
    printf("Enter number of elements: ");
    scanf("%d", &n);
    int a[n];

    printf("Enter elements: \n");
    for (i = 0; i < n; i++) {
        scanf("%d", &a[i]);
    }

    display(a, n);  // Function call
    return 0;
}

// Function definition
void display(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
}
```

🔹 Example (No prototype needed):

c Copy Edit

```c
#include <stdio.h>

// Function defined before main
void greet() {
    printf("Hello!\n");
}

int main() {
    greet();  // Function called after definition
    return 0;
}
```

✅ No prototype needed because the compiler already knows about `greet()` before `main()` uses it.

```c
#include <stdio.h>

// Function defined before main (no need for a prototype)
int display(int b[], int n1) {
    int i;
    for(i = 0; i < n1; i++) {
        printf("%d ", b[i]);  // Displaying array elements
    }
    return 0;  // Return 0 to indicate successful display (or any other return value if needed)
}

int main() {
    int n, i;
    printf("Enter number of elements: ");
    scanf("%d", &n);
    int a[n];

    printf("Enter elements to array: \n");
    for(i = 0; i < n; i++) {
        scanf("%d", &a[i]);
    }

    display(a, n);  // Calling the function to display array elements
    return 0;
}
```

If the function is defined before main(), you don't need a prototype (it works for both void and int functions).

1. Passing 1D array as an argument to the function
2. Passing 2D array as an argument to the function.

1.passing 1D array to functions:-- refer to examples

Unit – 5:--

Pointers in C (unit v)

Defn :-- pointer is a special type of variable that shows memory address of another variable.

Advantages of pointers:--

1.storage retrieval of a variable using its address is much faster than accessing the variable by its name.

2. for dynamic memory allocation, pointers are required.

3.call by reference→ any change done to formal arguments should affect value of actual arguments.

4.Manipulating arrays using pointers easily.

5.complex data structures like trees, graph, linked list, stack , que all areimplemented using pointer.

Computer memory (RAM) is a sequential collection of storage class.each cell is known as a byte which has an address associated with it.

Bits and bytes

8 bits = 1 byte

2^10 bytes = 1Kb

2^10 KB = 1MB

…..

whenever we declare a variable, eg. int a = 10

system allocates required memory to the variable

let us assume variable a is stored at memory address 5000

| variable --> a |

| value --> 10 |

| address --> 5000 |

say another variable p is storing the address of a. this p is a pointer to a

| variable --> p |

| value --> 5000 |

| address --> 6000 |

Accessing address of a variable (& operator is used)

P = &a;

This statement would assign 5000 to p.

Declaring pointer variable(* symbol)

Int a = 10; // single variable

Int *p; // pointer variable

P = &a' // initialization

Syntax:-- <data_type> * <ptr_name>;

Int *p;

Float *p;

Char *p;

Pointer flexibility

1. We can make some pointer point to different variables of some date type.

Int a, b, c;

Int *p;

P = &a;

.. ..

P =&b;

P = &c;

2.we can have multiple pointers pointing to some variable.

Int a;

```
Int *p1, *p2, *p3;


// WAP to calucate sqrt of number.
#include <stdio.h>
#include<math.h>
int main () {
    int n,s;
    printf("Enter a number: ");
    scanf("%d", &n);
    s =sqrt(n);
    printf("SQRT of number is %d",s);
    return 0;
}
```

how negative sign deals with /(division operator) and %(modulo operator)

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

```
    int i = 3;
    int l = i /-2;
    int k = i % -2;
    printf("%d and %d\n",l, k);
    return 0;
}
```

// (21-01-2025)

// always follow from left to right during complex arithmetic operations

// modulous always work on integer's not with float value

/* Bitwise operator: it operates on data bit by bit

6 types of bitwise operator

1.Bitwise complement(~)

2.Bitwise AND(&)

3.Bitwise OR(|)

4.Bitwise Xor(^)

5.Bitwise left shift <<

6.Bitwise right shift >>

*/

```
#include<stdio.h>
int main () {
    int a = 10;
    long unsigned int b;
    b = ~a; // bcz no. -11(tool's complement) is out of range of integer data type
    printf("\n%u",b);  // 4294967285
}

//int number store in memory in 16 bits(2 Bytes)
#include <stdio.h>
int main () {
    int a= 10, b = 13,c;
    c = a | b; // firstly convert into binary to both the number's a and b then add if | OR or if AND & then add or multiply according to their binary then binary to decimal convert at last
```

```
    printf("%d",c);
    return 0;
}

#include <stdio.h>
int main () {
    int a= 10, b = 13,c;
    c = a ^ b;  // 7
    printf("%d",c);
    return 0;
}

#include <stdio.h>
// left shift
int main () {
    int a= 10,c;
    c = a << 2;  // formula = no. * 2 ^ no. of bits shift
--> 10 * 2 ^ 2 = 10 * 40 = 40
    printf("%d",c);
```

```
    return 0;
}


#include <stdio.h>
// Right shift
int main () {
    int a= 20,c;
    c = a >> 2;  // formula = no. / 2 ^ no. of bits shift --> 10 * 2 ^ 2 = 10 * 40 = 40
    printf("%d",c);
    return 0;
}


// special operator (,)
#include <stdio.h>
int main () {
    int a = 2, b =3, x = 0;
    x =(++a, b+=a);// always after comma right side value assigned to variable this is the working of special operator ,
```

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


#include <stdio.h>
int main () {
    double b = 5 % 3 & 4 + 5 * 6 ;
    printf("%lf", b);
}


// Bitwise and operator if first cdn before and is false then everything is false
#include <stdio.h>
int main () {
    int a = -1, b = 4, c = 1, d;
    d = ++a {&&} ++b {||} ++c;
    printf("%d,%d,%d,%d",a,b,c,d);
    return 0;
}
```

```
// if only write statement 1 then don't use curly bracket in if,else if, else

#include <stdio.h>

int main () {

    int x = 0;

    if(x)

    {

        printf("Hi"); // if cdn is false then it goes towards else

    }

    else{

        printf("Hello");

    }

    return 0;

}


// if only write statement 1 then don't use curly bracket in if,else if, else

#include <stdio.h>

int main () {
```

```
    int x = 0;
    if(x++)
    {
        printf("Hi"); // if cdn is false then it goes towards else
    }
    else if (x == 1){
        printf("Hello");
    }
    return 0;
}
```

```
#include <stdio.h>
int main () {
    int num1, num2;
    printf("Please enter both the number's: ");
    scanf("%d%d", &num1,&num2);
```

```
    num1 > num2  ? printf("Num1 is greater") :
printf("Num2 is greater");

    return 0;

}


#include <stdio.h>

int main () {

    int a,b,c;

    printf("Please enter both the number's: ");

    scanf("%d%d%d", &a,&b,&c);


    (a > b && a >c) ? printf("a is greater") :(b > a
&& b > c) ? printf("b is greater"):printf("c is
greater");

    return 0;

}


#include <stdio.h>

int main () {

    int i;
```

```
    for (i = 10; i > 0 ; i --) // Decrement
    {
        printf("Prem Kumar Gupta\n");
    }
    return 0;
}

#include <stdio.h>
int main () {
    int i;
    for (i = 1; i < 100 ; i++) // increment
    {
        if (i % 2 == 0){
        printf("%d\n", i);
    }
    }
    return 0;
}
```

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

#include <stdio.h>
int main () {
    int i,num,sum = 0;
    printf("Enter the value of n: ");
    scanf("%d", &num);
    for (i = 1; i <= num ; i++) // increment
    {
        if (i % 2 == 0){
            sum = sum + i ;
```

```
    }
    }
        printf("%d\n", sum);
    return 0;
}
```

```
#include <stdio.h>
int main () {
    int i,j;
    for(i = 1; i <= 5; i++)
    {
        for(j = 1; j <= i; j++)
        {
            printf("*");
        }
        printf("\n");
    }
    return 0;
}
```

```
// write a program to enter a no. of count digits in it
#include <stdio.h>
int main () {
    int no, ctr = 0;
    printf("enter a number: ");
    scanf("%d", &no);
    while(no != 0){
        no = no / 10;
        ctr++;
    }
    printf("%d", ctr);
    return 0;
}
```

```
// write a program to count even number of digits in it.
#include <stdio.h>
int main () {
```

```
    int no,rem, ctr = 0;
    printf("enter a number: ");
    scanf("%d", &no);
    while(no != 0){
        rem  = no % 10;
        if(rem % 2 == 0){
            ctr++;
        }
        no = no / 10;
    }
    printf("%d", ctr);
    return 0;
}

// write a program to reverse the number.
#include <stdio.h>
int main () {
    int no, rem,rev = 0;
    printf("enter a number: ");
```

```
    scanf("%d", &no);
    while(no != 0)
    {
        rem  = no % 10;
        rev = rev * 10+ rem;
        no = no / 10;
    }
    printf("%d", rev);
    return 0;
}
```

// write a program to check the number is palimdrome or not.

```
#include <stdio.h>
int main () {
    int no, rem,rev = 0, a;
    printf("enter a number: ");
    scanf("%d", &no);
    a = no;
```

```
    while(no != 0)
    {
        rem  = no % 10;
        rev = rev * 10+ rem;
        no = no / 10;
    }
    if (a == rev )
    {printf("Palimdrome no");
    }
    else {
        printf("Not palimdrome");
    }
    return 0;
}
```

// write a program to check whether the number is armstrong number or not.

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

```
    int no, rem,sum = 0, a;
    printf("enter a number: ");
    scanf("%d", &no);
    a = no;
    while(no != 0)
    {
        rem  = no % 10;
        sum = sum + rem * rem * rem;
        no = no / 10;
    }
    if (a == sum )
    {printf("armstraong no");
    }
    else {
        printf("Not armstrong");
    }
    return 0;
}
```

// write a program to print all armstrong number between 1 and 10,000.

```
#include <stdio.h>

int main () {
    int n, rem, sum = 0,i,a, b;
    printf("Enter range: ");
    scanf("%d", &n);
    for (i = 1; i<= n; i++) {
        b = i;
        a = b;
        while(b != 0){
            rem = b % 10;
            sum += rem * rem * rem;
            b = b / 10;
        }
        if (sum == a){
            printf("%d\n",i);
        }
        sum = 0;
```

```
    }
        return 0;
    }

// wap to find the number is prime or not.
#include <stdio.h>
int main () {
    int num,i;
    printf("please enter the number: ");
    scanf("%d", &num);

    if (num == 1){
        printf("Not a prime");
    }
    for (i = 2; i <= num; i++){
        if(num % i == 0) {
            break;
        }
    }
```

```
        if (i == num) {
            printf("Prime number.");
        }
        else {
            printf("not a prime.");
        }
        return 0}

// in switch statement maximum 257 case exists.
// wap greater of two no's using goto.
#include <stdio.h>
int main () {
    int a,b;
    printf("Enter two number's: ");
    scanf("%d %d", &a, &b);
    if ( a > b)
    {
        goto output1;
    }
```

```
    else {
        goto output2;
    }
    output1:
    printf("a is greater");
    exit(1);
    output2:
    printf("b is greater");
    return 0;
}
```

```
#include <stdio.h>
#include<stdlib.h>
int main () {
    int a,b;
    printf("Enter two number's: ");
    scanf("%d %d", &a, &b);
    if ( a > b)
```

```
    {
        goto output1;
    }
    else {
        goto output2;
    }
    output1:
    printf("a is greater");
    exit(0); // for this we use #include<stdlib.h>
    output2:
    printf("b is greater");
    return 0;
}
```

//wap to print the sqrt of a number enter by the user.

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

```
    int x,b;
    tryagain:
    printf("Enter the number's: ");
    scanf("%d", &x);
    if (x < 0)
    {
        goto tryagain;
    }
    else {
        b = sqrt(x);
        printf("%d", b);
    }
    return 0;
}

// unformatted functions
#include <stdio.h>
int main () {
    char ch;
```

```
    printf("Enter a character: ");
    ch = getchar();
    printf("\ncharacter entered is %c", ch);
    return 0;
}

#include <stdio.h>
int main () {
    char ch;
    printf("Enter a character: ");
    ch = getch();
    printf("\ncharacter entered is %c", ch);
    return 0;
}

#include <stdio.h>
int main () {
    char ch;
    printf("Enter a character: ");
```

```
    ch = getche();
    printf("\ncharacter entered is %c", ch);
    return 0;
}

//?? doubt {putch and putche}
#include <stdio.h>
int main () {
    char ch;
    printf("Enter a character: ");
    ch = getch();
    putch(ch);
    return 0;
}

#include <stdio.h>
int main () {
    char name[20];
    printf("Enter name: ");
```

```
    scanf("%s", &name);
    printf("%s", name);
    return 0;
}

#include <stdio.h>
int main () {
    char name[20];
    printf("Enter name: ");
    gets(name);
    printf("%s", name);
    return 0;
}

// implicit data-type example
#include <stdio.h>
int main () {
    float a = 5.99;
    int b;
```

```
    b = a;
    printf("%d", b);
    return 0;
}

#include <stdio.h>
int main () {
    int a = 5;
    float b;
    b = a/2;
    printf("%.2f", b);
    return 0;
}

//wap to print the table of number enter by the user.
#include<stdio.h>
int main () {
    int num,i;
    printf("Welcome to print the tables.\n");
```

```
    printf("please enter the number: ");
    scanf("%d", &num);
    for (i = 1; i <=10; i++){
        printf("%d * %d  = %d\n", num, i, num * i);
    }
    return 0;
}

//wap to check whether the year is leap year or not.
#include<stdio.h>
int main() {
    int num;
    printf("Enter the year: ");
    scanf("%d", &num);
    if((num % 4 == 0 && num % 100 != 0) ||( num % 100 == 0)) {
        printf("This year is an leap year");
    }
    else {
```

```
        printf("This year is not an leap year");
    }
    return 0;
}
```

// wap to swap of two number's without using temp variable

```
#include<stdio.h>
int main() {
    int a,b;
    printf("Enter the number's: ");
    scanf("%d %d", &a, &b);
    a = a + b;
    b = a - b;
    a = a- b;
    printf("%d %d", a, b);
    return 0;
}
```

```
#include<stdio.h>
int main (void) {
    printf("%d\n", 890);
    printf("%d\n", 890);
    printf("%d\n", +890);
    printf("%d\n", -890);
    printf("%hd\n", 25890);  // short integers
    printf("%ld\n", 20000000000l); // long integers
    // literal a long
    printf("%u\n", 890);
    printf("%u\n", -890);
}

#include<stdio.h>
int main ( void ) {
    printf("%e\n", 1234567.89);  // here 1.234568e+006 is output == means ==> 1.234568 * 10^(006)
    printf("%e\n", -1234567.89);
```

```
    printf("%E\n", 1234567.89);
    printf("%f\n", 1234567.89);
    printf("%g\n", 1234567.89);
    printf("%G\n", 1234567.89);
    printf("100%%");
}

#include<stdio.h>
int main () {
    printf("%8d\n", 1234);
    printf("%7d\n", 1234);
    printf("%6d\n\n", 1234);
    // printf("100%%");
}

// write a program to print all the fibnacci series till 100.
#include <stdio.h>
int main () {
```

```
    int n, a = 0, b = 1, fib = 0;
    printf("Enter range: ");
    scanf("%d", &n);
    int i = 1;
    do
    {
        printf("%d\n", fib);
        a = b;
        b = fib;
        fib = a + b;
    i++;
    } while (i <= n);
    return 0;
}

#include <stdio.h>

void x(); // jitne fn honge utne void lagenge
```

```
void y(); // the sequence in which fncs are defined/declared does not matters in what sequence they run and print the statements.

void z();

int main () { // main is an calling fn

    x();   // called fn is x

    printf("I am in main.\n");

    return 0;

}

void x(){

    y();

    printf("I am in x\n");

}


void y(){

    z();

    printf("I am in y\n");

}
```

```
void z(){
    printf("I am in z\n");
}

// wap to sum of two number's by using fncs
#include <stdio.h>
void add (); // prototype declaration
int main () {
    add ();
    return 0;
}
void add () {
    int a,b,c;
    printf("Enter 2 numbers: ");
    scanf("%d %d", &a, &b);
    c = a + b;
    printf("%d", c);
}
```

```
// wap to sum of two number's by using fncs

#include <stdio.h>

void add (int, int);

int main () {

    int a, b; // the arguments present in fn call eg a and b :: Actual argument

    printf("Enter 2 numbers: ");

    scanf("%d %d", &a, &b);

    add(a, b);

    return 0;

}

void add (int x, int y) { // argument present in fn defn eg; x and y:: formal argument

    int z;

    z = x + y;

    printf("%d", z);

}


// wap to reverse a no. but using a fn reverse()
```

```
#include<stdio.h>
void reverse (int);
int main () {
    int no;
    printf("Enter a no: ");
    scanf("%d", &no);
    reverse(no);
    return 0;
}
void reverse(no){
    int rem, rev = 0;
    while (no != 0) {
        rem = no % 10;
        rev = rev * 10 + rem;
        no = no / 10;
}
    printf("%d", rev);
}
```

```
// wap to calculate factorial of a no. using fn fact()
#include <stdio.h>
void fact(int);
int main () {
    int no;
    printf("Enter a no: ");
    scanf("%d", &no);
    fact(no);
    return 0;
}
void fact (int m) {
    int i, fact = 1;
    for (i = 1; i<= m; i++){
        fact = fact * i;
    }
    printf("%d", fact);
}

// wap to calculate factorial of a no. using fn fact()
```

```
#include <stdio.h>
int fact(int);
int main () {
    int no;
    printf("Enter a no: ");
    scanf("%d", &no);
    printf("%d", fact(no));
    return 0;
}
int fact (int m) {
    int i, fact = 1;
    for (i = 1; i<= m; i++){
        fact = fact * i;
    }
    return fact;
}

// wap to add two no.s by using return
#include <stdio.h>
```

```
int add(int, int);
int main() {
    int a,b,c;
    printf("Enter 2 no: ");
    scanf("%d %d", &a, &b);
    c = add(a, b);
    printf("%d", c);
    return 0;
}
int add(int x, int y) {
    int z ;
    z = x + y;
    return z;
}

// wap to add two no.s by using return
#include <stdio.h>
int large(int x, int y) {
    if (x > y)
```

```
    return x;
    else
    return y;
}
int main() {
    int a,b,c;
    printf("Enter 2 no: ");
    scanf("%d %d", &a, &b);
    c = large(a, b);
    printf("%d", c);
    return 0;
}

// wap to swap two number's by using functions
#include<stdio.h>
void swap(a,b);
int main (){
    int a,b;
    printf("Please enter the number: ");
```

```
    scanf("%d %d", &a, &b);
    swap(a,b);
    printf("%d %d", a,b);
    return 0;
}

void swap(int x, int y){
    int z;
    z = x;
    x = y;
    y = z;
    printf("%d %d\n", x,y);
}

//pointer concept
#include<stdio.h>
int main() {
    int a = 10; // variable declaration
    int *p; // pointer declaration
```

```
    p = &a; // pointer initialization / referencing(technical terms)
    printf("%d", a);
    printf("%d", *p); // deferencing
    return 0;
}

#include <stdio.h>
void swap(int *, int *);
int main (){
    int a,b;
    printf("Please enter the number: ");
    scanf("%d %d", &a, &b);
    swap(&a, &b); // call by refference
    printf("%d %d", a,b);
    return 0;
}

void swap(int * x, int *y){
```

```
    int z;
    z = *x;
   *x = *y;
   *y = z;
    printf("%d %d\n", x,y);
}

//wap to reverse a no. but using a fn reverse()
#include<stdio.h>
void reverse (int *);
int main (){
    int no;
    printf("Enter a no: ");
    scanf("%d", &no);
    reverse(& no);
}
void reverse(int *x){
int rem, rev = 0;
while (*x != 0 ){
```

```
    rem = *x % 10;
    rev = rev * 10 + rem;
    *x = *x / 10;
}
printf("%d", rev);
}
```

// fn call itself infinite times -- recursive fn

// wap to check whether the number is prime or not using function.

```
#include <stdio.h>

int prime(int);

int main() {
    int n, result;
    printf("Enter a number: ");
    scanf("%d", &n);
```

```
    result = prime(n);
    if (result == 0) {
        printf("\nNumber is prime");
    } else {
        printf("\nNumber is not prime");
    }
    return 0;
}

int prime(int x) {
    int i, status = 0;
    for (i = 2; i <= x / 2; i++) {
        if (x % i == 0) {
            status = 1;
            break;
        }
    }
    return status;
}
```

```
// wap to calculate the factorial of number
#include <stdio.h>
int main (){
    int n, fact = 1;
    scanf("%d", & n);
    for (int i = 1; i <= n; i++){
        fact = fact * i;
    }
    printf("%d", fact);
    if (n == 0) {
        printf("%d", 1);
    }
    return 0;
}
```

/* Recursion:-- phenomena of a function to call itself again and again is called recursion .

such a function calls itself again and again is called recursive function.

2 main conditions for recursion :--

1. Recursive condition:-- function calling itself again and again

2. Stopping condition:-- which ends the function call.*/

// wap to print the factorial of a number"

```
#include<stdio.h>
int fact(int);
int main ()
{
    int no;
    printf("Please enter the number: ");
    scanf("%d", &no);
    printf("Factorial of %d is %d", no, fact(no));
    return 0;
```

```
}
int fact(int x) {
    if (x == 1 || x ==0) {
        return 1;
    }
    else {
        return (x * fact(x-1));
    }
}

#include <stdio.h>
int fib(int);
int main () {
    int no, a = 0, b = 1,i;
    printf("please enter the number: ");
    scanf("%d", &no);
    printf("%d\n%d\n", a,b);
    for ( i = 1; i <= no - 2; i++)
    {
```

```
        printf("%d\n", fib(i));
    }
    }
    int fib(int x) {
        if (x == 0 || x== 1) {
        return 1;
        }
        else {
        return (fib(x-1) + fib(x - 2));
        }}
```

```
#include <stdio.h>
void fn(int) ;
int main () {
    fn(5);
    return 0;
}
void fn(int x) {
    if (x > 0) {
```

```
        fn(--x);
    }
    printf("%d", x);
}

#include<stdio.h>
void inc(int);
int main () {
    int i ;
    for (i = 0; i < 3; inc(i)){
        printf("%d",i);
    }
    return 0;
}
void inc(int i) {
    i++;
}

#include<stdio.h>
```

```
int inc(int *);
int main () {
    int i ;
    for (i = 0; i < 3; inc(&i)){
        printf("%d",i);
    }
    return 0;
}
int inc(int *x) {
    return *x++;
}
```

```
#include <stdio.h>
int addnumbers(int n);
int main () {
    int num;
    printf("please enter the number: ");
    scanf("%d", &num);
```

```
    printf("%d\n", addnumbers(num));
    }
    int addnumbers(int n) {
        if (n == 0) {
        return n;
        }
        else {
        return n + addnumbers(n-1);
        }}



        #include <stdio.h>
#include<math.h>
int main () {
        double x = 9.0 , y = 8.0, z = 7.0;
        printf("\nLog value is: %lf", log(x));
        printf("\nLog value with base 10 is: %lf",
log10(x));
        printf("\nexponential value is: %lf", exp(x));
```

```
        printf("\nCeil value is: %lf", ceil(8.94));
        printf("\nfloor value is: %lf", floor(2.34));
        printf("\npower value is: %lf", pow(3.0,2.0));
        printf("\nfloating value is: %lf", fabs(-2.9));
        printf("\nSquare root value is: %lf", sqrt(9));
        printf("\nsin:%f,cos%f,tan:%lf",
sin(x),cos(y),tan(z));
        printf("\nfMod:%lf", fmod(2.0,1.5));
        return 0;
}


#include <stdio.h>
int myrecursivefn(int);
int main () {
        myrecursivefn(5);
        return 0;
}
int myrecursivefn(int x){
        if(x == 0)
```

```
        return 0;
        printf("%d", x);
        return myrecursivefn(x-1);
}

#include <stdio.h>
int fn(int);
int main () {
        printf("%d", fn(2));
        return 0;
}
int fn(int x) {
        if (x == 4)
        return x;
        else
        return 2 * fn(x+1);
}

#include <stdio.h>
```

```
int fun(int, int);
int main () {
	printf("%d", fun(4,3));
	return 0;
}
int fun(int x, int y) {
	if (x == 0)
	return y;
	else
	return fun(x - 1, x + y);
}

// wap to using gets and puts
#include<stdio.h>
int main () {
	char name[50];
	printf("Enter your name: ");
	fgets(name, sizeof(name), stdin);
	printf("Your name is: ");
```

```
    puts(name);
    return 0;
}
```

```
#include <stdio.h>
#include <conio.h>  // Required for getch() and getche()

int main() {
    char ch1, ch2, ch3;

    printf("Enter a character using getchar(): ");
    ch1 = getchar();  // Press Enter after input

    printf("\nEnter a character using getch(): ");
    ch2 = getch();  // Does not show input on screen

    printf("\nEnter a character using getche(): ");
    ch3 = getche();  // Displays input without Enter
```

```
    printf("\n\nYou entered: %c, %c, %c\n", ch1,
ch2, ch3);

    return 0;

}



//wap to check whether the year is leap year or not.

#include<stdio.h>

int main() {

    int num;

    printf("Enter the year: ");

    scanf("%d", &num);

    if((num % 4 == 0 && num % 100 != 0) || num %
400 == 0) {

        printf("This year is an leap year");

    }

    else {

        printf("This year is not an leap year");
```

```
    }
    return 0;
}
```

// wap to take input integers and tell which one first repeated digits

```
#include<stdio.h>
int main () {
    int n, temp, check, digit, found = -1;
    scanf("%d", &n);
    temp = n;
    while (temp > 0 && found == -1) {
        digit = temp % 10;
        check = n /10;
        while (check > 0) {
            if (check % 10 == digit) {
                found = digit;
                break;
            }
```

```
            check /=10;
        }
        temp /=10;
    }
    if (found != -1)
        printf("%d", found);

    return 0;
}

// wap to determine first factor of the number.
#include <stdio.h>
int main () {
    long long int n;
    long long factor = 2;
    scanf("%lld", &n);
    while (n > 1) {
        if (n % factor == 0) {
            printf("%lld\n", factor);
```

```
            break;
        }else
            factor++;
    }
    return 0;
}

#include<stdio.h>
int main () {
    int road_type, current_speed, speed_limit;
    scanf("%d", &road_type);
    scanf("%d", &current_speed);
    switch (road_type) {
        case 1:
        speed_limit = 25;
        case 2:
        speed_limit = 35;
        case 3:
        speed_limit = 55;
```

```
    }
    if (current_speed < speed_limit) {
        printf("Below");
    }
    else if (current_speed == speed_limit) {
        printf("Equal");
    }
    else {
        printf("Above");
    }
    return 0;
}


// <storage class> variable name = value
// if not mention any storage class it defines by default automatic storage class



//  static storage class do
```

/*

1. variable stored in RAM

2. default value if variable is unitialized

3. life time --> is till the control remains in that block in which it is depend (till inside main function i.e int main ())

4.Local scope of variable

*/

// register storage class do

/*

1. variable stored in CPU

2. default value if variable is unitialized

3. life time --> is till the control remains in that block in which it is depend (till inside main function i.e int main ())

4.Local scope of variable

*/

```
//  static storage class do
/*
1. variable stored in RAM
2. zero value if variable is unitialized
3. life time --> persists between two fn calls
4.Local scope of variable


*/



#include<stdio.h>
void function1();
void function2();
int main () {
    auto int a = 999;
    function1();
    printf("%d ", a);
```

```
    return 0;
}

void function1() {
    auto int a = 99;
    function2();
    printf("%d ", a);
}

void function2() {
    auto int a = 9;
    printf("%d ", a);
}



#include<stdio.h>
int main () {
    register int i, sqr;
    for (i = 1; i <= 5; i++) {
```

```
        sqr = i * i;
        printf("%d \t %d\n", i, sqr);
    }
    return 0;
}

#include<stdio.h>
void call();
int main () {
    call();
    call();
    return 0;
}

void call(){
    int i = 0;
    register int j = 0;
```

```
    static int k = 0; // once change is done that change is permanent in case of static (every time a new value is generated)
    i++;
    j++;
    k++;
    printf("%d\n%d\n%d\n", i,j,k);
}


/*
4. external storage class:-- extern
a. variable is stored in ram.
b. default value is zero.
c. scope is global.
d. lifetime--> till the program does not end
*/
#include<stdio.h>
void output();
extern int num = 100;
```

```
int main() {
    output();
    return 0;
}
void output(){
    printf("%d", num);
}


/*
call by value: --
1. only a copy of actual argument is passed to the function.
2.any change done to formal argument does not affect value of actual argument.
// */
// #include<stdio.h>
// void add(int);
// int main () {
//     int num = 2;
```

```
//     printf("value of num before calling the function: %d\n", num);

//     add(num);

//     printf("Value of num after calling the function: %d\n", num);

//     return 0;

// }

// void add(int x){

//     x = x + 10;

//     printf("value inside function: %d\n", x);

// }


/*

call by reference:

1.here actual memory address of argument is passed.

2 any change done to formal argument will automatically affect the actual argument.

*/

#include <stdio.h>
```

```
void add(int *);


int main() {
    int num = 2;
    printf("Value of num before calling the function: %d\n", num);
    add(&num);  // ✅ Passing address of num
    printf("Value of num after calling the function: %d\n", num);
    return 0;
}


void add(int *x) {
    *x = *x + 10;  // Modifying the original variable
    printf("Value inside function: %d\n", *x);
}


#include <stdio.h>
```

```
#include <string.h>

// 1. Factorial
int factorial(int n) {
    if (n == 0) return 1;
    return n * factorial(n - 1);
}

// 2. Fibonacci
int fibonacci(int n) {
    if (n == 0) return 0;
    if (n == 1) return 1;
    return fibonacci(n - 1) + fibonacci(n - 2);
}

// 3. Sum of digits
int sumOfDigits(int n) {
    if (n == 0) return 0;
    return (n % 10) + sumOfDigits(n / 10);
```

```
}

// 4. Reverse a string
void reverseString(char str[], int start, int end) {
    if (start >= end) return;
    char temp = str[start];
    str[start] = str[end];
    str[end] = temp;
    reverseString(str, start + 1, end - 1);
}

// 5. Power calculation (x^n)
int power(int x, int n) {
    if (n == 0) return 1;
    return x * power(x, n - 1);
}

// 6. GCD using Euclidean algorithm
int gcd(int a, int b) {
```

```
    if (b == 0) return a;
    return gcd(b, a % b);
}

// 7. Tower of Hanoi
void hanoi(int n, char src, char aux, char dest) {
    if (n == 1) {
        printf("Move disk 1 from %c to %c\n", src, dest);
        return;
    }
    hanoi(n - 1, src, dest, aux);
    printf("Move disk %d from %c to %c\n", n, src, dest);
    hanoi(n - 1, aux, src, dest);
}

// 8. Generate subsets (Power Set)
void generateSubsets(char *set, char *subset, int i, int j) {
```

```
    if (set[i] == '\0') {
        subset[j] = '\0';
        printf("%s\n", subset);
        return;
    }
    subset[j] = set[i];
    generateSubsets(set, subset, i + 1, j + 1); // include
    generateSubsets(set, subset, i + 1, j);     // exclude
}

// 9. Permutations of string
void permute(char *str, int l, int r) {
    if (l == r) {
        printf("%s\n", str);
        return;
    }
    for (int i = l; i <= r; i++) {
```

```
        char temp = str[l];
        str[l] = str[i];
        str[i] = temp;

        permute(str, l + 1, r);

        temp = str[l];
        str[l] = str[i];
        str[i] = temp;
    }
}

// 10. N-Queens (Simplified for 4x4)
#define N 4
int board[N][N];

int isSafe(int row, int col) {
    for (int i = 0; i < col; i++)
        if (board[row][i]) return 0;
```

```
    for (int i = row, j = col; i >= 0 && j >= 0; i--, j--)
        if (board[i][j]) return 0;
    for (int i = row, j = col; i < N && j >= 0; i++, j--
)
        if (board[i][j]) return 0;
    return 1;
}

int solveNQueens(int col) {
    if (col >= N) return 1;
    for (int i = 0; i < N; i++) {
        if (isSafe(i, col)) {
            board[i][col] = 1;
            if (solveNQueens(col + 1)) return 1;
            board[i][col] = 0;
        }
    }
    return 0;
}
```

```
void printBoard() {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++)
            printf("%d ", board[i][j]);
        printf("\n");
    }
}

int main() {
    printf("Factorial(5): %d\n", factorial(5));
    printf("Fibonacci(6): %d\n", fibonacci(6));
    printf("Sum of digits (1234): %d\n",
sumOfDigits(1234));

    char str1[] = "hello";
    reverseString(str1, 0, strlen(str1) - 1);
    printf("Reversed String: %s\n", str1);
```

```
printf("2^5 = %d\n", power(2, 5));
printf("GCD of 48 and 18: %d\n", gcd(48, 18));

printf("\nTower of Hanoi (3 disks):\n");
hanoi(3, 'A', 'B', 'C');

printf("\nSubsets of \"abc\":\n");
char subset[10];
generateSubsets("abc", subset, 0, 0);

printf("\nPermutations of \"abc\":\n");
char str2[] = "abc";
permute(str2, 0, strlen(str2) - 1);

printf("\nN-Queens Solution (4x4):\n");
if (solveNQueens(0))
    printBoard();
else
    printf("No solution found.\n");
```

```
    return 0;
}


//wap to calculate largest prime factor of no
#include <stdio.h>
int largest_prime(int);
int main()
{   int num,r;
    printf("enter a num: ");
    scanf("%d",&num);
    r = largest_prime(num);

    printf(" %d \n",r);
    return 0;
}
int largest_prime(int n)
{
```

```
    int i = 2;
    while (i != n)
    {
        if (n%i ==0)
        {
            n = n/i;
        }
        else
        i++;
    }
    return n;
}

#include<stdio.h>
int iseven(int);
int main () {
    int no;
    printf("Enter the number: ");
    scanf("%d", &no);
```

```
    iseven(no) ? printf("Even") : printf("Odd");
    return 0;
}
int iseven(int x){
    return ((x/2) * 2 == x);
}

// wap to check power of 2
#include<stdio.h>
int iseven(int);
int main () {
    int no;
    printf("Enter the number: ");
    scanf("%d", &no);
    iseven(no) ? printf("power") : printf("not");
    return 0;
}
int iseven(int no){
    return ((no & (no-1)) == 0);
```

```
}


#include<stdio.h>
int ispower(int);
int main () {
    int no;
    printf("Enter the number: ");
    scanf("%d", &no);
    ispower(no) ? printf("power") : printf("not");
    return 0;
}
int ispower(int no){
    if (no == 0)
    return 0;
    while(no != 1){
        if(no % 2 != 0){
        return 0;
    }
    no = no / 2;
```

```
    }
    return 1;
}


#include<stdio.h>
#include<string.h>
int main () {
    int i, j, len;
    char num[100];
    printf("Enter a number: ");
    scanf("%s", num);
    len = strlen(num); // calulate the length of any fn using strlen(include<string.h>)
for (i = 0; i <= len; i++)
{
    for (j = i + 1; j < len; j++){
    if (num[i] == num[j]){
        printf("1st repeated digit is: %c", num[i]);
        return 0;
```

```
    }
    }
}
printf("No repeated digit");
return 0;
}

#include<stdio.h>
int x;
void m();
int main ()
{
    m();
    printf("%d", x);
    return 0;
}
void m()
{
    x = 4;
```

```
}


#include<stdio.h>
int x = 5;
void m();
void n();
int main ()
{
    int x = 3;
    m();
    printf("%d", x);
    return 0;
}
void m() {
    x = 8;
    n();
}
void n (){
    printf("%d",x);
```

```
}

// local variable(scope) prioritise most than global if local variable and print fn is in one block



#include<stdio.h>

int main(){

register int a;

printf("Enter: ");

scanf("%d", a);

return 0;

}

//

// local variable(scope) prioritise most than global if local variable and print fn is in one block

#include<stdio.h>

int main () {

    register int i, sqr;

    for (i = 1; i <= 5; i++) {
```

```
        sqr = i * i;
        printf("%d \t %d\n", i, sqr);
    }
    return 0;
}
```

//(16-03-2025) WAP to input n number’s in a 1D array, then ask to enter another number and count how many times this number is present in a 1D array.

```
#include<stdio.h>
int main() {
    int a[100], i, n, ctr = 0, b;
    printf("Enter no of element: ");
    scanf("%d", &n);
    printf("Enter elements to the array: \n");
    for(i = 0; i < n; i++){
        scanf("%d", &a[i]);
    }
    printf("Enter another no: ");
```

```
    scanf("%d", &b);
    for(i = 0; i<n; i++){
        if(a[i] == b)
        ctr++;
    }
    printf("%d", ctr);
    return 0;
}
```

// 2. sorting of 1D array in Ascending/Descending order

```
#include<stdio.h>
int main() {
    int a[100], i, j, n, temp;
    printf("Enter no of elements: ");
    scanf("%d", &n);
    printf("Enter elements to array: ");
    for(i = 0; i <n;i++){
        scanf("%d", &a[i]);
```

```
    }
    for(i = 0; i<n; i++){
    for(j = i+1; j<n;j++){
        if( a[i] < a[j]){ // for ascending sign change >
            temp = a[i];
            a[i] = a[j];
            a[j] = temp;
        }
    }
    }
    for(i = 0; i < n;i++){
        printf("%d ", a[i]);
    }
    return 0;
}
```

// 3.wap to find factorial of no's from 1 to n and store factorial in 1D array.

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

```
int main (){
    int a[100], i, n, fact = 1;
    printf("Enter no. of elements: ");
    scanf("%d", &n);
    for (i = 0; i < n; i++){
        fact = fact * (i + 1);
        a[i] = fact;
    }
    printf("Array with factorial is: ");
    for(i = 0; i<n; i++){
        printf("%d",a[i]);
    }
    return 0;
}
```

// 4.WAP to find factorial of even no's from 1 to n and store factorial in 1D array.

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

```
    int a[100], i, n, fact = 1, j, k = 0;
    printf("Enter no. of elements: ");
    scanf("%d", &n);
    for (i = 1; i<=n; i++){
        if(i % 2 ==0)
        {
        for(j = i; j >=1 ;j--){
            fact = fact * j;
            }
        a[k] = fact;
        k++;
        fact = 1;
        }
    }
    for(i = 0; i<k; i++){
        printf("%d ", a[i]);
    }
    return 0;
}
```

```
//5.wap to insert a no. of particular position in 1D
#include<stdio.h>
int main(){
    int a[100], n, i, pos, no;
    printf("Enter range: ");
    scanf("%d", &n);
    for(i = 0; i<n; i++) {
        scanf("%d", &a[i]);
    }
    printf("Enter position: ");
    scanf("%d", &pos);
    printf("Enter no to be inserted: ");
    scanf("%d", &no);
    for(i = n; i > pos; i--){
        a[i] = a[i-1];
    }
    a[pos] = no;
    for(i = 0; i<=n;i++){
        printf("%d", a[i]);
```

```
    }
    return 0;
}

//6.wap to delete a no. of particular position in 1D
#include<stdio.h>
int main(){
    int a[100], n, i, pos;
    printf("Enter no of elements:\n ");
    scanf("%d", &n);
    printf("Enter elements to array ");
    for(i = 0; i<n; i++) {
        scanf("%d", &a[i]);
    }
    // scanf("%d", &pos);
    printf("Enter position: ");
    scanf("%d", &pos);
    for(i = pos; i < n - 1; i++){
        a[i] = a[i+1];
```

```
    }
    printf("Find array of: ");
    for(i = 0; i < n - 1; i++){
        printf("%d",a[i]);
    }
    return 0;
}
```

//deleting an element at particular position in 1D array

//7.wap to input n elements in 1D array, ask user to enter an element and delete the element if present in 1d array

```
#include<stdio.h>
int main(){
    int a[100], n, i, no, k;
    printf("Enter range\n ");
    scanf("%d", &n);
    printf("Enter elements to array\n ");
```

```
    for(i = 0; i<n; i++) {
        scanf("%d", &a[i]);
    }
    printf("Enter element to be deleted: ");
    scanf("%d", &no);
    for(i = 0; i < n; i++){
        if(a[i] == no){
            for(k = i; k < n - 1; k++){
            a[k] = a[k+1];
            }
        }
    }
    for(i = 0; i < n - 1; i++){
        printf("%d",a[i]);
    }
    return 0;
}
// LINEAR SEARCH AND BINARY SEARCH
```

```
// EXAMPLE  OF LINEAR SEARCH(when size of array is small so we use linear search)

#include<stdio.h>

int main () {
    int a[100], i, n, s;
    printf("Enter no of elements\n");
    scanf("%d", &n);
    printf("Enter elements to array\n");
    for(i = 0; i < n; i++){
        scanf("%d", &a[i]);
    }
    printf("Enter the element to search\n");
    scanf("%d", &s);
    for(i = 0; i< n; i++){
        if(a[i] == s){
            printf("Element found at index %d",i);
            break;
        }
    }
```

```
    if(i == n) {
        printf("Element not found\n");
    }
    return 0;
}
```

// EXAMPLE OF BINARY SEARCH(pre-request cdn -->array should sorted in ascending order)

```
#include<stdio.h>
int main () {
    int a[100], i, n, m, l, f, s;
    printf("Enter no of elements:\n");
    scanf("%d", &n);
    printf("Enter elements to array\n");
    for(i = 0; i < n; i++){
        scanf("%d", &a[i]);
    }
    printf("Enter element to be searched\n");
    scanf("%d", &s);
```

```
f = 0;
l = n - 1;
m = (f + l)/2;
while(f <= l){
    if(a[m] == s){
        printf("element found at index %d", m);
        break;
    }
    else if(a[m] < s){
        f = m + 1;
    }
    else {
        l = m - 1;
    }
    m = (f + l)/2;
}
if(f > l){
    printf("Element Not found\n");
}
```

```
    return 0;
}
// // for 2D arrays
// for(i = 0; I < 3; i++){
//     for(j = 0; j < 3; j++){
//     scanf(“%d”, &a[i][j])
//     }
//     }
```

// wap to input elements, rows and column matrix and display matrix on screen.

```
#include<stdio.h>
int main () {
    int a[100][100], i, j, r, c;
    printf("Enter elements to matrix: ");
    scanf("%d %d", &r, &c);
    for(i = 0; i < r; i++){
        for(j = 0; j < c; j++){
            scanf("%d", &a[i][j]);
```

```
        }
    }
    printf("Matrix entered is\n");
    for(i = 0; i < r; i++){
        for(j = 0; j < c; j++){
            printf("%d\t", a[i][j]);
        }
        printf("\n");
    }
    return 0;
}

//wap for matrix addition
#include<stdio.h>
int main () {
    int a[3][3], b[3][3],c[3][3], i, j;
    printf("Enter elements to 1st matrix: \n");
    for(i = 0; i < 3; i++){
        for(j = 0; j<3; j++){
```

```
        scanf("%d", &a[i][j]);
    }
}
printf("Enter elements to 2nd matrix\n");
for(i = 0; i < 3; i++){
    for(j = 0; j<3; j++){
        scanf("%d", &b[i][j]);
    }
}
for(i = 0; i < 3; i++){
    for(j = 0; j<3; j++){
        c[i][j] = a[i][j] + b[i][j];
    }
}
printf("resultant matrix adter addition is\n");
for(i = 0; i <3; i++){
    for(j = 0; j < 3; j++){
        printf("%d\t", c[i][j]);
    }
```

```
        printf("\n");
    }
    return 0;
}

// wap to sum all the elements of a matrix.(3 * 3)
#include <stdio.h>
int main()
{
    int a[3][3],i,j,add=0;
    printf("Enter elements : \n");
    for(i=0;i<3;i++)
    {
        for(j=0;j<3;j++)
        {
            scanf("%d",&a[i][j]);
        }
    }
    printf("SUM OF Matrix entered is \n");
```

```
    for(i=0;i<3;i++)
    {
        for(j=0;j<3;j++)
        {
            add = add + a[i][j];
        }
    }printf("%d",add);
 return 0;
}
```

// wap for matrix multiplication in 2D arrays.

```
#include<stdio.h>
int main() {
    int a[3][3], b[3][3], c[3][3], i, j, k;
    printf("Enter elements to 1st matrix: ");
    for(i = 0; i < 3; i++){
        for(j = 0; j < 3; j++){
            scanf("%d", &a[i][j]);
        }
```

```
}
printf("enter elements to 2nd matrix: ");
for(i = 0; i < 3; i++){
    for(j = 0; j < 3; j++){
        scanf("%d", &b[i][j]);
    }
}
for(i = 0; i < 3; i++){
    for(j = 0; j < 3; j++){
        c[i][j] = 0;
        for(k = 0; k < 3; k++){
            c[i][j] = c[i][j] + a[i][j] * b[i][j];
        }
    }
}
printf("resultant matrix is\n ");
for(i = 0; i < 3; i++){
    for(j = 0; j < 3; j++){
        printf("%d\t", c[i][j]);
```

```
        }
        printf("\n");
    }
}
// 6. wap to print multiplication of a matrix.
#include <stdio.h>

int main() {
    int a[10][10], b[10][10], c[10][10];
    int r1, c1, r2, c2, i, j, k;

    printf("Enter no. of rows and columns for 1st matrix: ");
    scanf("%d %d", &r1, &c1);
    printf("Enter no. of rows and columns for 2nd matrix: ");
    scanf("%d %d", &r2, &c2);

    if (c1 == r2) {
```

```
printf("Enter elements for 1st matrix:\n");
for (i = 0; i < r1; i++) {
    for (j = 0; j < c1; j++) {
        scanf("%d", &a[i][j]);
    }
}

printf("Enter elements for 2nd matrix:\n");
for (i = 0; i < r2; i++) {
    for (j = 0; j < c2; j++) {
        scanf("%d", &b[i][j]);
    }
}

// Matrix multiplication
for (i = 0; i < r1; i++) {
    for (j = 0; j < c2; j++) {
        c[i][j] = 0;
        for (k = 0; k < c1; k++) {
```

```
                c[i][j] += a[i][k] * b[k][j];
            }
        }
    }

    printf("Resultant matrix is:\n");
    for (i = 0; i < r1; i++) {
        for (j = 0; j < c2; j++) {
            printf("%d\t", c[i][j]);
        }
        printf("\n");
    }
} else {
    printf("Matrix multiplication is not possible. Columns of 1st matrix must equal rows of 2nd matrix.\n");
}
return 0;
}
```

```
// wap to calculate and print transpose of a matrix
#include <stdio.h>
int main() {
    int a[10][10], transpose[10][10];
    int r, c, i, j;

    printf("Enter rows and columns of the matrix: ");
    scanf("%d %d", &r, &c);

    printf("Enter elements of the matrix:\n");
    for (i = 0; i < r; i++) {
        for (j = 0; j < c; j++) {
            scanf("%d", &a[i][j]);
        }
    }
    for (i = 0; i < r; i++) {
        for (j = 0; j < c; j++) {
```

```
            transpose[j][i] = a[i][j];
        }
    }

    printf("Transpose of the matrix is:\n");
    for (i = 0; i < c; i++) {
        for (j = 0; j < r; j++) {
            printf("%d\t", transpose[i][j]);
        }
        printf("\n");
    }

    return 0;
}

// WAP to add all diagonal elements of a matrix
#include <stdio.h>

int main() {
```

```
    int a[10][10], r, c, i, j, sum = 0;

    printf("Enter rows and columns of the matrix: ");
    scanf("%d %d", &r, &c);

    if (r != c) {
        printf("Matrix is not square, diagonals not possible.\n");
        return 0;
    }

    printf("Enter elements of the matrix:\n");
    for (i = 0; i < r; i++) {
        for (j = 0; j < c; j++) {
            scanf("%d", &a[i][j]);
        }
    }

    for (i = 0; i < r; i++) {
```

```
        sum += a[i][i];  // Adding main diagonal elements
    }

    printf("Sum of diagonal elements is: %d\n", sum);
    return 0;
}
```

// wap to display elements above and below the diagonal of a matrix.

```
#include <stdio.h>

int main() {
    int a[10][10], r, c, i, j;

    printf("Enter rows and columns of the matrix: ");
    scanf("%d %d", &r, &c);
```

```
    if (r != c) {
        printf("Matrix is not square, diagonals not possible.\n");
        return 0;
    }

    printf("Enter elements of the matrix:\n");
    for (i = 0; i < r; i++) {
        for (j = 0; j < c; j++) {
            scanf("%d", &a[i][j]);
        }
    }

    printf("Elements above the diagonal are:\n");
    for (i = 0; i < r; i++) {
        for (j = 0; j < c; j++) {
            if (j > i) {
                printf("%d ", a[i][j]);
            }
```

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

    printf("Elements below the diagonal are:\n");
    for (i = 0; i < r; i++) {
        for (j = 0; j < c; j++) {
            if (i > j) {
                printf("%d ", a[i][j]);
            }
        }
    }
    printf("\n");

    return 0;
}
```

// WAP to check whether 2 matrices are identical or not

```
#include <stdio.h>

int main() {
    int a[10][10], b[10][10], r, c, i, j, flag = 1;

    printf("Enter rows and columns of the matrices:
");
    scanf("%d %d", &r, &c);

    printf("Enter elements of the first matrix:\n");
    for (i = 0; i < r; i++) {
        for (j = 0; j < c; j++) {
            scanf("%d", &a[i][j]);
        }
    }

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

```
            scanf("%d", &b[i][j]);
        }
    }

    for (i = 0; i < r; i++) {
        for (j = 0; j < c; j++) {
            if (a[i][j] != b[i][j]) {
                flag = 0;
                break;
            }
        }
        if (flag == 0)
            break;
    }

    if (flag == 1)
        printf("Matrices are identical.\n");
    else
        printf("Matrices are not identical.\n");
```

```
    return 0;
}

// wap to add opposite diagonal elements of a matrix.
#include <stdio.h>

int main() {
    int a[10][10], r, c, i, sum = 0;

    printf("Enter rows and columns of the matrix: ");
    scanf("%d %d", &r, &c);

    if (r != c) {
        printf("Matrix is not square, opposite diagonal not possible.\n");
        return 0;
    }
```

```
    printf("Enter elements of the matrix:\n");
    for (i = 0; i < r; i++) {
        for (int j = 0; j < c; j++) {
            scanf("%d", &a[i][j]);
        }
    }
    for (i = 0; i < r; i++) {
        sum += a[i][r - i - 1];  // Opposite diagonal element
    }

    printf("Sum of opposite diagonal elements is: %d\n", sum);
    return 0;
}

// Enter rows and columns of the matrix: 3 3
// Enter elements of the matrix:
```

```
// 1 2 3

// 4 5 6

// 7 8 9

// Sum of opposite diagonal elements is: 15


// wap to enter elements to a 1D array and display them using a functions.

#include<stdio.h>

void display( int [], int);

int main () {

    int a[100], i, n;

    printf("Enter no. of elements\n");

    scanf("%d", &n);

    printf("Enter elements to array\n");

    for(i = 0; i < n; i++){

        scanf("%d", &a[i]);

    }

    display(a, n);

    return 0;
```

```
}

void display(int b[], int n1)
{
    for(int i = 0; i < n1;  i++){
        printf("%d", b[i]);
    }
}

// passing 2D array/ matrix to a function
#include<stdio.h>
void display(int [3][3]);
int main()
{
    int a[3][3], i, j;
printf("Enter elements to matrix\n");
for(i = 0; i < 3; i++){
    for(j = 0; j < 3 ; j++){
        scanf("%d", &a[i][j]);
```

```
    }
}
display(a);
return 0;
}

void display(int b[3][3])
{
    int i, j;
    for(i = 0; i < 3; i++){
        for(j = 0; j < 3; j++){
        printf("%d\t", b[i][j]);
        }
        printf("\n");
    }
}

// size is known
#include<stdio.h>
```

```
void display(int [10][10], int , int);
int main () {
    int a[10][10], i, j, r, c;
    printf("Enter rows and columns: ");
    scanf("%d %d", &r, &c);
    printf("Enter elements\n");
    for(i = 0; i < r; i++){
        for(j = 0; j < c; j++){
            scanf("%d", &a[i][j]);
        }
    }
    display(a, r, c);
    return 0;
}

void display(int b[10][10], int x, int y){
    int i, j;
    for(i = 0; i < x; i++){
        for(j = 0; j < y; j++){
```

```
            printf("%d\t", b[i][j]);
        }
        printf("\n");
    }
}
```

// POINTERS START

//wap to illustrate the use of operator to access the value pointed by pointers

```
#include<stdio.h>
int main() {
    int x,y;
    int *ptr;
    x = 10;
    ptr = &x; // and -- address ; * for their value(get)
    /* ptr -- address of x
    *ptr value of x
    &ptr address of pointer
```

```
    */
    y = *ptr;
    printf("value of x is %d \n", x);
    printf("%d is stored at %u\n", *&x, &x);
    printf("%d is stored at %u\n", x, &x);
    printf("%d is stored at %u\n", *ptr, ptr);
    printf("%u is stored at %u\n", ptr, &ptr);
    *ptr = 25;
    printf("Now x = %d\n", x);
    return 0;
}

#include<stdio.h>
int main () {
    int x, *p1, **p2, ***p3;
    // pointer to a variable :--> *
    // pointer to a pointer :--> **
    // pointer to a another pointer :--> ***
    x = 100;
```

```
    p1 = &x;
    p2 = &p1;
    p3 = &p2;
    printf("%d\n", x);
    printf("%d\n", *p1);
    printf("%d\n", **p2);
    printf("%d\n", ***p3);
    return 0;
}
/* Pointer expressions:
```

>> like other variables , pointers can be also used in expressions.

>> eg: if p1 & p2 are properly declared and initialized pointers then statements like :

```
Y = (*p1) * (*p2);
sum = sum + (*p1); are valid
```

2. if a pointer stores memory address 1000 we add 1 to the pointer

p = 1000;

p = p + 1; (scaling)

Result depends on the type of pointer

if we add 1 to a pointer of type integer then result will be 1002

if we add 1 to a pointer of type float then result will be 1004

if we add 1 to a pointer of type char then result will be 1001

3. why 2 pointer can not be added, multiplied or divided but can only be subtracted

ans :-- since addition , multiplication and division of two memory address does not specify anything so it is not allowed by the compiler where as subtraction can be useful

if p1 & p2 are 2 pointers pointing to 1st and last element of an 1D array then p2 - p1 gives you total no. of elements between p1 and p2.

```
*/
#include<stdio.h>
int main () {
    int a,b,x,y,*p1, *p2;
    a = 12;
    b = 4;
    p1 = &a;
    p2 = &b;
    x = (*p1) * (*p2) - 6;
    y = 4 * - ((*p2)/(*p1)) + 10;
    printf("%d\n", x);
    printf("%d", y);
    return 0;
}


/*
types of pointers (depending on the state of pointer which is define by the operations we performed pointer can be categorised into 4 categories)
```

1.Null pointer :-- a pointer which is assgned value null is called a null pointer

ex:-- int *p = Null;

it is used for implementing a linked list data structures

2.Generic/void :-- in cases when we do not know the exact data type of variable, then we can use void or generic pointer

* derefrencing using cast operator

```
#include<stdio.h>

int main () {

    int a = 10;

    float b = 3.5;

    void *ptr;

    ptr = & a;

    printf("%d\n",*(int *)ptr);

    ptr = &b;

    printf("%f",*(float *)ptr);

    return 0;
```

}

3.wild :-- a pointer which is declared but never initialised in the program and thus it holds junk value.

4.dangling pointer:-- dangling pointer is a pointer that does not point to a valid object/ variable

in short pointer pointing to a non - valid memory location is called a dangling pointer

soln :-- null value should be assigned to the pointer so as it does not become a dangling pointer

*/

```
#include<stdio.h>

int main () {

    int a = 10;

    float b = 3.5;

    void *ptr;

    ptr = & a;

    printf("%d\n",*(int *)ptr);

    ptr = &b;
```

```
    printf("%f",*(float *)ptr);
    return 0;
}

#include<stdio.h>
int main() {
    char x[] = "a1b2c3d4e5f6g7h8i9j0";
    int t = 0;
    for(t = 1; x[t] != 0 && t <= strlen(x); t = t + 2){
        printf("%c", x[t]);
    }
    return 0;
}

// Arrays and pointers

/*
1. pointer to an array:--
```

2. array of pointer (instead of values we have memory address)

*/

// wap to create a poiner to 1D array and access the array elements using pointer

```
#include<stdio.h>

int main () {
    int a[5] = {10, 20, 30, 40, 50}, i = 0;
    int *ptr;
    ptr = &a[0];
    while(i < 5){
        printf("%d\n",*ptr);
        ptr++; //means ptr++ badhao address update hoga then *ptr se address ki value pick up kr lo
        i++;
    }
    return 0;
}
```

```
// int a[5] --> array of values
// int *a[5] --> array of pointers



// array of pointers
#include<stdio.h>
int main () {
    int a[10], *b[10], i, n;
    printf("Enter no. of elements\n");
    scanf("%d", &n);
    printf("Enter elements to array\n");
    for(i = 0; i < n; i++)
    {
        scanf("%d", &a[i]);
    }
    for(i = 0; i < n; i++){
        b[i] = &a[i];
    }
```

```
    for(i = 0; i < n; i++){
        printf("%d\n", *b[i]);
    }
    return 0;
}
```

// Dynamic Memory allocation:--

/*

C language requires the size of array to be provided at compile time.

which may cause 2 problems:--

1. wastage of memory:-- in case size required is les than the reserved memory.

2.Program failure :-- in case size required is more than the reserved memory.

In C following 4 forms will help for DMA

1. malloc() :-- memory allocation

2.Calloc()   :-- contigous allocation

3.realloc() :-- reallocation

4.free() :-- to clear the used memory

1. malloc() :-- memory allocation

defn :-- this function allocates / requested amount of memory in bytes and returns the pointers to the first byte of allocated memory is called malloc()

only take 1 argument does not sets all block 0.

ptr = (void *) malloc (size in bytes);

*/

// wap to input marks of n student and calculate average marks

```
#include<stdio.h>
int main() {
    int n, *ptr, sum = 0, avg, i;
    printf("Enter no of students\n");
    scanf("%d", &n);
    ptr = (int *) malloc(n * sizeof(int));
    printf("Enter marks\n");
    for(i = 0; i < n; i++) {
```

```
        scanf("%d", &*(ptr + i));
        sum = sum + *(ptr + i);
    }
    avg = sum / n;
    printf("Average marks are %d", avg);
    return 0;
}
```

// wap to input marks of n student and calculate average marks

```
#include<stdio.h>
#include<stdlib.h>
int main() {
    int n, *p, i;
    printf("Enter no of int\n");
    scanf("%d", &n);
    p = (int *) malloc(n * sizeof(int));
```

```
    if(p == NULL){
        printf("Memory not available\n");
        exit(1);
    }
    else {
        printf("Memory allocation was sucsessful");
        printf("\nEnter intger values: ");
        for(i = 0; i < n; i++){
            scanf("%d",p + i );
        }
        for(i = 0; i < n; i++){
            printf("\n%d",*(p + i));
        }
    }
    return 0;
}

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

```
int main() {
    int n, *p, i;
    printf("Enter no of int\n");
    scanf("%d", &n);
    p = (int*)calloc(n,sizeof(int));
    if(p == NULL){
        printf("Memory not available\n");
        exit(1);
    }
    else {
        printf("Memory allocation was sucsessful");
        printf("\nEnter intger values: ");
        for(i = 0; i < n; i++){
            scanf("%d",p + i );
        }
        for(i = 0; i < n; i++){
            printf("\n%d",*(p + i));
        }
    }
```

```
    return 0;
}
/*
calloc() :-- contigous allocation
calloc allocates memory for multiple blocks, each of same size and also sets all bytes to 0.
syntax:--
ptr = (void *) calloc(no of blocks, size of each block);
example :--
ptr = (int *) calloc(3, 4);
 */

#include<stdlib.h>
#include<stdio.h>
 int main () {
     int i, n, *ptr;
     printf("Enter no of elements\n");
     scanf("%d", &n);
```

```
    ptr = (int *) calloc(n, sizeof(int));
    printf("Enter elements\n");
    for(i = 0; i < n; i++){
        scanf("%d", &*(ptr + i));
    }
    printf("You have entered\n");
    for(i = 0; i < n; i++){
        printf("%d", *(ptr + i));
    }
    return 0;
}

// difference between calloc and malloc()

/*
calloc:--
1. function :-- allocates momory for n block each of same size
2. no. of arguments :-- 2
```

3. ptr = (void *) calloc(n, size);

4. contents of allocated memory:-- contains initialized to zero.

5.Speed :-- slower due to extra step of initialization to 0.

malloc:--

1. function :-- allocates "size in bytes" of memory

2. no. of arguments :-- 1

3. ptr = (void *) malloc(size);

4. contents of allocated memory:-- contents not itialized to zero.

5.Speed :-- faster than calloc().

*/

/*

3.realloc():-- reallocation

synatx: ptr = (void *) realloc(ptr, newsize);

```
note:-- ptr = realloc(ptr, 0);


same as free(ptr);
*/
#include<stdlib.h>
#include<stdio.h>
int main () {
    int i, n, *ptr;
    ptr = (int *) calloc(5, sizeof(int));
    *ptr = 10;
    *(ptr + 1) = 20;
    *(ptr + 2) = 30;
    *(ptr + 3) = 40;
    *(ptr + 4) = 50;
    printf("Now allocating more memory");
    ptr = (int *) realloc(ptr, 7 );
    *(ptr + 5) = 60;
    *(ptr + 6) = 70;
    for(i = 0; i < 7; i++){
```

```
        printf("%d\n",*(ptr + i));
    }
    free(ptr);
    return 0;
}

#include<string.h>
#include<stdlib.h>
#include<stdio.h>
int main () {
    char *ptr;
    ptr = (char *) malloc(20);
    printf("Enter a string\n");
    gets(ptr);
    printf("String = %s, address = %u\n", ptr, &ptr);
    ptr = (char *) realloc(ptr, 25);
    strcat(ptr," university");
    printf("string = %s, address = %u\n", ptr, &ptr);
    free(ptr);
```

```
    return 0;
}


//Memory Leak ?
// occurs when a computer program consumes memory, but is unable to release it back to the operating system
//forgot to use free(ptr);
// hume jb bhi DMA use krna ho to free(ptr) use krna hi h


#include <stdio.h>
#include <string.h>
int main (){
    char a[20], b[20];
    printf("Enter a string: ");
    gets(a);
    strcpy(b, a);
    puts(b);
    return 0;
```

```
}


// wap to copy 1 string to another without using strcpy()
#include<stdio.h>
#include<string.h>
int main () {
    char a[20], b[20];
    int len, i;
    printf("Enter a string: ");
    gets(a);
    len = strlen(a);
    for(i = 0; a[i] != '\0'; i++){
        b[i] = a[i];
    }
    b[i] = '\0';
    puts(b);
    return 0;
}
```

```
// wap for reverse string using str rev
#include<stdio.h>
#include<string.h>
int main () {
    char a[10];
    printf("Enter a string: ");
    gets(a);
    strrev(a);
    puts(a);
    return 0;
}

// wap for reverse string without using str rev
#include<stdio.h>
#include<string.h>
int main () {
    char a[10];
    int i, len;
```

```
    printf("Enter a string: ");
    gets(a);
    len = strlen(a);
    for(i = len - 1; i >= 0; i--){
        printf("%c", a[i]);
    }
    return 0;
}

// strcmp() --> string compare
#include<stdio.h>
#include<string.h>
int main () {
    char a[20], b[20];
    printf("Enter 1st string: ");
    gets(a);
    printf("Enter 2nd string: ");
    gets(b);
    if(stricmp(a, b) == 0){
```

```
    printf("Both strings are equal");
    }
    else if (stricmp(a, b) > 0)
    printf("String 1 is greater.");
    else
    printf("String 2 is greater");
    return 0;
}

// other variants

//1. stricmp() --> ignores upper and lower case
//2. strncmp() --> compares 1st n elements of 2 strings.
//3. strnicmp() --> combination of above 2.

// wap for checking substring
#include<stdio.h>
#include<string.h>
```

```
int main () {
    char a[20], b[20];
    printf("Enter 1st string: ");
    gets(a);
    printf("Enter 2nd string: ");
    gets(b);
    if(strstr(a, b) == '\0'){
    printf("substring not found.");
    }
    else
    printf("substring found.");
    return 0;
}

//7.strlwr():--converts upper case to lower case
#include<stdio.h>
#include<string.h>
int main () {
    char a[20];
```

```
    printf("Enter an upper case string");
    gets(a);
    strlwr(a);
    puts(a);
    return 0;
}

//8.strupr():--converts lower case to upper case
#include<stdio.h>
#include<string.h>
int main () {
    char a[20];
    printf("Enter an upper case string: ");
    gets(a);
    strupr(a);
    puts(a);
    return 0;
}
```

```
// wap to convert all lower ccase characters of string to upper case and vice versa

#include<stdio.h>

#include<string.h>

int main () {

    char a[20];

    int len, i;

    printf("Enter a string with both upper and lower case: ");

    gets(a);

    len = strlen(a);

    for(i = 0; i < len; i++){

        if(a[i] >= 65 && a[i] <= 90){

            a[i] = a[i] + 32;

        }

    }

    for(i = 0; i < len; i++){

        if(a[i] >= 97 && a[i] <= 122){

            a[i] = a[i] - 32;
```

```
        }
    }
    puts(a);
    return 0;
}

// wap to know about the palimdorme string.
#include<stdio.h>
#include<string.h>
int main () {
    char a[20], b[20];
    printf("Enter a string: ");
    gets(a);
    strcpy(b, a);
    strrev(a);
        if(strcmp(a, b) == 0){
            printf("Palimdrome string");
        }
        else{
```

```
            printf("Not palimdrome");
        }
    return 0;
}

// wap for check palimdrome string
#include <stdio.h>
#include <string.h>

int main() {
    char a[100], b[100];
    int i, x = 0, flag = 1;
    printf("Enter a string: ");
    gets(a);
    int len = strlen(a);
    for(i = len - 1; i >= 0; i--) {
        b[x++] = a[i];
    }
    b[x] = '\0';
```

```
    for(i = 0; i < len; i++) {
        if(a[i] != b[i]) {
            flag = 0;
            break;
        }
    }
    if(flag == 1)
        printf("It's a palindrome.\n");
    else
        printf("Not a palindrome.\n");
    return 0;
}

#include<Stdio.h>
int main () {
    char a[10] = "PROGRAM";
    int i, j;
    for(i = 0; i < 7; i++){
        for(j = 0; j <= i; j++){
```

```
            printf("%d", a[j]);
        }
        printf("\n");
    }
}
/*
P
PR
PRO
PROG
PROGR
PROGRA
PROGRAM
*/

// strset() --> sets all elements/characters of a string to a particular symbol.
//strnset() --> only 1st n characters.
#include<stdio.h>
```

```
int main () {
    char a[10] = "56743";
    char symbol = 'd';
    strset(a, symbol);
    puts(a);
    return 0;
}
```

// strset() --> sets all elements/characters of a string to a particular symbol.

//strnset() --> only 1st n characters.

```
#include<stdio.h>
#include<string.h>
int main () {
    char a[20] = "abcdefghijklmnop";
    char symbol = '*';
    strnset(a, symbol, 5);
    puts(a);
    return 0;
```

```
}

// a to i () --> convert string to integer.
// a to f()  --> convert string to float.
// a to l () --> convert string to long.

#include<stdio.h>
#include<stdlib.h>
int main() {
    char a[10] = "100.5";
    float val;
    val = atof(a);
    printf("%f", val);
    return 0;
}

// a to i () --> convert string to integer.
// a to f()  --> convert string to float.
// a to l () --> convert string to long.
```

```
#include<stdio.h>
#include<stdlib.h>
int main() {
    char a[10] = "100000";
    long int val;
    val = atol(a);
    printf("%ld", val);
    return 0;
}
```

//wap to enter a string and count no. of upper case, lower case, spaces and digits/special characters in it.

```
#include<stdio.h>
#include<string.h>
int main () {
    char a[100];
    int len, aa = 0,i, b = 0, c = 0, d = 0;
```

```
printf("Enter a string: ");
gets(a);
len = strlen(a);
for(i = 0; i < len; i++){
    if(a[i] >= 65 && a[i] <= 90){
        aa++;
    }
    else if(a[i] >= 97 && a[i] <= 122){
        b++;
    }
    else if(a[i] == 32){
        c++;
    }
    else {
        d++;
    }
}
    printf("No of uppercase letters %d\n", aa);
```

```
        printf("No of lowercase letters %d\n", b);

        printf("No of spaces %d\n",c);

        printf("No of digits %d\n", d);
    }
```

// wap to enter a string and sort in ascending/ descending order.

```
#include<stdio.h>
#include<string.h>
int main () {
    char a[100];
    int i, j, len, temp;
    printf("Enter a string: ");
    gets(a);
    len = strlen(a); // calculate the length.
    for(i = 0; i < len; i++){
        for(j = i + 1; j < len; j++){
```

```
            if(a[i] > a[j]){   // for ascending order
                temp = a[i];
                a[i] = a[j];
                a[j] = temp;
            }
        }
    }
    puts(a);
    return 0;
}
```

// wap to enter a string and sort in ascending/ descending order.

```
#include<stdio.h>
#include<string.h>
int main () {
    char a[100];
    int i, j, len, temp;
```

```
    printf("Enter a string: ");
    gets(a);
    len = strlen(a); // calculate the length.
    for(i = 0; i < len; i++){
        for(j = i + 1; j < len; j++){
            if(a[i] < a[j]){  // for descending order
                temp = a[i];
                a[i] = a[j];
                a[j] = temp;
            }
        }
    }
    puts(a);
    return 0;
}
```

// wap to check whether two strings are identical or not.

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

```
#include <string.h>

int main() {
    char a[100], b[100];
    int ctr = 0, i;

    printf("Enter 1st string: ");
    gets(a);

    printf("Enter 2nd string: ");
    gets(b);

    strlwr(a);
    strlwr(b);

    if (strlen(a) != strlen(b)) {
        printf("Not identical.");
    } else {
        for (i = 0; i < strlen(a); i++) {
```

```
            if (a[i] == b[i]) {
                ctr++;
            } else {
                break;
            }
        }

        if (ctr == strlen(a)) {
            printf("Identical strings.");
        } else {
            printf("Not identical.");
        }
    }

    return 0;
}
```

// Structure(user-defined data type) {OOPS, classes file handling related}

/*

structure :-- complex data organisation

(it is storing multi data types variables so here structure help)

every struct ends with a semicolon(;)

defn:-- structure is a user defined data-type that can store this similar type of data contiguous in memory.

struct<student>

{

};

note:-- '.' dot operator is used to i/p or o/p data from structure.

*/

```
#include<stdio.h>
struct student // any variable name
{
    char name[20];
    int roll;
    float marks;
} s1;

// or

#include<stdio.h>
struct student // any variable name
{
    char name[20];
    int roll;
    float marks;
};

int main () {
```

```
    struct student s1;
}



#include<stdio.h>
struct student // any variable name
{
    char name[20];
    int roll;
    float marks;
} s1;

int main() {
    printf("enter name: ");
    gets(s1.name);
    printf("Enter roll.no: ");
    scanf("%d", &s1.roll);
    printf("Enter marks: ");
```

```
    scanf("%f", &s1.marks);
    printf("Details of student are: \n");
    puts(s1.name);
    printf("%d\n", s1.roll);
    printf("%f", s1.marks);
    return 0;
}


// multiple structure variable of same type
#include<stdio.h>
struct simple // any variable name
{
    char ch;
    int num;
};

// always assume any code with their memory address.
```

```
int main() {
    struct simple a;
    struct simple b;
    a.num = 2;
    a.ch = 'x';
    b.num = 2;
    b.ch = 'y';
    printf("%d%c\n",a.num, a.ch);
    printf("%d%c", b.num,b.ch);
    return 0;
}

// initialization of structure
#include<stdio.h>
struct data {
   int no;
   float salary
};
```

```
int main () {
   struct data emp1 = {123, 35595.6};
   struct data emp2 = {345, 40756.7};
   printf("List of employees\n");
   printf("Employee1 no is %d\n", emp1.no);
   printf("Employee1 salary is %f\n", emp1.salary);
   printf("Employee2 no is %d\n", emp2.no);
   printf("Employee2 salary is %f", emp2.salary);
   return 0;
}

// Array of structure(important)

 #include<stdio.h>
 struct student // any variable name
 {
     char name[20];
     int roll;
     float marks;
```

```
};

int main() {
   struct student s[100];
   int i, n;
    printf("enter no. of student: ");
    scanf("%d", &n);

    for (i = 0; i < n; i++){
    printf("enter name: \n");
    fflush(stdin);
    gets(s[i].name);
    printf("Enter roll.no: \n");
    scanf("%d", &s[i].roll);
    printf("Enter marks: ");
    scanf("%f", &s[i].marks);
    }
    for(i = 0; i < n; i++){
       printf("output is: \n");
```

```
        puts(s[i].name);
        printf("%d\n", s[i].roll);
        printf("%f\n", s[i].marks);
    }
    return 0;
}

// nested structure(dob -- dd|mm|yyyy)

// // method 1
//  #include<stdio.h>
//  struct student // any variable name
// {
//      int roll;
//      char name[100];
//      float marks;
//      struct dob {
//          int dd;
//          int mm;
```

```
//         int yy;
//      }d1;
// }s1;


//  int main() {
//      printf("enter roll no: \n");
//      scanf("%d", &s1.roll);
//      printf("enter name: \n");
//      fflush(stdin);
//      gets(s1.name);
//      printf("enter marks: \n");
//      scanf("%f", &s1.marks);



//      printf("enter dob: \n");
//      scanf("%d%d%d",
&s1.d1.dd,&s1.d1.mm,&s1.d1.yy);
//      printf("roll no is %d\n", s1.roll);
//      puts(s1.name);
```

```
//      printf("marks are %f\n", s1.marks);
//      printf("dob is %d|%d|%d",
s1.d1.dd,s1.d1.mm,s1.d1.yy);
//      return 0;
//  }



// // // method 2 for this
// // struct dob{
// //     int dd;
// //     int mm;
// //     int yy;
// // };

// // struct student{
// //     int roll;
// //     char name[10];
// //     float marks;
```

// //   struct dob d1;

// // }s1;

// date : 15-04-2025

// 1) Passing structure as argument to a fxn

// i. call by value method

// ii. call by reference method.

// 2) pointer to structure

// 3) self referential structure

// 4)union

// 5)union vs structure

// 1) call by value method

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

// struct date{         //note: always write the struct first then function

//     int dd;

//     int mm;

//     int yy;

// };


// void display(struct date);


// int main(){

//     struct date a;

//     a.dd =15;

//     a.mm = 4;

//     a.yy = 2025;


//     display(a);

//     return 0;

// }
```

```
// void display(struct date b){
//     printf("%d %d %d", b.dd,b.mm,b.yy);
// }




// 2) call by reference


// #include <stdio.h>
// struct date{         //note: always write the struct first then function
//     int dd;
//     int mm;
//     int yy;
// };


// void display(struct date*);  //change 1
```

```
// int main(){
//     struct date a;
//     a.dd =15;
//     a.mm = 4;
//     a.yy = 2025;

//     display(&a);   //change 2
//     return 0;
// }

// void display(struct date *b){
//     printf("%d ", (*b).dd);
//     printf("%d ", (*b).mm);
//     printf("%d", (*b).yy);
// }

// #include <stdio.h>
// struct date{          //note: always write the struct first then function
```

```
//     int dd;
//     int mm;
//     int yy;
// };

// void display(struct date*);  //change 1

// int main(){
//     struct date a;
//     a.dd =15;
//     a.mm = 4;
//     a.yy = 2025;

//     display(&a);   //change 2
//     return 0;
// }

// void display(struct date *b){  //note : isme * and .
nhi use hoga print karte time
```

```
//     printf("%d ", b->dd);
//     printf("%d ", b->mm);
//     printf("%d", b->yy);
// }




// ---> pointer to structure


// #include <stdio.h>
// struct simple{          //note: always write the struct first then function
//     int a;
//     float b;
//     char c[20];
// };


// int main(){
```

```
//      struct simple s; //structure variable

//      struct simple *ptr; //ptr will hold the adrees of structure simple (declaring ptr to structure)

//      ptr = &s; //initialization of ptr

//      ptr -> a =5;

//      ptr -> b = 9.56;

//      scanf("%s",ptr -> c);

//      printf("%d\n", ptr ->a);

//      printf("%f\n", ptr ->b);

//      puts(ptr->c);

//      return 0;

// }


// #include <stdio.h>

// #include <string.h>

// struct simple{          //note: always write the struct first then function

//      int a;

//      float b;
```

```
//      char c[20];
// };


// int main(){
//      struct simple s; //structure variable
//      struct simple *ptr; //ptr will hold the adrees of structure simple (declaring ptr to structure)
//      ptr = &s; //initialization of ptr
//      ptr -> a =5;
//      ptr -> b = 9.56;
//      strcpy(ptr->c,"xyz");
//      printf("%d\n", ptr ->a);
//      printf("%f\n", ptr ->b);
//      puts(ptr->c);
//      return 0;
// }
```

// 3) self referal data

// #include<stdio.h>

// struct list

// {

// int value;

// struct list * next;

// }; //simply ye bus khali spaces dhoondta ha or unhe merge karta ha taki hum vha program run kar ske

// int main()

// {

// struct list n1,n2,n3,n4;

// struct list*ptr;

// ptr = &n1;

// n1.value = 100;

// n1.next = &n2;

// n2.value = 200;

```
//     n2.next = &n3;
//     n3.value = 300;
//     n3.next = &n4;
//     n4.value = 400;
//     n4.next=0;
//     while(ptr!=0)
//     {
//         printf("%d\n",ptr->value);
//         ptr = ptr->next;
//     }
//     return 0;
// }

// UNION:

// #include<stdio.h>
// union simple
// {
//     int a;
```

```
//     float b;
//     char c;
// }; //union mai memory kam use hoti ha but ek time par hum sirf 1 hi value access kar sakte ha like we can only print unt, float or char value at a time



// int main()
// {
//     union simple s;
//     s.a = 10;
//     s.b = 105;
//     s.c = 'y';
//     printf("%d\n%f\n%c",s.a,s.b,s.c);
//     return 0;
// }


// #include<stdio.h>
// union simple
```

```
// {
//     int a;
//     float b;
//     char c;
// };//union mai memory kam use hoti ha but ek time par hum sirf 1 hi value access kar sakte ha like we can only print unt, float or char value at a time


// int main()
// {
//     union simple s;
//     s.a = 10;
//     printf("%d\n",s.a);
//     s.b = 105;
//     printf("%f\n",s.b);
//     s.c = 'y';
//     printf("%c",s.c);
//     return 0;
// }
```

```
//pointer and union

#include<stdio.h>
union value{
    int i;
    float j;
};

int main () {
    union value a;
    union value *ptr;
    ptr = &a;
    ptr -> i = 10;   //  or  (*ptr).i = 10
    printf("%d\n", ptr ->i);
    ptr -> j = 15.5;
    printf("%f", ptr -> j);
    return 0;
}
```

```
//union of structure variables h but structure of union nhi h


#include<stdio.h>


struct student{
    char name[10];
    int roll;
};


struct employee {
    int id;
    float sal;
};
union simple {
    struct student a;
    struct employee b;
};
```

```
int main () {
    union simple s;
    printf("Enter name\n");
    gets(s.a.name);
    puts(s.a.name);
    printf("Enter roll no.\n");
    scanf("%d", &s.a.roll);
    printf("%d\n", s.a.roll);
    printf("Enter employee id\n");
    scanf("%d", &s.b.id);
    printf("%d\n", s.b.id);
    printf("Enter salary\n");
    scanf("%f", &s.b.sal);
    printf("%.f\n", s.b.sal);
    return 0;
}
```