i)
```
#include <stdio.h>
void fun (int *ptr)
{
  *ptr = 30;
}
int main()
{
  int y = 20;
  fun (&y);
  printf ("%d", y);
  return 0;
}
```
o/p: 30

↳ Call by reference

ii)
```
void fun (int x)
{
  x = 30;
}
int main()
{
  int y = 20;
  fun (y);
  printf ("%d", y);
  return 0;
}
```
o/P : 20

↳ Call by value

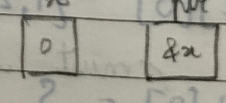
iii)
```
int main()
{
  float arr 5 = {12.5, 10.0, 13.5, 90.0, 0.5};
  float *ptr 1 = &arr[0];
  float *ptr 2 = ptr1 + 3;
  printf ("%f", *ptr 2);
  printf ("%d", ptr 2 - ptr 1);
  return 0;
}
```
o/P:- 90.0

3

2012 − 2000/4 = 3

↳ returns the no. of elements in b/w these 2 elements, addresses

iv)
```
int main() {
  int *ptr;
  int x;
  ptr = &x;
  *ptr = 0;        — *(&x) = x
  printf("x = %d", x);      0
  printf("*ptr = %d", *ptr);  0
  *ptr += 5;
  printf("x = %d", x);       5
  printf(*ptr = %d", *ptr);  5
  (*ptr)++;
  printf("x = %d", x);       6
  printf("*ptr = %d", *ptr);  6
```



# Practice Questions.

Q:1. Explain the declaration & initialization of multi-dimensional arrays in C.
(arrays of arrays)

Ans Declaration :

datatype arrayname [Size 1][Size 2];

- datatype: int, float..
- Size 1: no. of rows
- Size 2: no. of columns.

↳ int matrix[3][4];
This creates a 2D array with 3×4 = 12 integer elements.

Initialization :

```
datatype arrayname [Size 1][Size 2] = {
    {v1, v2, ..... vn},
    {v1, v2, ..... vn},
    ....
};
```

↳
```
int arr [2][3] = {
    {1, 2, 3},
    {4, 5, 6}
};
```

↳ int arr [2][3] = {1, 2, 3, 4, 5, 6};

↳ int arr [ ][3] = { (must specify column no.)
    {1, 2, 3},
    {4, 5, 6}
};

Q:2.

Q2. Derive and explain the formula for calculating the memory address of an element in a 2D array stored in C row-major order.

Given –

Let the 2D array be : $A[L_1 \ldots U_1][L_2 \ldots U_2]$

where,

$L_1$ = lower bound of row, $U_1$ = upper bound of row

$L_2$ = lower bound of column, $U_2$ = upper bound of column.

Let – BA = base address of the array.

(w) Size = size (in bytes) of each element.

i, j = indices of the element we want to access.

Row-major oder :

$$A[i][j] = BA + \frac{Size}{w} (i \times col + j)$$

i → row index; j → column index; col → total no. of columns in the array

Example : `int A[3][4];` (int → 4 bytes, 3 → row, 4 → column) want address of A[2][1] = ?

$$\text{Address} = 1000 + 4(2 \times 4 + 1)$$
$$= 1000 + 4 \times 9 = 1000 + 36 = 1036.$$

Q: 3. Explain the various methods of declaring and initializing strings in C. Discuss the concept of string initialization using arrays and pointers.

Ans A string is just a character array terminated with a special null character '\0'.

1. Declaring & Initializing Strings.

a) Using Char. array with String literal.

```
char str[] = "Hello";
```

automatically adds '\0' at the end.
size is determined by the compiler (6 bytes here).

b) Using Char. Array with Size.

```
char str[10] = "Hello";
```

c) Using Pointer to String Literal.

```
char *str = "Hello";
```

| Feature | Array Initialization | Pointer Initialization |
|---|---|---|
| Syntax | `char str[] = "Hello";` | `char *str = "Hello";` |
| Memory location | Stored in stack | Stored in read-only memory |
| Modifiability | Can modify | Can't modify |
| Null-terminated | yes | yes. |
| Flexibility | more | less |
| Risk of Crash | very low | High (if modified) |

Q: 4.

Ans

Q4. Describe the use of standard library functions for string manipulation in C.

Ans In C, strings are handled as arrays of characters terminated by the null char. Performing string operations manually is time-consuming & error-prone. Therefore the C lang. provides a collection of standard library func^n for string manipulation. It is defined in the <string.h> header file.
These functions allow programmers to perform operations like copying, comparing, concatenating & finding the length of strings efficiently.
In C, gets() & puts() are standard I/O func^n used for input & output of strings. They help in reading & displaying strings.

```
#include <stdio.h>
int main()
{
    char name[30];
    printf("Enter name: ");
    gets(name);
    printf("Name: ");
    puts(name);
    return 0;
}
```

Commonly used string functions:

↳ strlen: returns the length of the string str (excluding \0)

```
strlen("Hello")  // output 5
```

↳ strcpy: Copies the content of the source string into destination

```
char str[10];
strcpy(str, "Hi");  // str becomes "Hi"
```

↳ strcat (destination, source) : Appends the source string at the end of the destination ".

```
char str[20] = "Good";
strcat(str, "Morning"); // str becomes "Good Morning"
```

↳ strcmp (str1, str2) : Compares 2 strings lexicographically. It returns :

0, if both strings are Equal
<0, if str1 comes before str2.
>0, if str1 comes after str2.

```
strcmp("A", "B"); // Negative value
```

Q:5. Explain the basic concept of pointers in C. Include Examples to illustrate pointer declaration, initialization and usage.

Ans Pointer in C is a variable that holds the address of some other variable. It is called pointer because it points to a particular location in memory by storing the address of the location.
NULL pointer holds the value nthg.

Declaration : `int * ptr;`
It is declared using the * symbol before variable name

Initialization :

```
int num = 10;
int *ptr = &num; // 'ptr' now holds the address of 'num'.
```

Pointer needs to be initialized with a valid memory address. The '&' operator is used to get the address of a variable.

Q:6.

Ans

Pointer Usage : It is used to access & modify the value at that memory address they hold. The * operator is used to dereference the pointer accessing the value at that address.

Example code :

```
#include <stdio.h>
int main()
{
    int num = 25;
    int *p;
    p = &num;
    printf("Address of num: %p \n", p);
    printf("Value of num : %d", *p);
    return 0;
}
```

Pointers are useful in C because it -
↳ Directly access memory & manipulation.
↳ Efficient func$^n$ parameter passing (pass by reference).
↳ Dynamic memory allocation using malloc, calloc, etc.
↳ Handling arrays & strings effectively.

Q:6. Describe the relationship b/w pointers & arrays. Provide examples demonstrating pointer-based traversal & array indexing using pointers.

Ans In C, arrays & pointers are closely related. The name of an array acts as a pointer to its first element. This means that array elements can be accessed using either array indexing (arr[i]) or pointer arithmetic (*(arr+i)).

Ex 1. Array Indexing vs Pointer Notation

```
#include <stdio.h>
int main()
{
    int arr[5] = {10, 20, 30, 40, 50};
    printf("arr[2] = %d\n", arr[2]);
    printf("*(arr+2) = %d\n", *(arr+2));
    return 0;
}
```

Output : arr[2] = 30
*(arr+2) = 30.

Ex 2. Pointer-Based Array Traversal

```
#include <stdio.h>
int main()
{
    int arr[5] = {1,2,3,4,5};
    int *ptr = arr;
    printf("Traversing array using pointer:\n");
    for(int i=0; i<5; i++){
        printf("%d", *(ptr+i));
    }
    return 0;
}
```

Output : Traversing array using pointer:
1 2 3 4 5.

Q:7 WAP in C to accept 'n' no. from user, store these nos. into an array & count the no. of occurence of each no.

Ans

```
#include <stdio.h>
int main()
{
  int n, i, j, count;
  printf("Enter how many nos: ");
  scanf("%d", &n);
  int arr[n];
  printf("Enter %d no.: \n", n);
  for (i=0; i<n; i++)
  {
      scanf("%d", &arr[i]);
  }
  printf("\n freq. of each no.: \n");
  for (i=0; i<n; i++) {
      int iscounted = 0;
    for (j=0; j<i; j++) {
        if (arr[i] == arr[j]) {
            iscounted = 1;
            break;
        }
    }
    if (iscounted == 0) {
        count = 1;
      for (j=i+1; j<n; j++) {
          if (arr[i] == arr[j]) {
              count++;
          }
      }
      printf("%d occurs %d times\n", arr[i], count);
    }
}
```

Q:8 WAP to take the input into an array & reverse the element of the array.

Ans
```
#include <stdio.h>
int main()
{
  int n, i;
  printf("Enter the no. of elements :");
  scanf("%d", &n);
  int arr[n];
  printf("Enter %d element :\n", n);
  for (i=0; i<n; i++)
  {
      scanf("%d", &arr[i]);
  }
  for (i=0; i<n/2; i++) {
      int temp = arr[i];
      arr[i] = arr[n-1-i];
      arr[n-1-i] = temp;
  }
  printf("Reversed array:");
  for (i=0; i<n; i++) {
      printf("%d", arr[i]);
  }
  return 0;
}
```

Q:10.

Q:9. WAP to read 3*3 matrix & check sum of diagonal element is same or not.

Ans

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

```
int main () {
    int matrix [3][3];
    int i, j, Sum_primary = 0, Sum_secondary = 0;
    printf ("Enter elements of 3 x 3 matrix : ");
    for (i = 0; i < 3; i++) {
        for (j = 0; j < 3; j++) {
            scanf ("%d", &matrix [i][j]);
        }
    }
    for (i = 0; i < 3; i++) {
        Sum_primary += matrix [i][i];
    }
    for (i = 0; i < 3; i++) {
        Sum_secondary += matrix [i][3-1-i];
    }
    if (Sum_primary == Sum_secondary) {
        printf ("Sum of diagonal elements is same.\n");
    }
    else {
        printf ("Sum of diagonal elements is not same.");
    }
    return 0;
}
```

Q:10. WAP to accept 'n' names from user, store them into 2D array & Search whether a given name is present in array or not.

Ans

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

```
int main()
{
  int n, i, found = 0;
  char names[100][50];
  char searchName[50];
printf("Enter no. of names : ");
  scanf("%d", &n);
getchar();
  printf("Enter %d names : ", n);
for(i = 0; i < n; i++){
      printf("Name %d", i+1);
      gets(names[i]);
printf("Enter name to search : ");
gets(searchName);
  for(i=0;   i < n; i++){
      if(strcmp(names[i], searchName) == 0){
            found = 1;
            break;
      }
  }
if(found)
    printf("'%s' is present in the list.", searchName);
  else
    printf("'%s' is NOT ", searchName);
return 0;
}
```