(Unit-4)

Array :- An array is a collection of similar type of data elements, that are referenced by a common name.
one variable : Capable of storing multiple values.

imp. Characteristics of Array :-

*1. All Elements within an array must be of the same data type. This could be int, float, char, or any other valid C data type.

*2. All the elements of an array occupy a set of contiguous memory allocations, making it efficient for accessing and manipulating elements.

3. The size of an array is fixed at the time of declaration & can't be changed dynamically during program execution.

4. Each Element in an array is identified by a unique index, starting from 0.

5. C supports multidimensional arrays, which can be used to represent matrices or tables. Each index (dimension) has its own index.

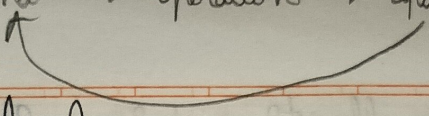
Ex - 1Dimensional Array. (Store elements in a single row)
int numbers[5] = {10, 20, 30, 40, 50};

↳ 2D Array. (Store elements in rows & columns).
int matrix[3][2] = {
{1, 2},
{3, 4},
{5, 6}
};

2 subscript

Order of Execution for iteration:

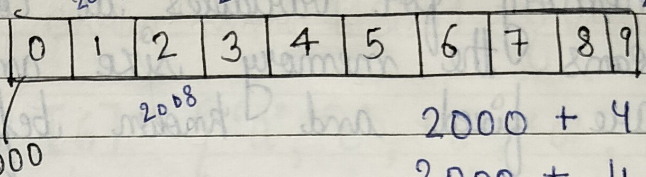
initialization → cond° → operation → updation



↳ Multidimensional Arrays

Can have more than two dimensions.

array :-



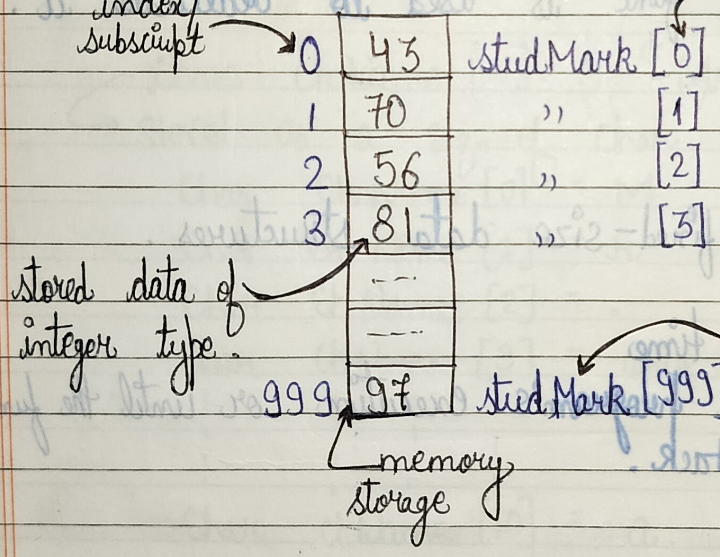
2000 + 4 * 2 = 2008

2000 + 4 * 3 = 2012

Base address + Data Type Size * No. of Element.

* Formula to Calculate :- the address of any element in array.

Variable declaration → memory allocation.



starting from 0 instead of 1.

Arrays are a powerful tool for organizing and managing data efficiently.

Array declaration

~~array_element~~ data_type array_name [array_size];

Eg. float, char, int, etc.

Static Memory allocation :- It is the type of allocation in which the Compiler allocates memory for variables at Compile time. This means the memory size and lifetime of these variables are fixed and known beforehand.

Dynamic Memory allocation :- It allows programmers to allocate memory at run time, providing more flexibility in managing memory usage.
The malloc() & Calloc() func" are used to allocate memory while the free() func" is used to deallocate it.

Static Allocation
↳ Simpler to use.
↳ More efficient for fixed-size data structures.
↳ Less flexible.
↳ Occurs at Compile time
↳ Exists throughout the program's execution or until the func" returns
↳ Usually on the stack.

Dynamic Allocation
↳ Can be less efficient due to overhead of memory allocation & deallocation
↳ More flexible for variable-sized data structures.
↳ Occurs at run time
↳ Exists until explicitly deallocated using free().
↳ Storage on the heap.

NOTE: Array index use indices from 0 to 999 to access the elements of the array.

Page No.
Date: 22 / 11 / 24

Array Initialization & declaration

data_type _ array_name [size] = { val 1, val 2, ... val n };

char chvowel [6] = { 'a', 'e', 'i', 'o', 'u', '\0' };

↓ memory allocate

Null character

after allocating, its our choice whether we place value or not.

String - Sequence of characters.

char chvowel [6] = "aeiou";

char ch Name [ ] = "Mr. Dracula";

↳ stored as a seq. of char.

Char ch Name [0] = M

Char ch Name [1] = r

Char ch Name [2] = .

Char ch Name [3] = white blank.

⋮ ⋮ ⋮ ⋮

Char ch Name [10] = a.

Taking 10 integer input from user & store then in an array & find the sum of all numbers stored in array.

```
#include <stdio.h>
int main()
int i, Sum 0, arr [10];
for(i=0; i<10; i++)
Scanf ("%d", &arr[i]);
for (i=0; i<10; i++)
Sum + = arr [i];
printf ("Sum of input integers
is %d \n", Sum);
return 0;
}
```

```
#include <stdio.h>
int main()
int i, j, x, arr[10];
printf("Enter 10 integer numbers");
for (i = 0; i<10; i++) {
   scanf("%d", &arr[i]);
}
for (i = 0; i<9; i++) {
   x = arr[i];
      assign
for (j = i+1; j<10; j++) {
   if (x == arr[j])
printf("%d no. appeared more than once", x);
   }
}
return 0;
}
```

2D Arrays.

↳ A two dimensional array has 2 subscripts/ indexes.
↳ 1st refers to the row & second refers to the column.

Declaration : data_type arrayname [1st dimension size][2nd D. size]

Eg., int xInteger [3][4];
float matrixNum [20][25];
row column.

We must know or declare that what type of approach we are using like row wise or column wise.

```
#include <stdio.h>
#define ROW 5
#define COL 4
int main (void) {
int i, j;
double total;
int marks [ROW][COL] = {10, 23, 31, 11, 20, 43, 21, 21 ----};
for (i=0 ; i<ROW ; i++) {
        total = 0.0;
        for (j=0 ; j<COL ; j++)
        total += marks[i][j];
printf("Average of student %d is %.f\n", i, total/4.0);
    }
}
```

For array storing string

```
char Name [6][10] = ["Mr. Bean", "Mr. Bush", "Nicole",
                     "Kidman", "Arnold", "Jarok"]
```

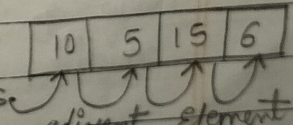
Searching Technique

Searching operation and sorting operation.

Sequential / linear search for searching the particular element in a given sequence. (no order req.)

We have to search the element linearly or one by one from the very first index i.e., 0

Binary search / Internal Search (order is req. (ascending or descending))



↓

Best Case where we find the element in least operations. Based on divide & Conquer mtd.

for (i → 0 to n-1)
if (key == a[i])

a[6] | 8 | 90 | 7 | 5 | 40 | 31 |
0 1 2 3 4 5

Key ⇒ 40

for i = 4
40 == 40.
→ 4 times comparison.

Ex:- a[] | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
0 1 2 3 4 5 6 7 8
Left, Mid, Right

↳ divide in two halves & search until element is found.
↳ for finding mid, use floor & Ceiling fun^{n} if no. is in decimal.
↳ Prerequisties Cond^{n}. The array must be sorted in either ascending or descending order.

middle element = (first + last)/2
0 + 8/2 = 4 mid term

while (first <= last)
middle < Element
first = middle + 1 (RHS)

last middle == Element
(Value of element is equal to middle element)

middle > Element
last = middle - 1 (LHS)

Element to be found 72

| 9 | 20 | 25 | 31 | 47 | 52 | 69 |
0 1 2 3 4 5 6
m-1, m+1
1st, 1st, mid, 1st, last
0 in (m-1) Cond^{n}
0 in (m+1) Cond^{n}

① ② ③ ④

Best case in Binary is when we find element at mid & worst will be when it is present at either end of it.

Sorting → Rearrangement of elements of an array in a specific or logical order, typically in ascending or descending order.

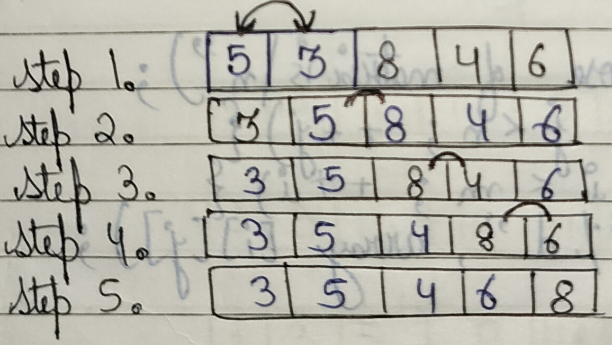
Common Sorting algorithm are as follows:

1. Bubble Sort.

↳ Each element is compared with its adjacent elements & swaps them if they are in the wrong order.

↳ Simple to implement but inefficient for large datasets.

↳ Ex:- Initial | 5 | 3 | 8 | 4 | 6 | Unsorted array



Ques!

① Print Transpose of a Matrix
② Add two matrix using multi-dimensional arrays.
③ Multiply 2 " " " " "
④ find the sum of each element in Matrix

Address Calculation

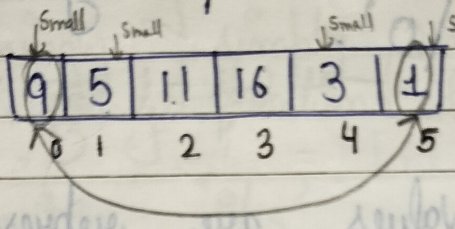
```
#include <stdio.h>
void main () {
static int array [10][10];
int i, j, m, n;
printf ("Enter the order of Matrix");
Scanf ("%d %d", &m, &n);
printf ("Enter the Coefficients of matrix");
  for (i = 0 ; i < n ; ++i){
    for ( j = 0 ; j<n ; ++j) {
      Scanf ("%d", &array [i] [j]);
    }
  }
printf ("The given matrix is \n");
  for (i = 0 ; i< m ; ++i) {
    for (j=0 ; j < n ; ++j) {
      printf ("%d", array [i][j]);
    }
printf ("\n");
}
printf ("Transpose of matrix is \n");
  for (j = 0 ; j <n ; ++j) {
    for (i=0 ; i < m ; ++i) {
      printf ("%d", array [i][j]);
    }
printf ("\n");
}
}
```

m row 0-(m-1)
n col 0-(n-1)

Page No.
Date 28 / 11 / 24

2. Selection sort

↳ Finds the minimum (smallest) element in the unsorted part of the array & swaps it with first element.

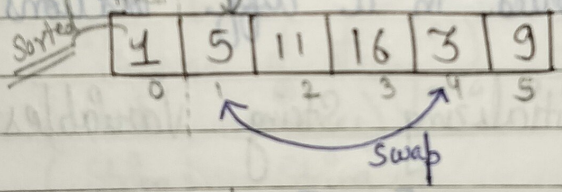


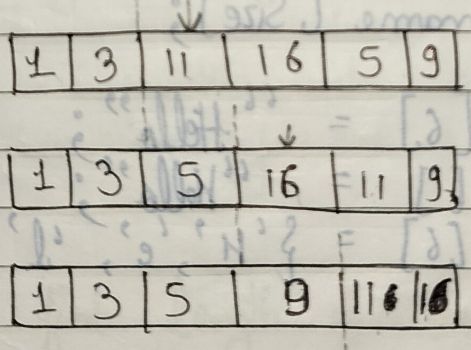
Max no. index
$(n-1)$

```
small  = a[0]
ismall = 0;
```

```
if ( a[j] < small) {
    small  = a[j];
    ismall = j ;
}
```

* This mthd doesn't compare with adjacent Elements.





$0 \rightarrow (n-1)$
$= n-1-0+1$
$= n$

$0 \rightarrow (n-2)$
$= n-2-0+1$
$= n-1$

The array is now sorted: [1, 3, 5, 9, 11, 16]

Strings :—

It's a sequence of characters treated as a group

↳ Any group of characters defined b/w double quote marks is a string constant.

↳ End of string is indicated by a delimeter, the zero character '\0'.
↳ Inputed by the compiler.

String literals values are represented by sequences of characters b/w double quotes.

"a" V/s 'a'

- String
- needs 2 bytes

- Char
- needs 1 byte

Duplicate String literals each string literals in a C-program is sorted in a diff. locations.

Declaring & Initializing String Variables.

predefin
func°.

```
char string_name [size];

char str1 [6] = "Hello";
char str2 [] = "Hello";
char str4 [6] = {'H', 'e', 'l', 'l', 'o', '\0'};

str1 = str2;
```

not allowable, but we can copy the contents of Str1 to Str2.

29/11/2

Q: Why do we need a terminating NULL character?

In C programming, the terminating NULL character, represented as '\0', is an essential element for defining & manipulating strings. It serves as a sentinel, marking the end of a character sequence. This seemingly simple concept plays a crucial role in various string operations & library functions.

format specifier '%s'

printf ("|%10s|", Name);
↳ gap before the name

o/p - /* | Neha| */

String Manipulation

predefined func°:
- strlen ("Name of string") — string.h
- strcpy (dest., source) — ↓
- strcmp (String1, String2) — header file
- strstr
- strcat()

29/11/24

Program to find length of string using 'for' loop.

```
#include <stdio.h>
#define MAX_SIZE 100
int main()
{
  char text[MAX_SIZE]; /* declares a string of size 100*/
```

```
int i;
int count = 0;
printf ("Enter any string:");
scanf ("%s", text);
/* iterate till the last character of string */
for (i=0; text[i] != '\0'; i++)
{
  count++;
}
printf ("length of string '%s' = %d", text, count);
return 0;
}
```

& can be written

→ not written b/c address of 1st element of array is passed automatically as array's first element's address is the base address.

O/P → Enter any string = Urmi
Length of string = 4

U	r	m	i	'\0'
0	1	2	3	4

* gets(text); It allows us to Enter multiword string separated by whitespaces. But shows some warning (not safe to use).

```
/* Program to Copy one string to another string */
#include <stdio.h>
# define MAX_SIZE 100 //
int main () {
  char text1 [MAX_SIZE];
  char text2 [MAX_SIZE];
  int i;
```

```
printf ("Enter any string:");
gets (text1);
for (i=0; text1[i]!='\0'; i++)
{
text2[i] = text[i]
}
text2[i] = '\0';
printf ("first string = %s\n", text1);
   ,, ("Second  ,,  =  ,,  ,, ", 2);
printf ("Total character copied = %d\n", i);
return 0;
}
```

consider blank space & will be terminated when '\0' is found

/* Prgm to Convert string to uppercase */

```
#include <stdio.h>
# define MAX_SIZE 100//
int main () {
char str[MAX_SIZE];
int i;
printf ("Enter your text:");
gets (str);
for (i=0; str[i]!='\0'; i++)
{
if (str[i]>='a' && str[i]<='z')
{
str[i] = str[i] - 32;
}
}
printf ("Uppercase string: %s", str);
return 0;
}
```

/* C prgm to Concatenate two strings using 'while' loop */

```
#include <stdio.h>
void concatenate strings (char str1(), char str2()) {
int i = 0, j=0;
while (str1(i) != '\0') {
   i++;
}
while (str2(j) != '\0') {
   str1(i) = str2(j);
   i++;
j++;
}
   str1(i) = '\0';
}
int main() {
Char str1(100), str2(100);
printf("Enter the first string:");
fgets(str1, sizeof(str1), stdin);
printf("Enter the second string:");
fgets(str2, sizeof(str2), stdin);
int len1 = 0;
while (str1(len1) != '\0') {
  if (str1(len1) == '\n') {
     str1(len1) = '\0';
     break;
}
  len1++;
}
int len2 = 0;
while (str2(len2) != '\0') {
```

```
    if (str2[len2] == '\n') {
        str2[len2] = '\0';
        break;
    }
    len2++;
}
concatenateStrings(str1, str2);
printf("Concatenated strings: %s\n", str1);
return 0;
}
```

O/P ⇒ Enter the first string: Hello
" " second " : World
Concatenated strings : HelloWorld.

↳ Tower of Hanoi

```
#include <stdio.h>
void TOH(int n, char source, char helper, char destination)
{
    if (n > 0){
        TOH(n-1, source, destination, helper);
        printf("%c → %c\n", source, destination);
        TOH(n-1, helper, source, destination);
    }
}
int main(){
    int n;
    printf("Enter size of array:");
    scanf("%d", &n);
    TOH(n, 'S', 'H', 'D');
    return 0;
}
```