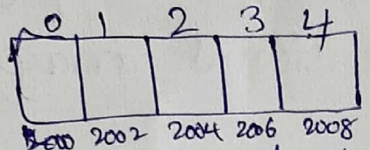
UNIT-V

ARRAYS

→ An array is defined as the collection of similar type of data items stored at contiguous memory locations.



→ Arrays are the derived data type in C Programming language which can store the primitive type of data such as int, char, double, float, etc.

→ Array is the simplest data structure where each data element can be randomly accessed by using its index number.

Advantage of C Array :—

1) code optimization : less code to the access the data.

2. Easy of traversing : By using the for loop, we can retriere the elements of an array easily

3. Ease of sorting :— To sort the elements of the array we need a few lines of code only.

Disadvantage of C Array :—

1) Fined size :— whatever size, we define at the time of declaration of the array, we can't exceed the limit. so, it doesn't grow the size dynamically like linked list

Scanned by CamScanner

Types of Arrays :-

Arrays divided into three types. They are

→ One dimensional arrays (1-D Array)
→ Two dimensional arrays (2-D Array)
→ Multi dimensional arrays (3-D array)

1) One dimensional Arrays (1-D Array) :-

A list of items can be given one variable name and elements are specified by one subscript are called single subscripted variable (or) one-dimensional array.

Declaration of 1-D Array :-

We can declare an array in the clanguage in the following way

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

Let us see the example to declare the array

Eg :- `int a[5];`

here, int is the data type, a is the array name, and 5 is the array size. Subscripts 0 to 4

Eg :- `float height[50];`

here, float is the datatype, height is the array name and 50 is the array size. Subscripts 0 to 49

Scanned by CamScanner

Initialization of 1-D Array :-

After an array is declared, its elements must be initialized. otherwise they will contain "garbage"

We can initialize the c Array at the time of declaration.

The general form of initialization of array is:

Syntax :- datatype arrayname [size] = { list of values };

Eg :- int a[5] = { 10, 20, 30, 40, 50 };

In such case, there is no requirement to define the size. So it may also be written as the following code.

arrayname index

int a[] = { 10, 20, 30, 40, 50 };

a[0]	a[1]	a[2]	a[3]	a[4]
10	20	30	40	50
2000	2002	2004	2006	2008

Array length = 5
First index = 0
Last index = 4

Here,

a[0] is equal to 10
a[1] is equal to 20
a[2] is equal to 30
a[3] is equal to 40
a[4] is equal to 50.

Scanned by CamScanner

Eg: Write a 'c' program to read and display elements using 1D array.

```
#include<stdio.h>
Void main()
{
int a[20], i, n;
Printf("enter the array size\n");
scanf("%d", &n);
Printf("enter the array elements\n");
for(i=0; i<n; i++)
scanf("%d", &a[i]);
Printf("the elements are\n");
for(i=0; i<n; i++)
Printf("%d\n", a[i]);
}
```

Output:

```
Enter the array size
5
Enter the array elements
2 5 10 15 20
The elements are
2
5
10
15
20
```

Scanned by CamScanner

2 Two-dimensional arrays (2-D Array) :-

→ The two-dimensional array can be defined as an array of arrays.

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

Declaration of two dimensional Array :-

The syntax to declare the 2D array is given below

Syntax :- `data type array-name [rows] [columns];`

consider the following example

Eg: `int a[3][4];`
`float b[2][3];`

Here, 3 is the number of rows and 4 is the number of columns. a is the array name int is the what type of data it stores.

Initializing Two-dimensional Arrays :-

Like the one dimensional arrays, two-dimensional arrays may be initialized by following their declaration with a list of initial values

Scanned by CamScanner

enclosed in braces for example

Compile time initialization :-

The syntax to initialize values at compile time

Syntax : datatype array-name [rows][column] = { list of values };

Eg : int a[2][3] = {0, 0, 0, 1, 1, 1};

initialize the elements of the first row to zero and the second row to one. The initialization is done row by row. The above statement can be equivalently written as

int a[2][3] = {{0, 0, 0}, {1, 1, 1}};

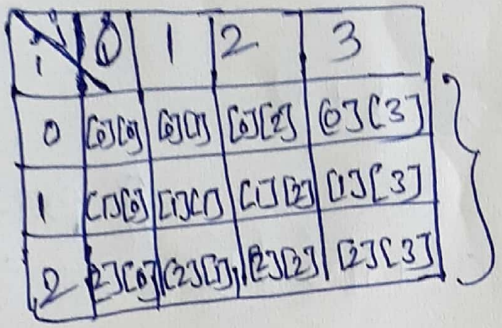
Accessing And Reading the array :-

In case of 2D arrays, we use index numbers (like 1D arrays) in the subscript to access the array elements.

The outer loop indicates the row index and the inner loop indicates the column index.

Scanned by CamScanner

The following figure shows how the array elements are indexed:



i \ j	0	1	2	3
0	[0][0]	[0][1]	[0][2]	[0][3]
1	[1][0]	[1][1]	[1][2]	[1][3]
2	[2][0]	[2][1]	[2][2]	[2][3]

index values for 2D array elements

Index numbers are specified as

[row number] [column number]

Ex: write a 'c' program to read and display array elements using 2 dimensional array.

```
#include<stdio.h>
void main()
{
int a[2][2], i, j
Printf("enter the array elements \n");
for( i=0; i<2 ; i++)
{
 for( j=0; j<2 ; j++)
   {
    scanf("%d", &a[i][j]); //reading 2D elements
   }
}
```

Scanned by CamScanner

```
printf("The elements are \n");
for(i=0; i<2; i++)
  {
  for(j=0; j<2; j++)
    {
    printf("%d", a[i][j]); // printing array elements of 2D.
    }
  printf("\n");
  }
```

O/P:

```
enter the array elements
1 2 3 4
The elements are
1 2
3 4.
```

Searching and sorting Techniques.

What is Searching :-

Searching is an operation (or) a technique that helps finds the place of a given element (or) value in the list. Any search is said to be successful or unsuccessful depending upon whether the element that is being searched is found (or) not.

Scanned by CamScanner

Some of the standard searching techniques are

→ Linear search (or) sequential search
→ Binary search

What is Linear search :—

→ This is the simplest method for searching.
→ In this technique of searching, the element to be found is searching sequentially in the list.
→ This method can be performed on a sorted list (or) an unsorted list (usually arrays)
→ sorted list searching starts from 0th element and continues until the element is found from the list

Eg: Write a 'c' program to perform Linear search.

```
#include<stdio.h>
void main()
{
    int a[10], i, found=0;
Printf("enter the array elements\n");
```

Scanned by CamScanner

```
for( i=0; i<9; i++)
scanf("%d", &a[i]);
Printf("enter the key to be searched\n");
scanf("%d", &key);
for( i=0; i<9; i++)
  {
    if( a[i] == key)
      {
        found = 1;
        break;
      }
  }
if( found == 1)
  Printf("The element is found\n");
  else
Printf("the element is not found\n");
}.
```

O/P: enter the array elements

2 30 15 10 20 1 50 45 60 100

enter the key to be searched

20

The element is found.

Scanned by CamScanner

Binary search :-

→ Binary search is a very fast and efficient searching technique.

→ It requires the list to be in sorted order.

→ In this method, to search an element you can compare it with the present element at the center of the list. if it matches, then the search is successful otherwise the list is divided into two halves

→ The searching mechanism proceeds from either of the two halves depending upon whether the target element is greater (or) smaller than the middle element.

→ If the element is smaller than the middle element, then searching is done in the first half. otherwise searching is done in the second half.

Eg:

```
#include<stdio.h>
void main()
{
    int a[10], i, key, first, last, found=0;
Printf("enter the array elements \n");
for(i=0; i<9; i++)
```

Scanned by CamScanner

```
scanf("%d", &a[i]);
Printf("enter the key to be searched\n");
    scanf("%d", &key);
    top=0, bottom=size-1;
for while(  first=0, bot last=n;
    while(first<=last)
      {
      middle=(first+last)/2;
    if(a[middle]<key)
        {  first = middle+1;
        }
    else if(a[middle]>key)
          {  last = middle-1;
          }
    else if(a[middle]==key)
        {
          found=1;
          break;
        }
    }.
      if(found==1)
  Pf("the element is found\n");
    else
  Pf("the element is not found\n");
```

Scanned by CamScanner

Sorting Techniques

→ Sorting is nothing but arranging the data in ascending (or) descending order.

→ There are so many things in our real life that we need to search for, like a particular record in database, roll numbers in merit list, a particular telephone number in telephone directory.

→ sorting arranges data in a sequence which makes searching easier.

→ The main purpose of sorting is to easily & quickly locate an element in a sorted list.

Some of the standard sorting techniques are

→ Bubble sort

→ Insertion sort.

① Bubble sort :—

→ Bubble sort is a simple sorting algorithm

→ which repeatedly compares the adjacent elements of the given array & swap them if they are in wrong order.

Eg: Suppose we have an array x which contains n elements which needs to be sorted using Bubble sort.

Scanned by CamScanner

consider the list 74, 39, 35, 97, 84

Pass 1: first element is compared with all other elements

1) compare 74 and 39, since 74 > 39, they are swapped.
The array changes like 39, 74, 35, 97, 84

2) compare 39 and 35, since 39 > 35, they are swapped
The array now changes like 35, 74, 39, 97, 84

3) compare 35 and 97, since 35 < 97, they are undistributed
The array is now 35, 74, 39, 97, 84

4) compare 35 and 84, since 35 < 84, they are undistributed
The array is now 35, 74, 39, 97, 84

At the end of this pass, the first element will be in the correct place.

Pass 2:- The second pass begins with the second element and is compared with all other elements.
after pass 2 the array is now 35, 39, 74, 97, 84

Pass 3:- third element is compared with all other elements
Now the array is 35, 39, 74, 97, 84

Pase 4: fourth element is compared with other elements
Now the array is 35, 39, 74, 84, 97.

No. of comparisons in the first pass = (n-1)
No. of comparisons in the second pass = (n-2)
No. of comparisons in the last pass = 1
∴ The order of the bubble sort is $O(n^2)$.

Scanned by CamScanner

Write a c program to sort a given array by using bubble sort

```
#include<stdio.h>
void main()
{
int a[5], i, j, temp;
printf("enter the elements of an array\n");
for(i=0; i<5; i++)
    scanf("%d", &a[i]);
for(i=0; i<5; i++)
{
for(j=i; j<4; j++)
{
if(a[i] > a[j+1])
{
temp= a[i];
a[i] = a[j+1];
a[j+1] = temp;
}
}
}
printf("sorted elements in ascending order:\n");
for(i=0; i<5; i++)
printf("%d", a[i]);
}
```

O/P: enter the elements of an array

9 5 1 4 3

Sorted elements in ascending order

1 3 4 5 9

Scanned by CamScanner

2) Insertion sort :—

Insertion sort is a simple sorting algorithm that works the way we sort playing cards in our hands.

→ Insertion sort is basically insertion of an element from a random set of numbers, to its correct position where it should actually be, by shifting the other elements if required

→ Insertion sort algorithm to arrange numbers of an array in ascending order.

Example :— consider the list | 12 | 11 | 13 | 5 | 6 |

Let us loop for i=1 (second element of the array) to 4 (last element of the array)

Step 1: i=1, since 11 is smaller than 12, move 12 and insert 11 before 12.

| 11 | 12 | 13 | 5 | 6 |

Step 2: i=2, 13 will remain at its position as all elements in a[0 --- n-1] are small than 13

| 11 | 12 | 13 | 5 | 6 |

Step 3: i=3, 5 will move to the beginning and all other elements from 11 to 13 will move one position ahead of their current position.

Scanned by CamScanner

5	11	12	13	6

Step 4 :- i=4, 6 will move to position after 5, and elements from 11 to 13 will move one position ahead of their current position.

5	6	11	12	13.

Write a 'c' program to implement insertion sort.

```
#include<stdio.h>
void main()
{
int a[10], i, temp, j, n;
printf("enter the array size \n");
scanf("%d", &n);
printf("enter the array elements \n");
for(i=0; i<n; i++)
scanf("%d", &a[i]);
for(i=1; i<n; i++)
{
temp = a[i];
j = i-1;
while(j>=0 && a[j]>temp) // to move elements greater than temp, to one position of their current position.
{ a[j+1]=a[j];
  j=j-1;
```

Scanned by CamScanner

```
        a[j+1] = temp;
    }
    printf("Sorted elements in ascending order:\n");
    for(i=0; i<n; i++)
        printf("%d\n", a[i]);
}
```

O/P:

```
enter the array size
5
enter the array elements
12 11 13 5 6
Sorted elements in ascending order
5 6 11 12 13.
```

Applications of Arrays :-

→ Arrays are used to implement mathematical vectors and matrices.

→ Arrays are used to implement data structures like lists, heaps, hashtables, queues and stacks

→ Arrays are used to store list of values

→ Arrays are used to implement Search algorithms

→ Arrays are used to implement Sorting algorithms

→ Arrays are also used to implement CPU scheduling algorithms.

Scanned by CamScanner

Time complexity :—

Time complexity is the computational complexity that describes the amount of time it takes to run an algorithm.

(or)

Time complexity of a particular algorithm is a function, this function takes input as length of data entered in that algorithm and determines how much time (or) cycles need to run that algorithm for that input.

Time complexity for searching algorithms :—

	Best	Average	worst
1) Linear search :	O(1)	O(n)	O(n)
2) Binary search	O(1)	O(log n)	O(log n)

Time complexity for sorting algorithms :

	Best case	Average case	worst time complexity
1) Bubble sort	O(n)	$O(n^2)$	$O(n^2)$
2) Insertion sort	O(n)	$O(n^2)$	$O(n^2)$

Scanned by CamScanner

difference between exit() and return statement:

→ ① exit() is a predefined library function of stdlib.h. where as return is a jumping statement and it is a keyword which is defined in the compiler.

→ ② exit() terminates the program's execution and returns the program's control to the OS.

return statement returns the program's control to the calling function from called function

if you use, return in the main() function it transfers control from main() (called function) to the operation system (OS) (calling function)

Example with exit() :-

```
void main()
  {
Printf("statement-1 \n");
 Printf("statement-2\n");
  exit(0);
Printf("statement-N\n");
}
```

O/P: statement-1
statement-2.

Scanned by CamScanner