**Module – 3**

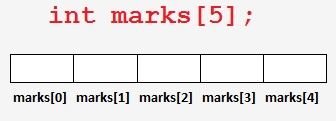
**Arrays**

**3.1 Arrays - 1-D and 2-D**

**Arrays in C Language**

An **array** is a collection of a fixed number of values of a single data type. For example: if you want to store 100 integers in sequence, you can create an **array** for it. int data[100]; The size and type of **arrays** cannot be changed after its declaration. An **array** is a collection of data items, all of the same type, accessed using a common name. A one-dimensional **array** is like a list; A two dimensional **array** is like a table; The **C** language places no limits on the number of dimensions in an **array**, though specific implementations may.

***Advantages of Using Arrays:*** It is used to represent multiple data items of same type by using only single name. It can be used to implement other data structures like linked lists, stacks, queues, trees, graphs etc. 2D **arrays** are used to represent matrices. Representation of array of integers is shown in Fig 3.1.

******

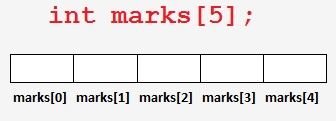
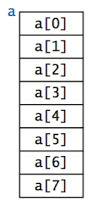
**Fig 3.1:** Array of integers

* **Types of Arrays**

There are two [type of array](https://goo.gl/5FhTk6) in C language:

* ***One dimensional array:***

Single or One Dimensional array is used to represent and store data in a linear form. Array having only one subscript variable is called One-Dimensional array. It is also called as Single Dimensional Array or Linear Array. Fig 3.2 shows one dimensional array.



**Fig 3.2:** One Dimensional Array

***Declaration of 1D array:***

For example: if you want to store 100 integers in sequence, you can create an array for it.

data\_type array\_name[array\_size];

Ex.:  **int data[100];**

***How to initialize a 1D-array?***

It's possible to initialize an array during declaration. For example,

**int mark[5] = {19, 10, 8, 17, 9};**

**or**

**int mark[ ] = {19, 10, 8, 17, 9};**

Here, the array mark is initialized as shown below.

**mark[0] is equal to 19**

**mark[1] is equal to 10**

**mark[2] is equal to 8**

**mark[3] is equal to 17**

**mark[4] is equal to 9**

## Example: C Arrays

**// Program to find the average of n (n < 10) numbers using arrays**

#include <stdio.h>

int main()

{

int marks[10], i, n, sum = 0, average;

printf("Enter n: ");

scanf("%d", &n);

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

{

printf("Enter number %d: ",i+1);

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

sum += marks[i];

}

average = sum/n;

printf("Average = %d", average);

return 0;

}

**Output**

Enter n: 5

Enter number1: 45

Enter number2: 35

Enter number3: 38

Enter number4: 31

Enter number5: 49

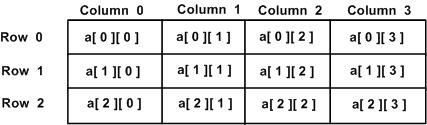
Average = 39

* ***Two dimensional arrays:***

Array having more than one subscript variable is called Multi-dimensional array.

Multi-Dimensional Array is also called as Matrix.

Ex. An 2D array int a[3][3], is shown in the Fig.3.3



**Fig 3.3:** 2D array of integers

***How to initialize 2D-arrays?***

There is more than one way to initialize a multidimensional array.

**int c[2][3] = {{1, 3, 0}, {-1, 5, 9}};**

**int c[][3] = {{1, 3, 0}, {-1, 5, 9}};**

**int c[2][3] = {1, 3, 0, -1, 5, 9};**

**3.2 Arrays Contd.. and Example programs**

// **C program to find the sum of two matrices of order 2\*2**

#include <stdio.h>

int main()

{

float a[2][2], b[2][2], c[2][2];

int i, j;

// Taking input using nested for loop

printf("Enter elements of 1st matrix\n");

for(i=0; i<2; ++i)

for(j=0; j<2; ++j)

{

printf("Enter a%d%d: ", i+1, j+1);

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

}

// Taking input using nested for loop

printf("Enter elements of 2nd matrix\n");

for(i=0; i<2; ++i)

for(j=0; j<2; ++j)

{

printf("Enter b%d%d: ", i+1, j+1);

scanf("%f", &b[i][j]);

}

// adding corresponding elements of two arrays

for(i=0; i<2; ++i)

for(j=0; j<2; ++j)

{

c[i][j] = a[i][j] + b[i][j];

}

// Displaying the sum

printf("\nSum Of Matrix:");

for(i=0; i<2; ++i)

for(j=0; j<2; ++j)

{

printf("%.1f\t", c[i][j]);

if(j==1)

printf("\n");

}

return 0;

}

**Output**

Enter elements of 1st matrix

Enter a11: 2;

Enter a12: 0.5;

Enter a21: -1.1;

Enter a22: 2;

Enter elements of 2nd matrix

Enter b11: 0.2;

Enter b12: 0;

Enter b21: 0.23;

Enter b22: 23;

Sum of Matrix:

2.2 0.5

-0.9 25.0

## 3.4 Character Array & Strings

**String** is a sequence of characters that is treated as a single data item and terminated by null character '\0'. Remember that C language does not support strings as a data type. A **string** is actually one-dimensional array of characters in C language. These are often used to create meaningful and readable programs.

**For example:** The string "hello world" contains 12 characters including '\0' character which is automatically added by the compiler at the end of the string.

## Declaring and Initializing a string variables

There are different ways to initialize a character array variable.

char name[13] = "StudyTonight"; // valid character array initialization

char name[10] = {'L','e','s','s','o','n','s','\0'}; // valid initialization

Remember that when you initialize a character array by listing all of its characters separately then you must supply the '\0' character explicitly.

Some examples of illegal initialization of character array are,

char ch[3] = "hell"; // Illegal

char str[4];

str = "hell"; // Illegal

* **String Input and Output**

Input function **scanf()** can be used with %s format specifier to read a string input from the terminal. But there is one problem with **scanf()** function, it terminates its input on the first white space it encounters. Therefore if you try to read an input string **"Hello World"** using **scanf()** function, it will only read Hello and terminate after encountering white spaces.

However, C supports a format specification known as the **edit set conversion code %[..]** that can be used to read a line containing a variety of characters, including white spaces.

#include<stdio.h>

#include<string.h>

void main()

{

char str[20];

printf("Enter a string");

scanf("%[^\n]", &str); //scanning the whole string, including the white spaces

printf("%s", str);

}

Another method to read character string with white spaces from terminal is by using the **gets()** function.

char text[20];

gets(text);

printf("%s", text);

* **String Handling Functions**

C language supports a large number of string handling functions that can be used to carry out many of the string manipulations. These functions are packaged in **string.h** library. Hence, you must include **string.h** header file in your programs to use these functions.

The following are the most commonly used string handling functions.

| **Method** | **Description** |
| --- | --- |
| **strcat()** | It is used to concatenate(combine) two strings |
| **strlen()** | It is used to show length of a string |
| **strrev()** | It is used to show reverse of a string |
| **strcpy()** | Copies one string into another |
| **strcmp()** | It is used to compare two string |

#### strcat() function

strcat("hello", "world");

**strcat()** function will add the string **"world"** to **"hello"** i.e it will output **helloworld.**

#### strlen() function

**strlen()** function will return the length of the string passed to it.

int j;

j = strlen("studytonight");

printf("%d",j);

**Output: 12**

#### strcmp() function

**strcmp()** function will return the ASCII difference between first unmatching character of two strings.

int j;

j = strcmp("study", "tonight");

printf("%d",j);

**Output: -1**

#### strcpy() function

It copies the second string argument to the first string argument.

#include<stdio.h>

#include<string.h>

int main()

{

char s1[50];

char s2[50];

strcpy(s1, "StudyTonight"); //copies "studytonight" to string s1

strcpy(s2, s1); //copies string s1 to string s2

printf("%s\n", s2);

return(0);

}

**Output: StudyTonight**

#### strrev() function

It is used to reverse the given string expression.

#include<stdio.h>

int main()

{

char s1[50];

printf("Enter your string: ");

gets(s1);

printf("\nYour reverse string is: %s",strrev(s1));

return(0);

}

**Output:**

Enter your string: **studytonight**

Your reverse string is: **thginotyduts**

**3.5 Searching Algorithms**

* ***Searching algorithm Definition:*** A search **algorithm** is the step-by-step procedure used to locate specific data among a collection of data. It is considered a fundamental procedure in computing. In computer science, when **searching** for data, the difference between a fast application and a slower one often lies in the use of the proper search **algorithm**.

These algorithms are generally **classified into two categories**:

1. ***Sequential Search:*** In this, the list or array is traversed sequentially and every element is checked. **For example:** [**Linear Search**](https://www.geeksforgeeks.org/linear-search/)**.**
2. ***Interval Search:*** These algorithms are specifically designed for searching in sorted data-structures. This type of searching algorithms are much more efficient than Linear Search as they repeatedly target the center of the search structure and divide the search space in half. **For Example:** [**Binary Search**](https://www.geeksforgeeks.org/binary-search/)**.**

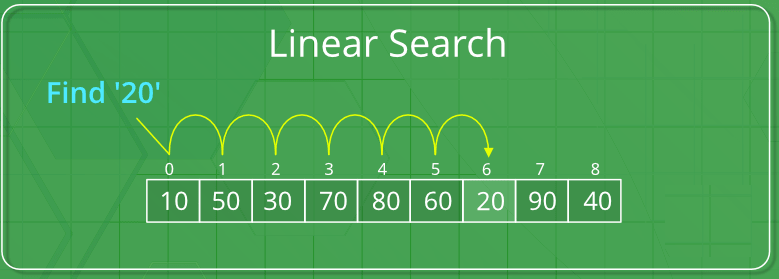
* ***Sorting Algorithm Definition:*** A **sorting algorithm** is an **algorithm** that puts elements of a list in a certain order. Efficient **sorting** is important for optimizing the efficiency of other **algorithms** (such as search and merge **algorithms**) which require input data to be in sorted lists.

These algorithms are generally **classified into six categories**:

1. *Bubble Sort.*
2. *Selection Sort.*
3. *Merge Sort.*
4. *Insertion Sort.*
5. *Quick Sort.*
6. *Heap Sort.*

**3.6 Searching algorithm- Linear Search**

Linear search is a very basic and simple search algorithm. In Linear search, we search an element or value in a given array by traversing the array from the starting, till the desired element or value is found. Fig 3.4 shows an illustration of Linear Search algorithm.

**

## Fig 3.4: Linear Search Algorithm

## Implementing Linear Search

The steps to implement linear search:

1. Traverse the array using a for loop.
2. In every iteration, **compare** the **target value with the current value** of the array.

* If the values match, return the current index of the array.
* If the values do not match, move on to the next array element.

1. If no match is found, **return -1**.

* ***Write a ‘C’ Program to search an element using Linear search***

#include <stdio.h>

int main()

{

  int array[100], search, c, n;

   printf("Enter number of elements in array\n");

  scanf("%d", &n);

  printf("Enter %d integer(s)\n", n);

   for (c = 0; c < n; c++)

  scanf("%d", &array[c]);

  printf("Enter a number to search\n");

  scanf("%d", &search);

  for (c = 0; c < n; c++)

  {

    if (array[c] == search)    */\* If required element is found \*/*

    {

      printf("%d is present at location %d.\n", search, c+1);

      break;

    }

  }

  if (c == n)

    printf("%d isn't present in the array.\n", search);

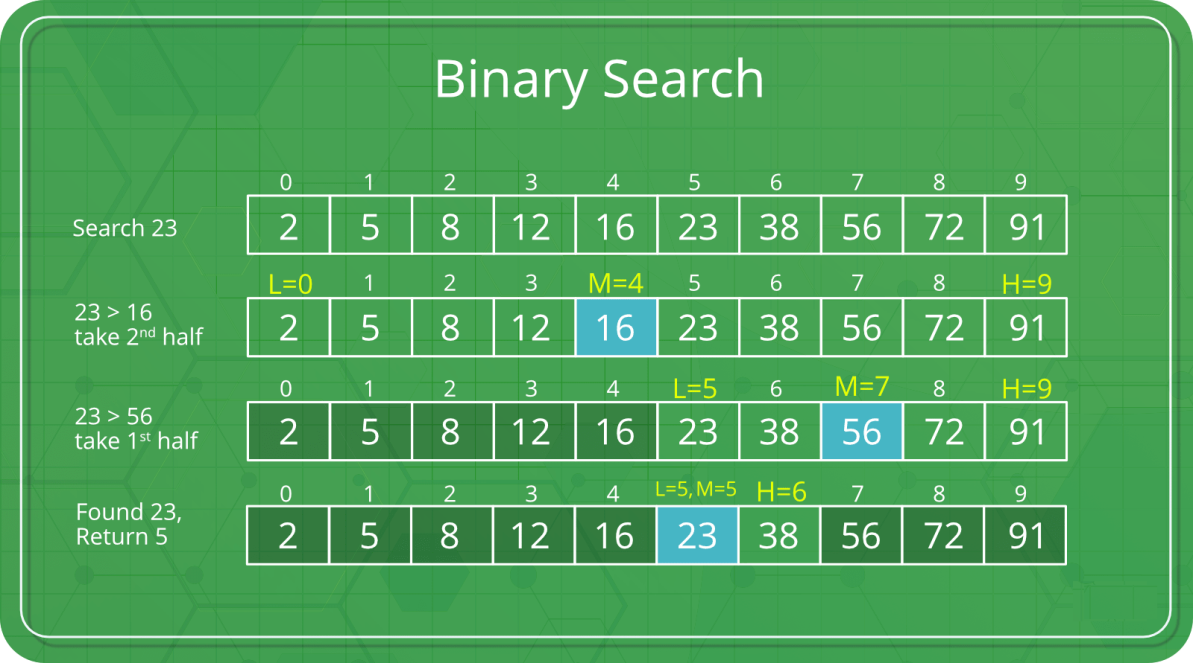
  return 0;

}

**3.7 Searching algorithm- Binary Search**

Binary search is a fast search algorithm and works on the principle of divide and conquer. For this algorithm to work properly, the data collection should be in the sorted form.

Binary search looks for a particular item by comparing the middle most item of the collection. If a match occurs, then the index of item is returned. If the middle item is greater than the item, then the item is searched in the sub-array to the left of the middle item. Otherwise, the item is searched for in the sub-array to the right of the middle item. This process continues on the sub-array as well until the size of the subarray reduces to zero. Fig 3.5 shows the illustration of Binary Search Algorithm.



**Fig 3.5:** Binary Search Algorithm

## *How Binary Search Works*?

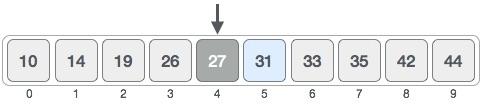
For a binary search to work, it is mandatory for the target array to be sorted. We shall learn the process of binary search with a pictorial example. The following is our sorted array and let us assume that we need to **search the location of value 31 using binary search.**



First, we shall determine **half of the array** by using this formula −

mid = low + (high - low) / 2

Here it is, 0 + (9 - 0 ) / 2 = 4 (integer value of 4.5). So, **4 is the mid of the array.**



Now we compare the value stored at location **4**, with the value being searched, i.e. **31**. We find that the value at location 4 is 27, which is not a match. As the value is greater than 27 and we have a sorted array, so we also know that the target value must be in the upper portion of the array.



We change our **low to mid + 1** and find the new mid value again.

low = mid + 1

mid = low + (high - low) / 2

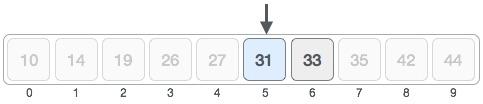
Our new mid is 7 now. We compare the value stored at location 7 with our target value 31.



The value stored at location **7** is not a match, rather it is more than what we are looking for. So, the value must be in the lower part from this location.



Hence, **we calculate the mid again. This time it is 5**.



We compare the value stored at location **5** with our target value. We find that it is a match.



We conclude that the target value **31 is stored at location 5.**

Binary search halves the searchable items and thus reduces the count of comparisons to be made to very less numbers.

**Write a ‘C’ Program to search an element using Binary search**

#include <stdio.h>

 int main()

{

   int c, first, last, middle, n, search, array[100];

   printf("Enter number of elements**\n**");

   scanf("%d",&n);

   printf("Enter %d integers**\n**", n);

   for (c = 0; c < n; c++)

      scanf("%d",&array[c]);

   printf("Enter value to find**\n**");

   scanf("%d", &search);

   first = 0;

   last = n - 1;

   middle = (first+last)/2;

   while (first <= last) {

      if (array[middle] < search)

         first = middle + 1;

      else if (array[middle] == search) {

         printf("%d found at location %d.**\n**", search, middle+1);

**break**;

      }

      else

         last = middle - 1;

      middle = (first + last)/2;

   }

   if (first > last)

      printf("Not found! %d isn't present in the list.**\n**", search);

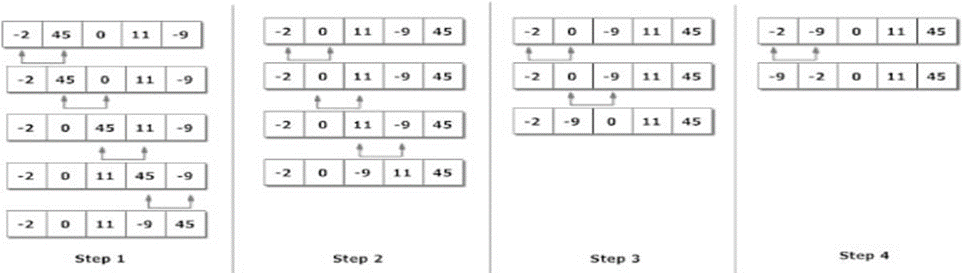
   return 0;

}

**3.8 Sorting algorithms - Bubble Sort**

Bubble sort algorithm starts by comparing the first two elements of an array and swapping if necessary, i.e., if you want to sort the elements of array in ascending order and if the first element is greater than second then, you need to swap the elements but, if the first element is smaller than second, you mustn't swap the element. Then, again second and third elements are compared and swapped if it is necessary and this process go on until last and second last element is compared and swapped. This completes the first step of bubble sort.

If there are n elements to be sorted then, the process mentioned above should be repeated n-1 times to get required result. But, for better performance, in second step, last and second last elements are not compared because; the proper element is automatically placed at last after first step. Similarly, in third step, last and second last and second last and third last elements are not compared and so on. Fig 3.6 shows the working of bubble sort algorithm.



**Fig 3.6:** Working of Bubble Sort Algorithm

* ***Write a ‘C’ Program to sort the elements using Bubble sort***

/\*C Program To Sort data in ascending order using bubble sort.\*/

#include **<stdio.h>**

**int main()**

**{**

**int data[100],i,n,step,temp;**

**printf("Enter the number of elements to be sorted: ");**

**scanf("%d",&n);**

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

**{**

**printf("%d. Enter element: ",i+1);**

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

**}**

**for(step=0;step<n-1;++step)**

**for(i=0;i<n-step-1;++i)**

**{**

**if(data[i]>data[i+1])** /\* To sort in descending order, change > to < in this line. \*/

**{**

**temp=data[i];**

**data[i]=data[i+1];**

**data[i+1]=temp;**

**}**

**}**

**printf("In ascending order: ");**

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

**printf("%d ",data[i]);**

**return 0;**

**}**

**Enter the number of elements to be sorted:** **6**

1. Enter element: **12**

2. Enter element: **3**

3. Enter element: **0**

4. Enter element: **-3**

5. Enter element: **1**

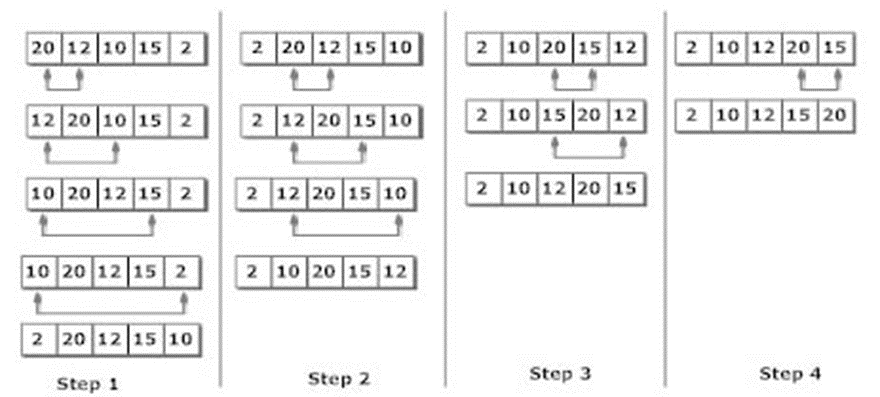
6. Enter element: **-9**

**In ascending order: -9 -3 0 1 3 13**

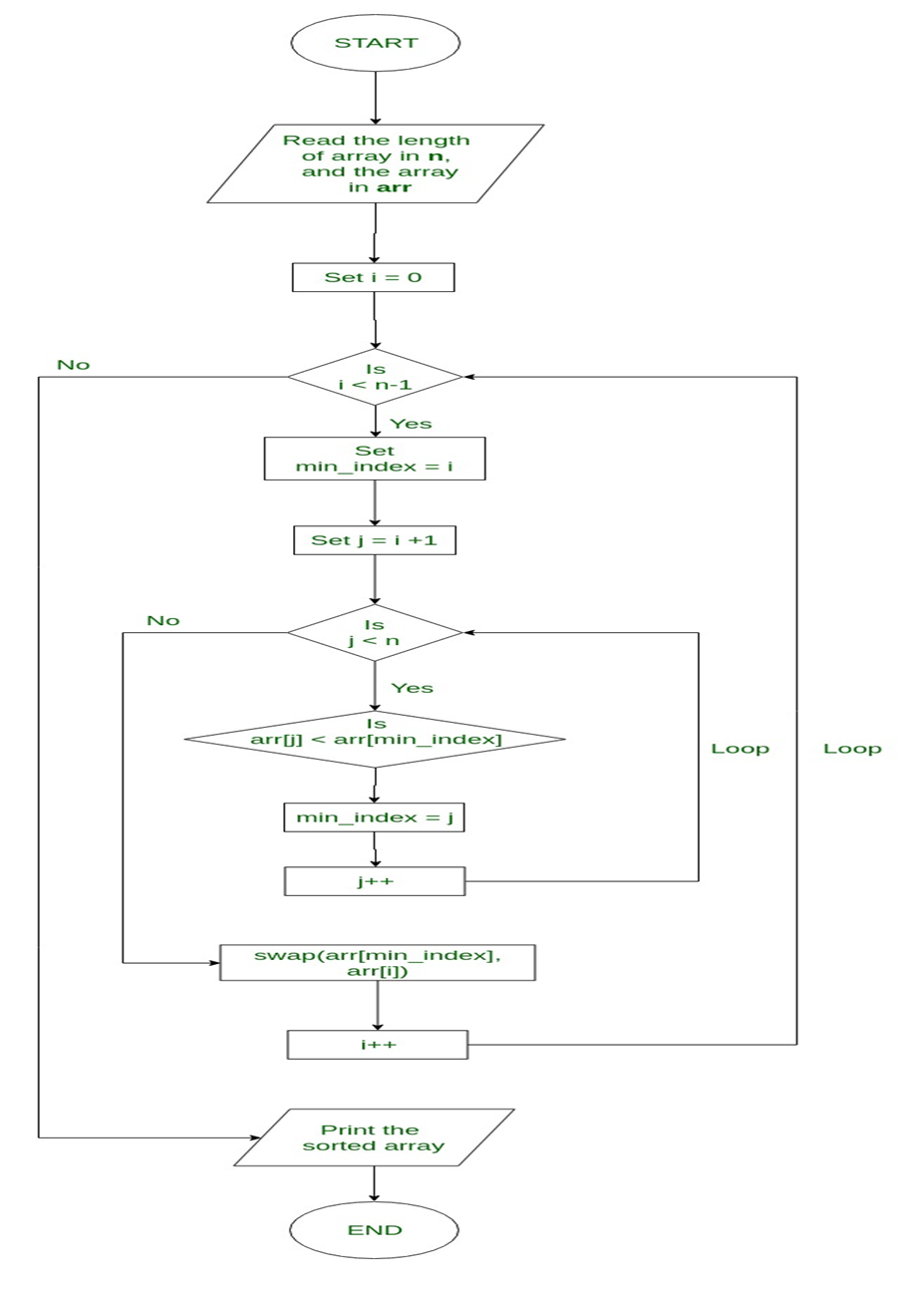
* ***Sorting algorithms - Selection Sort***

Selection sort algorithm starts by comparing first two elements of an array and swapping if necessary, i.e., if you want to sort the elements of array in ascending order and if the first element is greater than second then, you need to swap the elements but, if the first element is smaller than second, leave the elements as it is. Then, again first element and third element are compared and swapped if necessary. This process goes on until first and last element of an array is compared. This completes the first step of selection sort.

If there are n elements to be sorted then, the process mentioned above should be repeated n-1 times to get required result. But, for better performance, in second step, comparison starts from second element because after first step, the required number is automatically placed at the first (i.e, In case of sorting in ascending order, smallest element will be at first and in case of sorting in descending order, largest element will be at first.). Similarly, in third step, comparison starts from third element and so on. Fig 3.7 shows the working of selection sort algorithm and Fig 3.8 shows the flowchart of selection sort algorithm.



**Fig 3.7:** Working of Selection Sort Algorithm



**Fig 3.8:** Flowchart Selection Sort

**Write a ‘C’ Program to sort the elements using Selection Sort**

#include **<stdio.h>**

**int main()**

**{**

**int data[100],i,n,steps,temp;**

**printf("Enter the number of elements to be sorted: ");**

**scanf("%d",&n);**

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

**{**

**printf("%d. Enter element: ",i+1);**

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

**}**

**for(steps=0;steps<n;++steps)**

**for(i=steps+1;i<n;++i)**

**{**

**if(data[steps]>data[i])**

/\* To sort in descending order, change > to <. \*/

**{**

**temp=data[steps];**

**data[steps]=data[i];**

**data[i]=temp;**

**}**

**}**

**printf("In ascending order: ");**

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

**printf("%d ",data[i]);**

**return 0;**

**}**

**Practice Programs:**

1. **Write a C program to calculate Average Using Arrays**. Solution:

#include <stdio.h>

int main()

{

int n, i;

float num[100], sum=0.0, average; printf("Enter the numbers of data: "); scanf("%d",&n);

while (n>100 || n<=0)

{

printf("Error! number should in range of (1 to 100).\n"); printf("Enter the number again: ");

scanf("%d",&n);

}

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

{

printf("%d. Enter number: ",i+1); scanf("%f",&num[i]); sum+=num[i];

}

average=sum/n;

printf("Average = %.2f",average); return 0;

}

## Output

Enter the numbers of data: 6

1. Enter number: 45.3
2. Enter number: 67.5
3. Enter number: -45.6
4. Enter number: 20.34
5. Enter number: 33
6. Enter number: 45.6 Average = 27.69

This program calculates the average if the number of data is from 1 to 100. If user enters value of *n* above 100 or below 100 then, while loop is executed which asks user to enter value of *n* until it is between 1 and 100.

1. **Write a C program to Display Largest Element of an array**

#include <stdio.h> int main()

{

int i,n;

float arr[100];

printf("Enter total number of elements(1 to 100): "); scanf("%d",&n);

printf("\n");

for(i=0;i<n;++i) /\* Stores number entered by user. \*/

{

printf("Enter Number %d: ",i+1); scanf("%f",&arr[i]);

}

for(i=1;i<n;++i) /\* Loop to store largest number to arr[0] \*/

{

if(arr[0]<arr[i]) /\* Change < to > if you want to find smallest element\*/ arr[0]=arr[i];

}

printf("Largest element = %.2f",arr[0]); return 0;

}

## Output

Enter total number of elements(1 to 100): 8 Enter Number 1: 23.4

Enter Number 2: -34.5

Enter Number 3: 50

Enter Number 4: 33.5

Enter Number 5: 55.5

Enter Number 6: 43.7

Enter Number 7: 5.7

Enter Number 8: -66.5

This program takes *n* number of elements from user and stores it in array *arr[]*. To find the largest element, the first two elements of array are checked and largest of these two element is placed in *arr[0]*. Then, the first and third elements are checked and largest of these two element is placed in *arr[0]*. This process continues until and first and last elements are checked. After this process, the largest element of an array will be in *arr[0]* position.

**3. Write a C program to multiply to matrix in C programming**

#include <stdio.h>

int main()

{

int a[10][10], b[10][10], mult[10][10], r1, c1, r2, c2, i, j, k;

printf("Enter rows and column for first matrix: ");

scanf("%d%d", &r1, &c1);

printf("Enter rows and column for second matrix: ");

scanf("%d%d",&r2, &c2);

/\* If column of first matrix in not equal to row of second matrix, asking user to enter the size of matrix again. \*/

while (c1!=r2)

{

printf("Error! column of first matrix not equal to row of second.\n\n");

printf("Enter rows and column for first matrix: ");

scanf("%d%d", &r1, &c1);

printf("Enter rows and column for second matrix: ");

scanf("%d%d",&r2, &c2);

}

/\* Storing elements of first matrix. \*/

printf("\nEnter elements of matrix 1:\n");

for(i=0; i<r1; ++i)

for(j=0; j<c1; ++j)

{

printf("Enter elements a%d%d: ",i+1,j+1); scanf("%d",&a[i][j]);

}

/\* Storing elements of second matrix. \*/

printf("\nEnter elements of matrix 2:\n");

for(i=0; i<r2; ++i)

for(j=0; j<c2; ++j)

{

printf("Enter elements b%d%d: ",i+1,j+1); scanf("%d",&b[i][j]);

}

/\* Initializing elements of matrix mult to 0.\*/

for(i=0; i<r1; ++i)

for(j=0; j<c2; ++j)

{

mult[i][j]=0;

}

/\* Multiplying matrix a and b and storing in array mult. \*/

for(i=0; i<r1; ++i)

for(j=0; j<c2; ++j)

for(k=0; k<c1; ++k)

{

mult[i][j]+=a[i][k]\*b[k][j];

}

/\* Displaying the multiplication of two matrix. \*/ printf("\nOutput Matrix:\n");

for(i=0; i<r1; ++i)

for(j=0; j<c2; ++j)

{

printf("%d ",mult[i][j]);

if(j==c2-1)

printf("\n\n");

}

return 0;

}

**Output**

Enter rows and column for first matrix: 3 2

Enter rows and column for second matrix: 3 2

Error! column of first matrix not equal to row of second.

Enter rows and column for first matrix: 2 3

Enter rows and column for second matrix: 3 2

Enter elements of matrix 1:

Enter elements a11: 3

Enter elements a12: -2

Enter elements a13: 5

Enter elements a21: 3

Enter elements a22: 0

Enter elements a23: 4

Enter elements of matrix 2:

Enter elements b11: 2

Enter elements b12: 3

Enter elements b21: -9

Enter elements b22: 0

Enter elements b31: 0

Enter elements b32: 4

Output Matrix:

24 29

6 25

In this program, user is asked to enter the size of two matrix at first. The column of first matrix should be equal to row of second matrix for multiplication. If this condition is not satisfied then, the size of matrix is again asked using while loop. Then, user is asked to enter two matrix and finally the output of two matrix is calculated and displayed.

This program is little bit larger and it is better to solve this program by passing it to a function.

**4. What will be the output when you execute the below statements:**

#include<stdio.h>

void main()

{

char arr[7]=”Network”;

printf(“%s”,arr);

}

**Explanation:**

Size of a character array should one greater than total number of characters in any string which it stores. Inc every string has one terminating null character. This represents end of the string.

So in the string “Network” , there are 8 characters and they are ‘N’,’e’,’t’,’w’,’o’,’r’,’k’ and ‘\0’. Size of array arr is seven. So array arr will store only first seven characters and it will note store null character.

As we know %s in prinf statement prints stream of characters until it doesn’t get first null character. Since array arr has not stored any null character so it will print garbage value.

**5. What will be the output when you execute the below statements:**

#include<stdio.h>

void main()

{

char arr[11]=”The African Queeen”;

printf(“%s”,arr);

}

**Explanation:**

Size of any character array cannot be less than the number of characters in any string which it has assigned. Size of an array can be equal (excluding null character) or greater than but never less than. So compilation error.

---------------------------------------------------End of Module 3----------------------------------------------------------------------