Module 3

(Arrays, Matrix, Strings, Searching and Sorting)

1. **What is Arrays and Matrix? How to declare and initialize 1-D and 2-D Arrays? Explain with an example.**
2. Develop C Program for the following Problems:
   i) **To display principal diagonal elements of Matrix**
   ii) To Find the sum of two matrices
   ii) **To generate Fibonocci series using arrays**
   iv) To find the sum of array elements
   v) **To find the largest elements of an array**
3. What is searching and sorting? With an example explain linear search , Binary search, Bubble sort and selection sorting Algorithms
4. **Sort the following elements using: i) Selection Sort ii) Bubble Sort**
   89 78 65 90 75 40
5. **Write a C program for the following**
   a. Linear Search
   b. Binary search
   c. Selection sort
   d. Bubble sort
6. **What is String in C? With an example explain any six string handling functions in C**
7. Write a C Program to find the **length of string, copy the content of one string to other string, concatenate two string and compare two strings** without using built in functions

What is arrays? What are the different types of arrays? How to declare and initialize arrays? Explain with an example

Array is a collection of homogeneous elements (ie; the collection of elements with same data types)

Why do we need arrays?

We can use normal variables (a1, a2, a3, ..) when we have a small number of objects, but if we want to store a large number of instances, it becomes difficult to manage them with normal variables. The idea of an array is to represent many instances in one variable.

- Without using arrays: int a1,a2,a3,a4,a5;
- With array: int a[5]; need to remember only one variable name.

Types of an arrays:

- One Dimensional arrays : Known as List Ex: int A[10];
- Multi Dimensional (Ex: Two Dimensional) arrays: Ex: int B[5][5];

How to declare arrays?

- **One Dimensional array:**

  Syntax:

  Datatype Arrayname[Size];

  Ex: int A[5];
  float Bal[5];
  char Name[25];

A					
Indices	0	1	2	3	4

- **Two Dimensional array:**

  Data_type Array_name[RowSize][ColSize];

  Ex: int A[2][3];

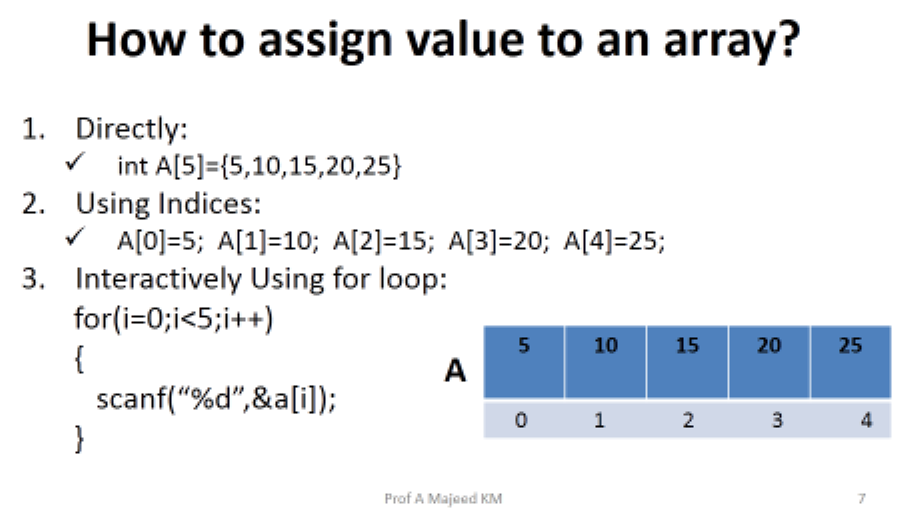
  float Bal[5][10];

A[2][3]

i/j	**0**	**1**	**2**
0	A[0][0]	A[0][1]	A[0][2]
1	A[1][0]	A[1][1]	A[1][2]

How to initialize arrays?

Assigning value to array is known as initialization of array



How to assign value to an array?

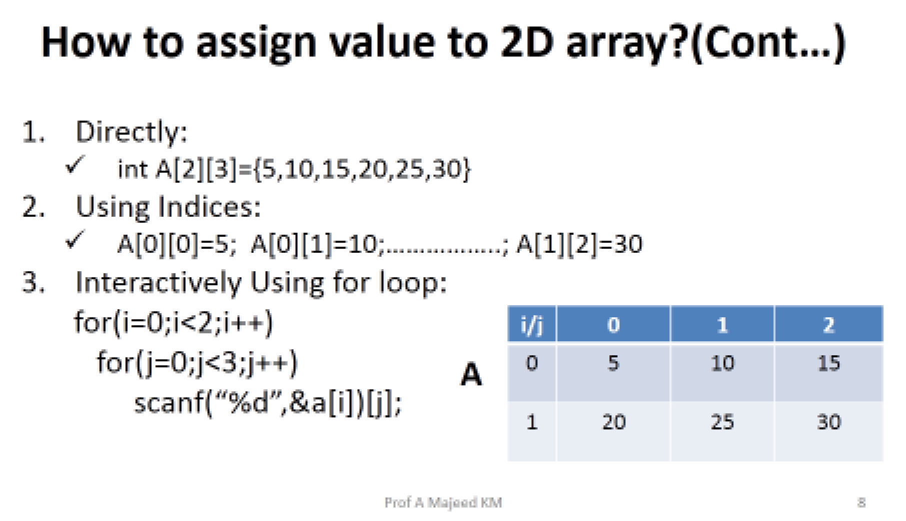
1. Directly:
   - ✓ int A[5]={5,10,15,20,25}
2. Using Indices:
   - ✓ A[0]=5; A[1]=10; A[2]=15; A[3]=20; A[4]=25;
3. Interactively Using for loop:

```
for(i=0;i<5;i++)
{
  scanf("%d",&a[i]);
}
```

A

5	10	15	20	25
0	1	2	3	4

Prof A Majeed KM 7



How to assign value to 2D array?(Cont…)

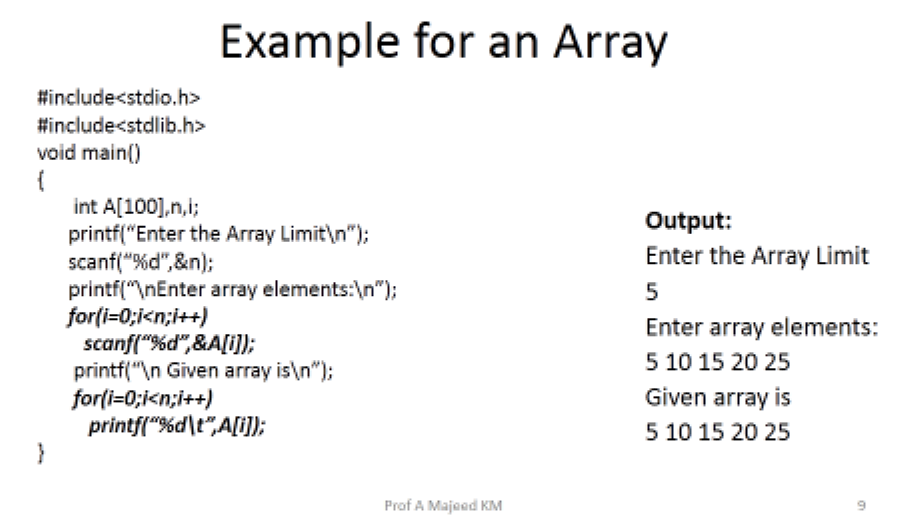
1. Directly:
   - ✓ int A[2][3]={5,10,15,20,25,30}
2. Using Indices:
   - ✓ A[0][0]=5; A[0][1]=10;……………..; A[1][2]=30
3. Interactively Using for loop:

```
for(i=0;i<2;i++)
  for(j=0;j<3;j++)
     scanf("%d",&a[i])[j];
```

A

i/j	0	1	2
0	5	10	15
1	20	25	30

Prof A Majeed KM 8



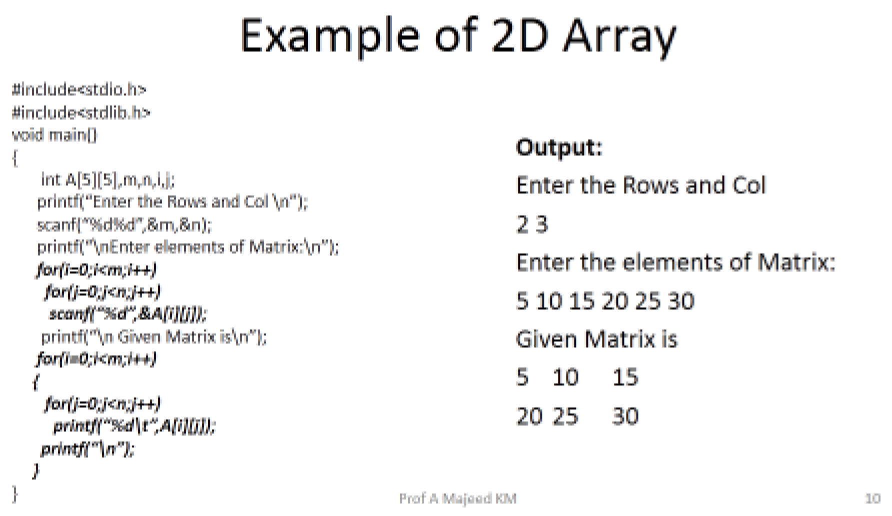
Example for an Array

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
     int A[100],n,i;
    printf("Enter the Array Limit\n");
    scanf("%d",&n);
    printf("\nEnter array elements:\n");
    for(i=0;i<n;i++)
      scanf("%d",&A[i]);
     printf("\n Given array is\n");
     for(i=0;i<n;i++)
       printf("%d\t",A[i]);
}
```

Output:

```
Enter the Array Limit
5
Enter array elements:
5 10 15 20 25
Given array is
5 10 15 20 25
```

Prof A Majeed KM 9



Example of 2D Array

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
    int A[5][5],m,n,i,j;
   printf("Enter the Rows and Col \n");
   scanf("%d%d",&m,&n);
   printf("\nEnter elements of Matrix:\n");
   for(i=0;i<m;i++)
    for(j=0;j<n;j++)
     scanf("%d",&A[i][j]);
    printf("\n Given Matrix is\n");
   for(i=0;i<m;i++)
   {
    for(j=0;j<n;j++)
      printf("%d\t",A[i][j]);
    printf("\n");
   }
}
```

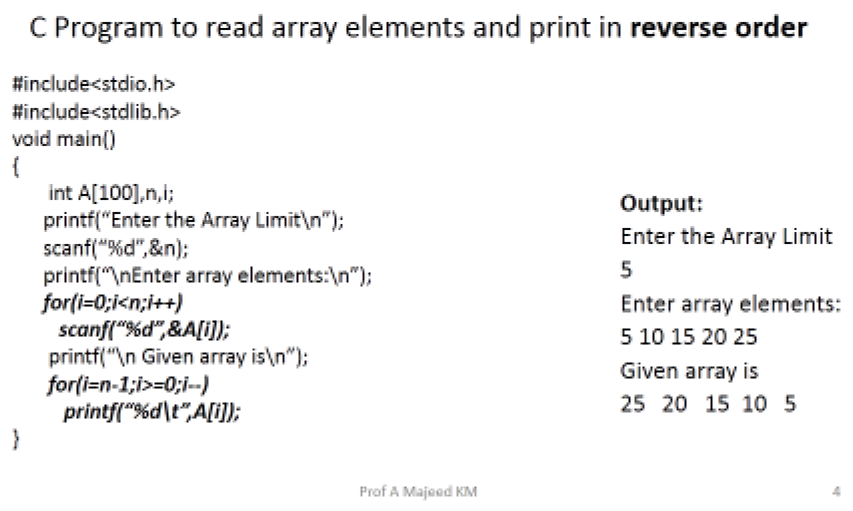
Output:

```
Enter the Rows and Col
2 3
Enter the elements of Matrix:
5 10 15 20 25 30
Given Matrix is
5   10    15
20  25    30
```

Prof A Majeed KM 10

Some Example:

- C Program to read array elements and print in **reverse order**



C Program to read array elements and print in **reverse order**

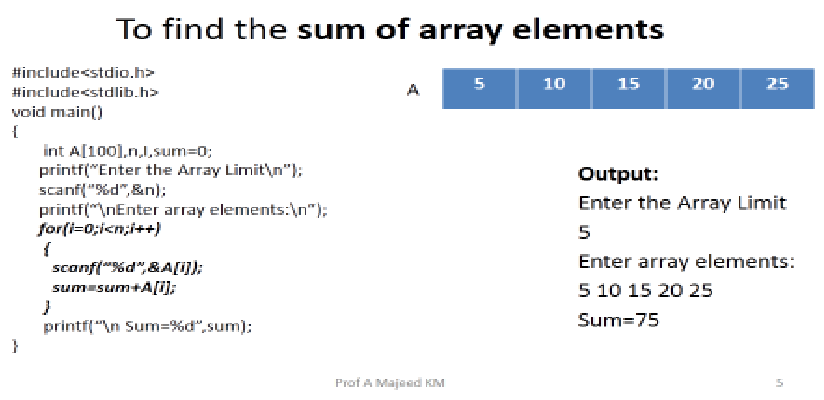
```
#include<stdio.h>
#include<stdlib.h>
void main()
{
     int A[100],n,i;
    printf("Enter the Array Limit\n");
    scanf("%d",&n);
    printf("\nEnter array elements:\n");
    for(i=0;i<n;i++)
      scanf("%d",&A[i]);
     printf("\n Given array is\n");
     for(i=n-1;i>=0;i--)
       printf("%d\t",A[i]);
}
```

Output:

```
Enter the Array Limit
5
Enter array elements:
5 10 15 20 25
Given array is
25  20  15  10  5
```

Prof A Majeed KM 4

- To find the **sum of array elements**



To find the **sum of array elements**

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
     int A[100],n,I,sum=0;
    printf("Enter the Array Limit\n");
    scanf("%d",&n);
    printf("\nEnter array elements:\n");
    for(i=0;i<n;i++)
     {
      scanf("%d",&A[i]);
      sum=sum+A[i];
     }
     printf("\n Sum=%d",sum);
}
```

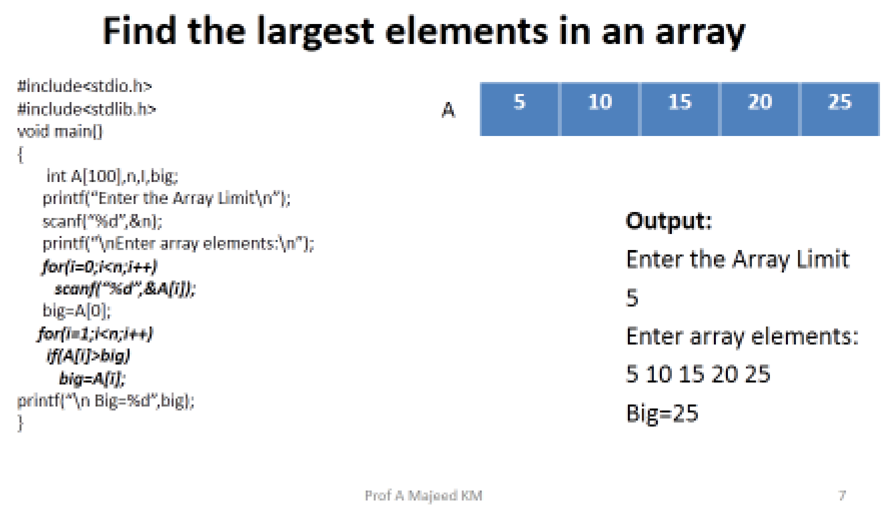
A	5	10	15	20	25

Output:

```
Enter the Array Limit
5
Enter array elements:
5 10 15 20 25
Sum=75
```

Prof A Majeed KM 5

- Find the **largest elements** in an array



Find the largest elements in an array

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
     int A[100],n,I,big;
    printf("Enter the Array Limit\n");
    scanf("%d",&n);
    printf("\nEnter array elements:\n");
    for(i=0;i<n;i++)
      scanf("%d",&A[i]);
    big=A[0];
   for(i=1;i<n;i++)
     if(A[i]>big)
       big=A[i];
printf("\n Big=%d",big);
}
```

A	5	10	15	20	25

Output:

```
Enter the Array Limit
5
Enter array elements:
5 10 15 20 25
Big=25
```

Prof A Majeed KM 7

Program To print transpose of matrix and principle diagonal elements of matrix

```
#include<stdio.h>
void main()
{
      int A[5][5],B[5][5],m,n,i,j;
     printf("Enter the Rows and Col \n");// reading order of matrix
     scanf("%d%d",&m,&n);
     printf("\nEnter elements of Matrix:\n");// matrix initialization
     for(i=0;i<m;i++)
      for(j=0;j<n;j++)
      {
       scanf("%d",&A[i][j]);
       B[j][i]=A[i]j[j];// generating transpose of matrix
      }
      printf("\n Given Matrix is\n");//print matrix
     for(i=0;i<m;i++)
     {
       for(j=0;j<n;j++)
         printf("%d\t",A[i][j]);
      printf("\n");
     }
printf("\n Transpose of  Matrix is\n");// print its transpose
     for(i=0;i<m;i++)
     {
       for(j=0;j<n;j++)
         printf("%d\t",B[i][j]);
      printf("\n");
   }
printf("\n Principle diagonal elements of  Matrix is\n");
   for(i=0;i<m;i++)
   {
     for(j=0;j<n;j++)
      if(i==j) // checking whether its diagonal elememts
       printf("%d\t",A[i][j]);// print diagonal elements
     else
       print"\t");
    printf("\n");
  }
}
```

Output:

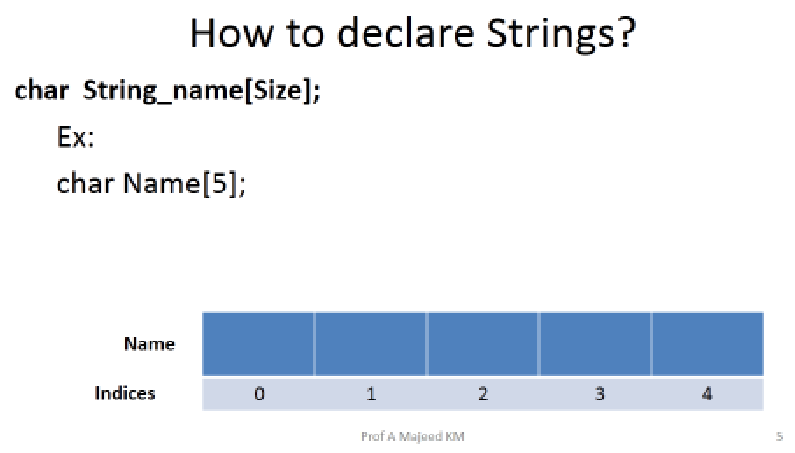
```
Enter the Rows and Col
3 3
Enter elements of Matrix:
1 2 3 4 5 6 7 8 9
 Given Matrix is
1	2	3
4	5	6
7	8	9
 Transpose of  Matrix is
1	4	7
2	5	8
3	6	9
 Principle diagonal elements of Matrix is
1
	5
		9
```

Character Array(Strings):

What is strings? How to declare and initialize strings? Explain with an example

What is string?

- **String is defined as an array of characters**
- **String is terminated with a special character '\0'.**



How to declare Strings?

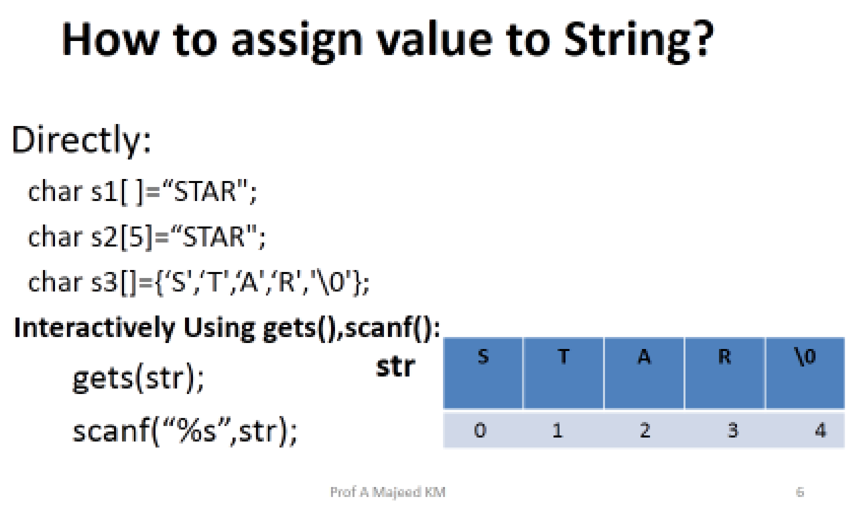
char String_name[Size];

Ex:

char Name[5];

Name					
Indices	0	1	2	3	4

Prof A Majeed KM 5



How to assign value to String?

Directly:

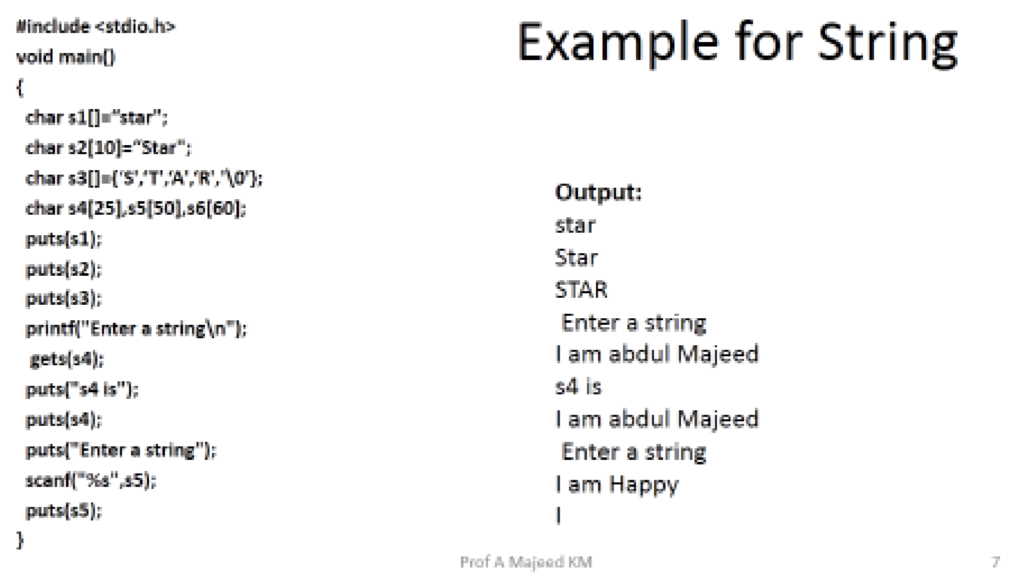
```
char s1[ ]="STAR";
char s2[5]="STAR";
char s3[]={'S','T','A','R','\0'};
```

Interactively Using gets(),scanf():

```
gets(str);
scanf("%s",str);
```

str	S	T	A	R	\0
	0	1	2	3	4

Prof A Majeed KM 6



Example for String

```
#include <stdio.h>
void main()
{
 char s1[]="star";
 char s2[10]="Star";
 char s3[]={'S','T','A','R','\0'};
 char s4[25],s5[50],s6[60];
 puts(s1);
 puts(s2);
 puts(s3);
 printf("Enter a string\n");
  gets(s4);
 puts("s4 is");
 puts(s4);
 puts("Enter a string");
 scanf("%s",s5);
 puts(s5);
}
```

Output:

```
star
Star
STAR
 Enter a string
I am abdul Majeed
s4 is
I am abdul Majeed
 Enter a string
I am Happy
I
```

Prof A Majeed KM 7

What are String Handling Functions/String manipulation functions? Explain with an example OR Demonstrate the use of of strlen(), strcpy(), strcat() strcmp() functions

- **C supports a large number of string handling functions in the standard library "string.h".**
- **#include <string.h> to be used**

Function	Description	Function	Description
strlen()	Calculates the length of string	strcat()	Concatenates(joins) two strings
strcpy()	Copies a string to another string	strcmp()	Compares two string

<u>strlen()</u>: The **strlen()** function calculates the length of a given string. The **strlen()** function is defined in **string.h** header file. It doesn't count null character '\0'.

Syntax:

```
length=strlen(str);
```

Parameter:

- **str:** It represents the string variable whose length we have to find.

Return: This function returns the length of string passed.

Ex: str="VTU";
 len=strlen(str); // ie;now len=3

<u>strcat()</u>: The strcat() function will append a copy of the source string to the end of destination string. The strcat() function takes two arguments:

1) dest
2) src

It will append copy of the source string in the destination string. **The terminating character at the end of dest is replaced by the first character of src .**

Return value: The strcat() function returns dest, the pointer to the destination string.

Syntax:

```
strcat(dest, src);
```

```
Ex:  dest="VTU";
     src="Belegavi";
     strcat(dest,src);// ie; now dest=VTUBelegavi
```

<u>strcpy</u>(): strcpy() is a standard library function in Cand is used to copy one string to another. In C it is present in **string.h** header file

Syntax:

```
strcpy(dest, src);
```

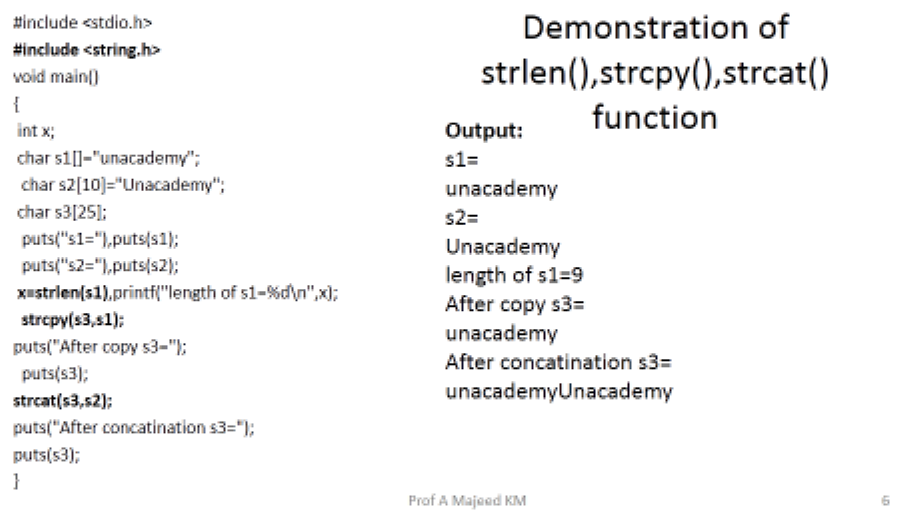
- **dest**: Pointer to the destination array where the content is to be copied.
- **src:** string which will be copied.

After copying the source string to the destination string, the strcpy() function returns a pointer to the **destination string**.

Ex:

strcpy(city,"Delhi");// ie; Delhi is copied to the string city

Below programs illustrate the strlen(),strcpy() and strcat() function in C:



Demonstration of strlen(),strcpy(),strcat() function

```
#include <stdio.h>
#include <string.h>
void main()
{
 int x;
 char s1[]="unacademy";
  char s2[10]="Unacademy";
 char s3[25];
  puts("s1="),puts(s1);
  puts("s2="),puts(s2);
 x=strlen(s1),printf("length of s1=%d\n",x);
  strcpy(s3,s1);
puts("After copy s3=");
  puts(s3);
strcat(s3,s2);
puts("After concatination s3=");
puts(s3);
}
```

Output:

```
s1=
unacademy
s2=
Unacademy
length of s1=9
After copy s3=
unacademy
After concatination s3=
unacademyUnacademy
```

Prof A Majeed KM 6

strcmp(): strcmp() is a built-in library function and is declared in **<string.h>** header file. This function takes two strings as arguments and compare these two strings lexicographically.
Syntax::

```
f=strcmp(str1,str2);
```

In the above prototype, function srtcmp takes two strings as parameters and returns an integer value based on the comparison of strings.

- strcmp() compares the **two strings lexicographically** means it starts comparison character by character starting from the first character until the characters in both strings are equal or a NULL character is encountered.
- If first character in both strings are equal, then this function will check the second character, if this is also equal then it will check the third and so on
- This process will be continued until a character in either string is NULL or the characters are unequal.
- If strings are **equal** then it will **return 0 , otherwise nonzero**

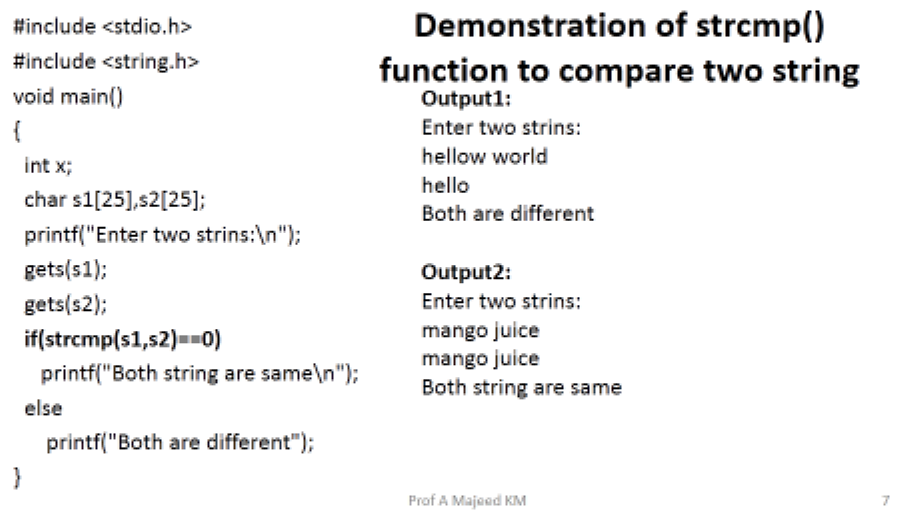
Below programs illustrate the strcmp() function in C:

Ex:

strcmp(name1,name2);

strcmp(name1,"John");

strcmp("RAM","ROM");



Demonstration of strcmp() function to compare two string

```
#include <stdio.h>
#include <string.h>
void main()
{
 int x;
 char s1[25],s2[25];
 printf("Enter two strins:\n");
 gets(s1);
 gets(s2);
 if(strcmp(s1,s2)==0)
   printf("Both string are same\n");
 else
    printf("Both are different");
}
```

Output1:

```
Enter two strins:
hellow world
hello
Both are different
```

Output2:

```
Enter two strins:
mango juice
mango juice
Both string are same
```

Prof A Majeed KM 7

Write a C Program to find the **length of string, copy the content of one string to other string, concatenate two string and compare two strings** ***without using built in functions***

Program:

```
#include <stdio.h>
char s1[20],s2[20],s3[20];
/* Function to find string Length*/
int stringlen(char s1[])
{
int len=0,i;
for(i=0;s1[i]!='\0';i++)
        len++;
return len;
}
/*Function to Compare string*/
void stringcmp(char s1[],char s2[])
{
int i=0,f=0;
while(s1[i]!='\0'||s2[i]!='\0')
{
if(s1[i]!=s2[i])
{
        f=1; //srings are different
        break;
}
i++;
}
if(f==0)
        printf (" Strings are Equal\n"); Flow chart : for stringcmp()
else
        printf (" Strings are Different\n");
}
/*Function to Concatinate strings*/
void stringcat(char a[],char b[])
{
int j,i;
i=stringlen(a);
for(j=0;b[j]!='\0';j++,i++)
        a[i]=b[j];
a[i]='\0';
puts(a);
}
```

//Function to copy string

```
void stringcopy(char a[],char b[])
{
int i;
for(i=0;b[i]!='\0';i++)
a[i]=b[i];
a[i]='\0';
puts(a);
}
```

/* Main function*/

```
int main()
{
int n;
printf("Enter string1\n") ;
gets(s1); // Reading string1
printf("Enter string2\n") ;
gets(s2); //Reading string2
printf("string1 is:\n");
puts(s1); //Display String1
printf("string2 is: \n");
puts(s2); //Display String2
puts("Finding String length\n");
n=stringlen(s1); //Function call to find string length
printf ("length of string1=%d\n",n);
n=stringlen(s2); //Function call to find string length
printf ("length of string2=%d\n",n);
puts(" Comparing string1 and string2\n");
stringcmp (s1,s2);//Function call to Compare strings
puts("After Concatenation of string1 and string2\n");
stringcat(s1,s2);//Function call for concatenation of string
puts("after copying string1 to string3 , string3 is\n") ;
stringcopy(s3,s1);//Function call to copy the content of s1 to s3
return 0;
}
```

Basic Algorithms: Searching and Sorting

Searching:

Searching is the process of finding a given value position in a list of values. It decides whether a search key is present in the data or not. It is the algorithmic process of finding a particular item in a collection of items.

- Linear Search
- Binary Search

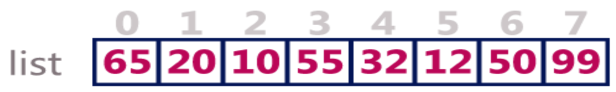
Linear Search:

It is one of the searching technique which is used to find whether a given number is present in an array and if it is present then at what location it occurs. It is also known as **sequential search**. It is straightforward and works as follows: We keep on comparing each element with the element to search until it is found or the list ends.

Linear search is implemented using following steps...

- **Step 1 -** Read the search element from the user.
- **Step 2 -** Compare the search element with the first element in the list.
- **Step 3 -** If both are matched, then display "Given element is found!!!" and terminate the function
- **Step 4 -** If both are not matched, then compare search element with the next element in the list.
- **Step 5 -** Repeat steps 3 and 4 until search element is compared with last element in the list.
- **Step 6 -** If last element in the list also doesn't match, then display "Element is not found!!!" and terminate the function.

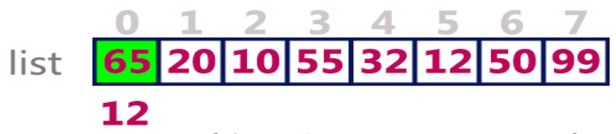
- **Example**
- Consider the following list of elements and the element to be searched...



search element **12**

Step 1:

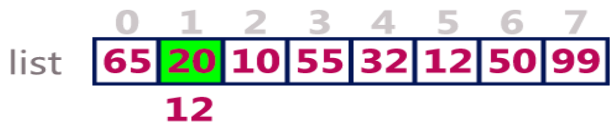
search element (12) is compared with first element (65)



Both are not matching. So move to next element

Step 2:

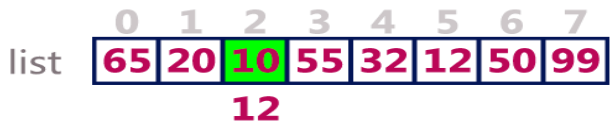
search element (12) is compared with next element (20)



Both are not matching. So move to next element

Step 3:

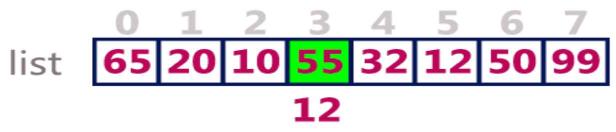
search element (12) is compared with next element (10)



Both are not matching. So move to next element

Step 4:

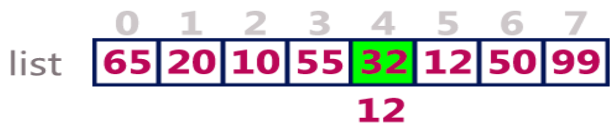
search element (12) is compared with next element (55)



Both are not matching. So move to next element

Step 5:

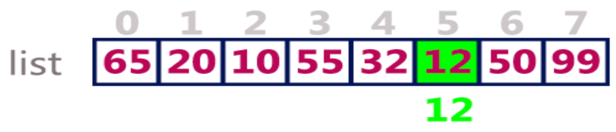
search element (12) is compared with next element (32)



Both are not matching. So move to next element

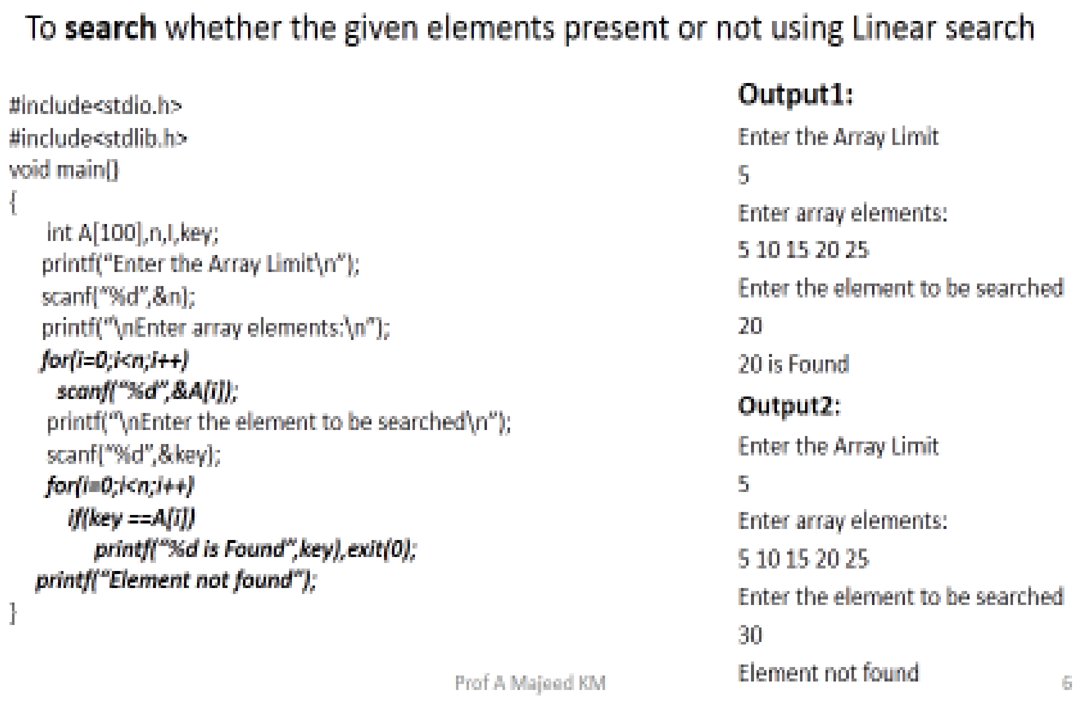
Step 6:

search element (12) is compared with next element (12)



Both are matching. So we stop comparing and display element found at index 5.

C Program for linear search(Demonstration of linear search)



To **search** whether the given elements present or not using Linear search

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
    int A[100],n,I,key;
   printf("Enter the Array Limit\n");
   scanf("%d",&n);
   printf("\nEnter array elements:\n");
   for(i=0;i<n;i++)
     scanf("%d",&A[i]);
    printf("\nEnter the element to be searched\n");
    scanf("%d",&key);
    for(i=0;i<n;i++)
      if(key ==A[i])
          printf("%d is Found",key),exit(0);
  printf("Element not found");
}
```

Output1:

```
Enter the Array Limit
5
Enter array elements:
5 10 15 20 25
Enter the element to be searched
20
20 is Found
```

Output2:

```
Enter the Array Limit
5
Enter array elements:
5 10 15 20 25
Enter the element to be searched
30
Element not found
```

Prof A Majeed KM 6

Binary Search:

Binary Search is a type of searching algorithm. A searching algorithm means you find an item with a particular value in a sorted sequence. Binary search is a type of searching algorithm which finds an item by repeatedly halving the search space.

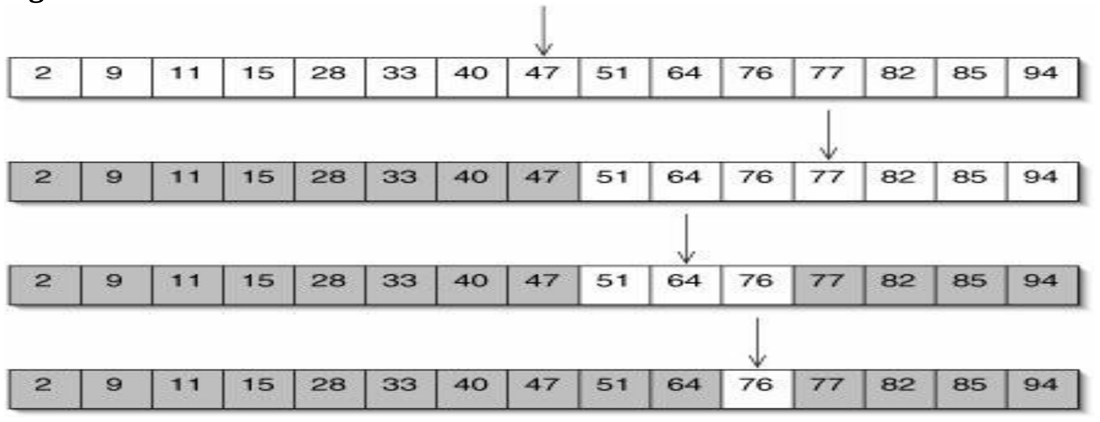
Explanation of Binary Search:

Binary Search: Steps on how it works:

To find the index of element key with a certain value:

1. Start with an array sorted in Ascending/descending order.
2. In each step: Pick the middle element of the array mid and compare it to key. If element values are equal, then return index of mid. If key is greater than mid, then key must be in right subarray. If key is less than mid, then key must be in the left subarray.
3. Repeat those steps on new subarray.

Search whether key =76 present or not in the array given below using binary search algorithm



Yes key found at position 11

C Program for Binary Search:

```
#include<stdio.h>
#include<stdlib.h>
main()
{
   int arr[50],i,n,key,first,last,mid;           // Declaring array and other variables
    printf("Enter size of array:");
   scanf("%d",&n);
   printf("\nEnter array element(ascending order)\n");
    for(i=0;i<n;++i)
      scanf("%d",&arr[i]);//Reading array elements
    printf("\nEnter the element to search:");
   scanf("%d",&key);
 //Binary search implementation
   first=0;
   last=n-1;
   while(first<=last)         // searching in arrray from first to last
   {
      mid=(first+last)/2;  // calculate mid
       if(key==arr[mid])                 // key element found or not
      {
        printf("\nElement found at position %d",mid);
        exit(0);     //Terminate the program
      }
      else if(key>arr[mid])
            first=mid+1;     // searching only right side of mid
      else
            last=mid-1;       //searching only left side of mid
   }
    printf("\nElement not found");
   }
```

Output:

```
Enter size of array: 3
Enter array element (ascending order)
1 2 3
Enter the element to search: 6
Element not found
------------------------------------------------------------------------------------------------------------
Enter size of array: 5
Enter array element (ascending order)
1 2 33 41 55
Enter the element to search: 41
Element found at position 3
```

Differences between linear search binary search:

- A linear search scans one item at a time, without jumping to any item. In contrast, binary search cuts down your search to half as soon as you find the middle of a sorted list.
- Time taken to search elements keep increasing as the number of elements is increased when searching through linear process. But binary search compresses the searching period by dividing the whole array into two half.
- Linear search does the sequential access whereas Binary search access data randomly.
- Input data needs to be sorted in Binary Search and not in Linear Search.
- Binary search is better and quite faster than linear search.
- Binary search is efficient for the larger array. If the amount of data is small, then linear search is preferable because this searching process is fast when data is small.

Sorting:

Arranging element in a particular order is known as sorting. There are many technique or algorithms used for sorting. Bubble sort and selection sort are the Examples for sorting algorithms.

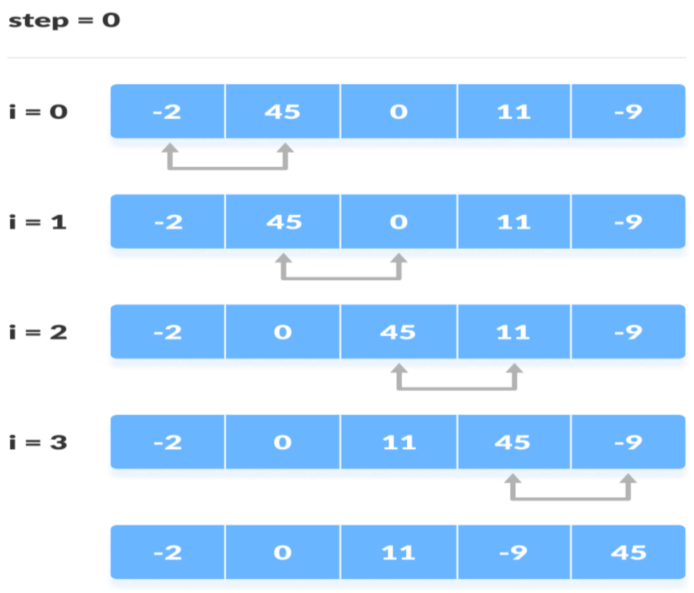
Bubble sort:

Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order.

Note:Refer class note for complete demonstration of bubble sort

How Bubble Sort Works?

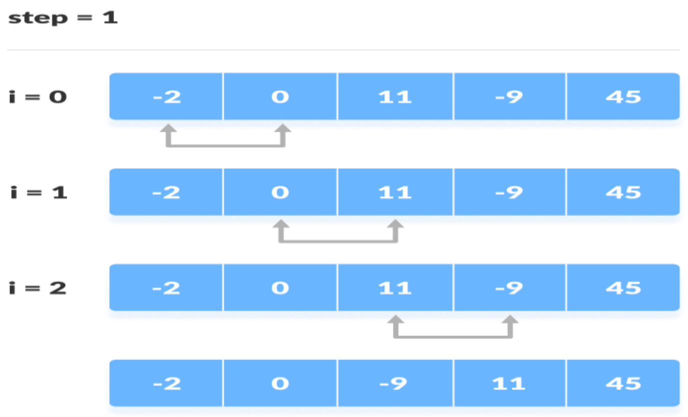
1. Starting from the first index, compare the first and the second elements. If the first element is greater than the second element, they are swapped.
   Now, compare the second and the third elements. Swap them if they are not in order.
   The above process goes on until the last element.

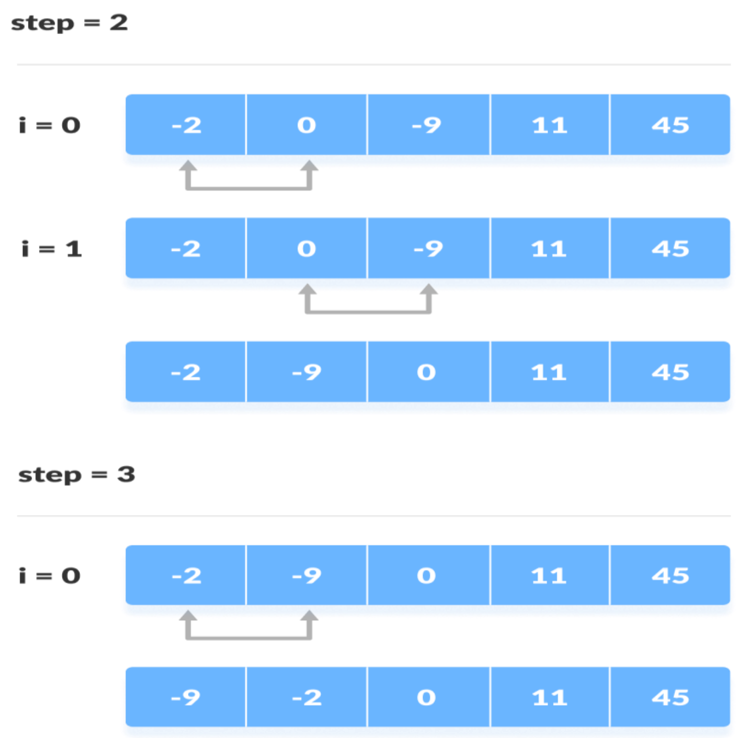


2. The same process goes on for the remaining iterations. After each iteration, the largest element among the unsorted elements is placed at the end.

   In each iteration, the comparison takes place up to the last unsorted element.

   The array is sorted when all the unsorted elements are placed at their correct positions.





Program: Bubble Sort

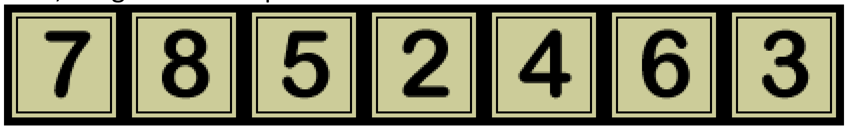
```
#include <stdio.h>
main()
{
 int A[20], n, i, j, temp;
  printf("Enter number of elements\n");
 scanf("%d", &n);
  printf("Enter %d integers\n", n);
  for (i = 0; i < n; i++)
            scanf("%d", &A[i]);
/* Bubble sort*/
 for (i = 0 ; i< n - 1; i++)
 {
            for (j = 0 ; j < n - i - 1; j++)
           {
                        if (A[j] > A[j+1])
                       {
                                  temp  = A[j];
                                   A[j]   = A[j+1];
                                   A[j+1] = temp;
                       }
            }
 }
 printf("Sorted list in ascending order:\n");
  for (i = 0; i < n; i++)
           printf("%d\t", A[i]);

}
```

Output:

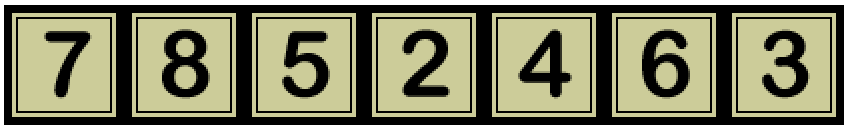
```
Enter number of elements
5
Enter 5 integers
6 2 5 0 1
Sorted list in ascending order:
0     1     2     5      6
```

Selection Sort:

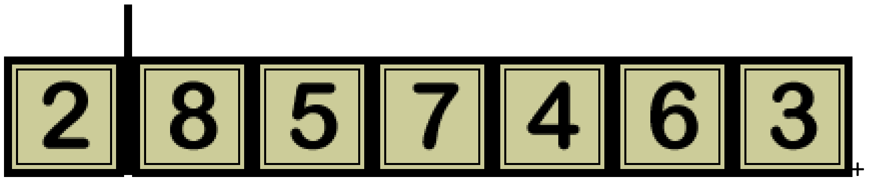
First, we give the computer a list of unsorted numbers and store them in an array of memory cells.



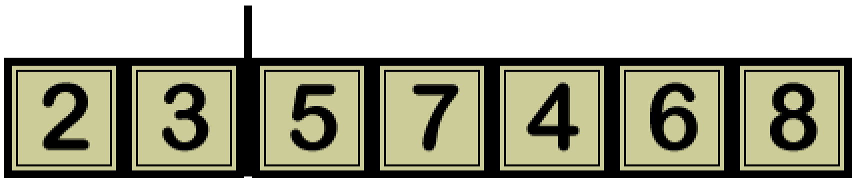
To begin the sort, the computer divides the sorted and unsorted sections of the list by placing a marker before the first number. To sort the numbers, the computer will repeatedly search the unsorted section for the smallest number, swap this number with the first number in the unsorted section, and update the sort marker.



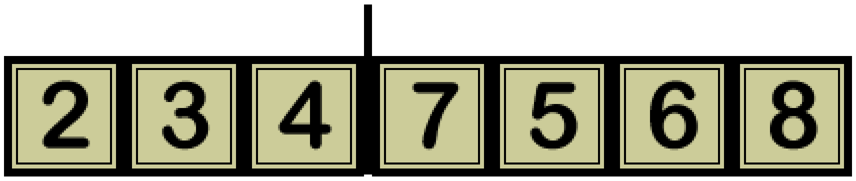
To find the smallest number in the unsorted section, the computer must make six comparisons: (7 < 8), (7 > 5), (5 > 2), (2 < 4), (2 < 6), and (2 > 3). After these comparisons, the computer knows that 2 is the smallest number, so it swaps this number with 7, the first number in the unsorted section, and advances the sort marker.



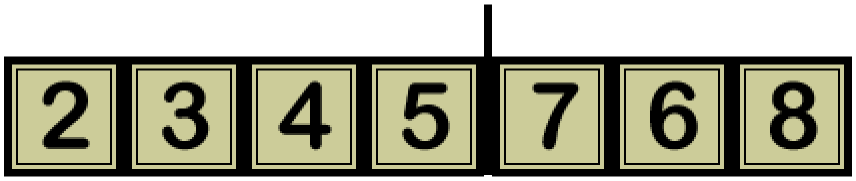
Now five more comparisons are needed to find the smallest number in the unsorted section: (8 > 5), (5 < 7), (5 > 4), (4 < 6), and (4 > 3). After these comparisons, the computer swaps 3, the smallest number in the unsorted section, with 8, the first number in the unsorted section, and advances the sort marker.



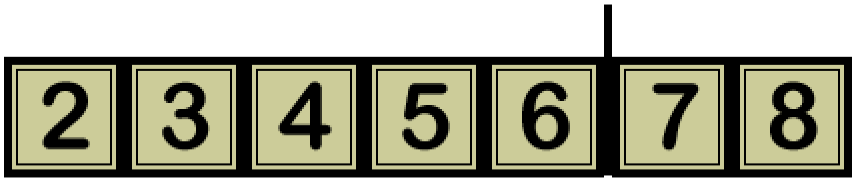
This time four comparisons are needed to determine that 4 is the smallest number in the unsorted section: (5 < 7), (5 > 4), (4 < 6), and (4 < 8). After these comparisons, the computer swaps 4 with 5 and then advances the sort marker.



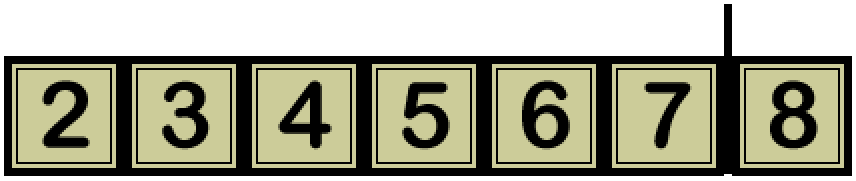
After three more comparisons, the computer identifies 5 as the smallest unsorted number: (7 > 5), (5 < 6), and (5 < 8). Then the computer swaps 5 with 7 and advances the sort marker.



This time only two comparisons are needed to determine that 6 is the smallest number: (7 > 6) and (6 < 8). After these two comparisons, the computer swaps 6 with 7 and then advances the sort marker.



Now we only need a single comparison to find the right position for 7: (7 < 8). Since 7 is the smallest number and it is also the first number in the unsorted section, the computer does not need to swap this number. It only needs to advance the sort marker. Now there is only one number in the unsorted section, so the list of numbers is sorted and the Selection Sort algorithm is complete.



Note: Refer Class note for more examples

Program : Selection Sort

```
#include <stdio.h>
main()
{
 int A[20], n, i, j, temp,min;
  printf("Enter number of elements\n");
 scanf("%d", &n);
  printf("Enter %d integers\n", n);
  for (i = 0; i < n; i++)
           scanf("%d", &A[i]);
//Selection sort
  for (i = 0; i < n-1; i++)
   {
       // Find the minimum element in unsorted array
       min = i;
       for (j = i+1; j < n; j++)
       if (arr[min] >arr[j])
             min = j;
      //Swap if required
      If(min!=i)
        {
             temp=A[i];
            A[i]=A[min];
            A[min]=temp;
         }
 }
 printf("Sorted Array using selection sort\n");
  for (i = 0; i < n; i++)
           printf("%d\t", A[i]);
}
```

Output:

```
Enter number of elements
5
Enter 5 integers
6 2 5 0 1
Sorted Array using selection sort
0      1      2      5       6
```

Difference between bubble sort and selection sort:

- Bubble sort is simplest sorting technique compare to selection sort
- In the bubble sort, each element and its adjacent element is compared and swapped if required. On the other hand, selection sort works by selecting the element and swapping that particular element with the last element. The selected element could be largest or smallest depending on the order i.e., ascending or descending. to selection sort
- Bubble sort is slower than selection sort
- Selection sort has improved efficiency compared to bubble sort