

# Arrays and strings

## **Outline**

- 4.1 Introduction**
- 4.2 Arrays**
- 4.3 Declaring Arrays**
- 4.4 Examples Using Arrays**
- 4.5 Passing Arrays to Functions**
- 4.6 Sorting Arrays**
- 4.7 Case Study: Computing Mean, Median and Mode Using Arrays**
- 4.8 Searching Arrays: Linear Search and Binary Search**
- 4.9 Multiple-Subscripted Arrays**



you will be able to

- discuss how to group related data items together with the use of arrays.
- explain single dimension, two dimensional and multi-dimensional arrays
- state what are strings
- use various standard string library functions and in programs
- create a two dimensional array of strings
- develop a number of programs using arrays.



# Arrays

- So far we have worked primarily with primitive variables
  - One characteristic of primitive variables is that they hold one value at a time.
    - `int x = 100` – can only hold one integer value at a time (100 at this time)
- Imagine that you want to hold the names of 100 people at a time.
  - What would you have to do?
    - Declare 100 variables, one for each name.
  - Better solution: Use Arrays.

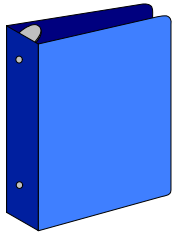


## 4.1 Introduction

- Arrays
- An array is a collection of data elements that are of the same type and share a **common name**
- (e.g., a collection of integers, collection of characters, collection of doubles).
- Array Applications:
- Given a list of test scores, determine the maximum and minimum scores.
- Read in a list of student names and rearrange them in alphabetical order (sorting).



# Array Declaration



- Syntax:

**<type> <arrayName> [<array\_size>]**

- The array elements are all values of the type **<type>**.
- The size of the array is indicated by **<array\_size>**, the number of elements in the array.
- **<array\_size>** must be an **int** constant or a constant expression. Note that an array can have multiple dimensions.



# Array Declaration

```
// array of 10 uninitialized ints
```

```
int Ar[10];    // array of 10 ints
```

Array name is Ar

Size is 10



# Subscripting

- To access an individual element we must apply a subscript to array named **Ar**.

- A subscript is a bracketed expression.
  - The expression in the brackets is known as the index.
- First element of array has index 0.

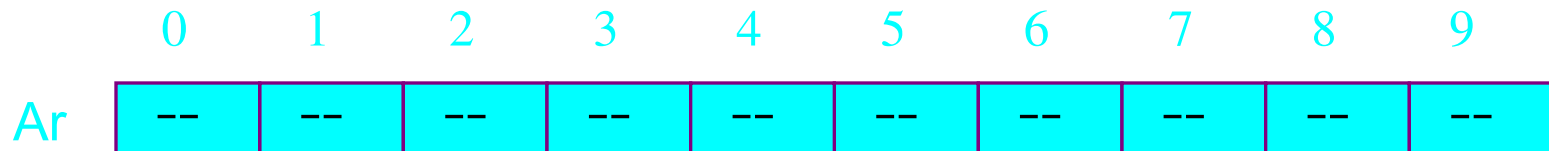
**Ar[0]**

- Second element of array has index 1, and so on.

**Ar[1], Ar[2], Ar[3], ...**

- Last element has an index one less than the size of the array.

**Ar[9]**



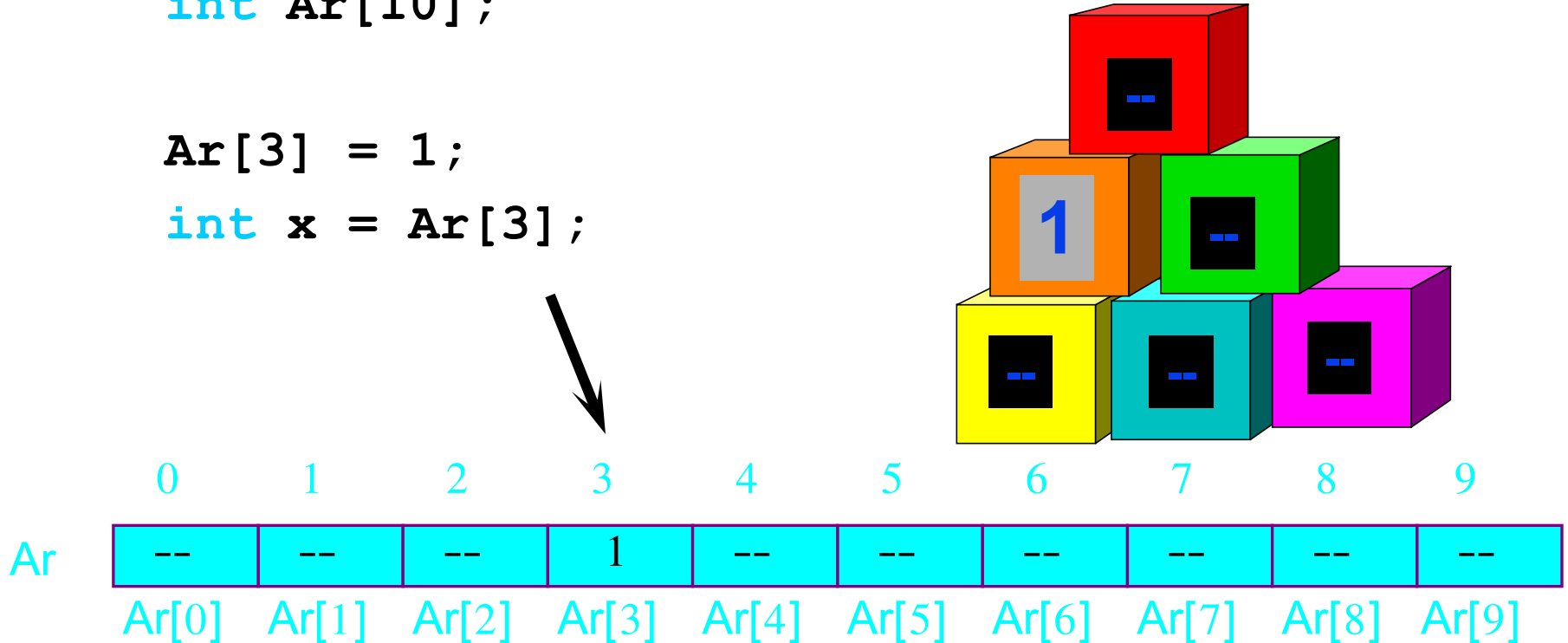
# Subscripting

```
// array of 10 uninitialized ints
```

```
int Ar[10];
```

```
Ar[3] = 1;
```

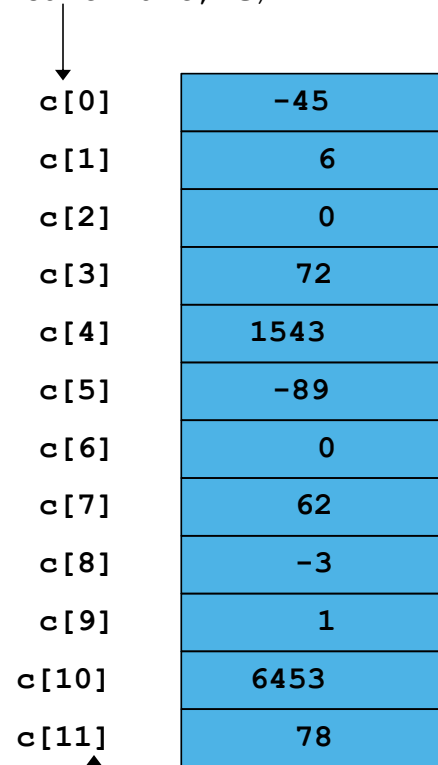
```
int x = Ar[3];
```





## 4.2 Arrays

Name of array (Note that all elements of this array have the same name, **c**)



<b>c[0]</b>	<b>-45</b>
<b>c[1]</b>	<b>6</b>
<b>c[2]</b>	<b>0</b>
<b>c[3]</b>	<b>72</b>
<b>c[4]</b>	<b>1543</b>
<b>c[5]</b>	<b>-89</b>
<b>c[6]</b>	<b>0</b>
<b>c[7]</b>	<b>62</b>
<b>c[8]</b>	<b>-3</b>
<b>c[9]</b>	<b>1</b>
<b>c[10]</b>	<b>6453</b>
<b>c[11]</b>	<b>78</b>

Position number of the element within array **c**



- Initializing arrays
  - For loop
    - Set each element  
for (i=0; i<10; i++)  
{ scanf(“%d”, &marks[i]); }
  - Initializer list
    - Specify each element when array declared  
**int n[ 5 ] = { 1, 2, 3, 4, 5 };**
    - If not enough initializers, rightmost elements 0
    - If too many syntax error
  - To set every element to same value  
**int n[ 5 ] = { 0 };**
  - If array size omitted, initializers determine size  
**int n[] = { 1, 2, 3, 4, 5 };**
    - 5 initializers, therefore 5 element array



- Array size

- Can be specified with constant variable (**const**)
  - **const int size = 5;**
  - #define size 5**
- Constants cannot be changed
- Constants must be initialized when declared
- Also called named constants or read-only variables
- Array DECLARATION:
  - **int marks[size];**

Or

**int marks [size ]= {1, 3, 2, 48, 34)**



- An array is a collection of similar elements
- - The individual values in the array are called as elements
- - The first element in the array is numbered 0
- - Each individual element in an array is accessed by its position in the array.
- This position is called the array subscript or index number.
- - The array has to be declared for type and dimension before it can be used
- - All the elements in an array are stored in contiguous memory locations



# Program to find average of 5 numbers

```
main()
{
int num[5];
int sum, i;
float avg;
sum = 0;
for(i =0; i< 5;i++)
{
scanf("%d", &num[i]);
sum = sum + num[i];
}
avg = sum/5;
printf("The average of numbers is %7.2", avg);
}
```



# To find the smallest number in an array

```
#define MAX 20
main()
{
int i, n, min, num[MAX];
printf("\nHow many elements do you wish to enter ? :");
scanf("%d", &n);
printf("Enter elements of array :\n");
for(i=0; i < n; i++)
scanf("%d", &num[i]);
min = num[0];
for (i = 1; i < n; i++)
{
if(num[i] < min)
min = num[i];
}
printf("\nThe smallest element of the array is : %d", min); }
```



# Program to input numbers into array a and then copy them to array b

```
main()
{
int i, a[10], b[10];
for (i = 0; i < 10; i++)
{
printf("Enter element:");
scanf("%d", & a[i]);
printf("\n");
}
for (i = 0; i < 10; i++)
b[i] = a[i];
for (i = 0; i < 10; i++)
printf("%d\t%d\n", a[i], b[i]);
}
```



- To sort  $n$  numbers





```

main()
{
int i, j, temp;
float num[50];
printf("\nEnter elements of array :");
for (i=0; i<5; i++)
scanf("%f", &num[i]);
for(i = 0; i< 4; i++)
{
for(j = i+1; j<5; j++)
{
if(num[i] > num[j])
{
temp = num[i];
num[i] = num[j];
num[j] = temp;}
}
}
}

```

```

printf("\nThe sorted array :\n");
for(i=0;i<5;i++)
printf("%6.2f\n", num[i]);
}

```



## Homework

- 1 a. Enter elements into an array of type **float**. Determine how many of them are positive and how many are negative. Assume an array size of 15.
- b. Modify the above program and write functions to
  - i. store all the positive numbers in a separate array and all negative numbers in a different array.
  - ii. Print the sum of elements of these new arrays



## Two dimensional Arrays

- possible to declare arrays with two or more dimensions.
- A two dimensional array is also called a **matrix**.
- Such a matrix or a **table** can be stored in a two dimensional array
- Subject marks of 5 students in 3 subjects

Student_Id	Sub1	Sub2	Sub3
1100	40	50	67
2100	80	34	56
1330	90	98	89
1331	76	76	76
1485	80	70	65



## Declaration of 2-d arrays

- The two dimensional array can be declared as follows :
- type array\_name[row\_size][col\_size];*  
eg:- *studnt\_marks[5][3]*

	Column0	Column1	Column2	Column3
Row0	[0][0] 1100	[0][1] 40	[0][2] 50	[0][3] 67
Row1	[1][0] 2100	[1][1] 80	[1][2] 34	[1][3] 56
Row2	[2][0] 1330	[2][1] 90	[2][2] 98	[2][3] 89
Row3	[3][0] 1331	[3][1] 76	[3][2] 76	[3][3] 76
Row4	[4][0] 1485	[4][1] 80	[4][2] 70	[4][3] 65



## Initialising Two Dimensional Arrays :

- can also be initialised like the one dimensional arrays with their declaration followed by the list of values of the elements
- For example :

```
int arr[2][3] = {10,5,3,15,20,25};
```

or

```
int arr[2][3] = {{10, 5, 3}, {15, 20, 25}};
```

or

```
int arr[2][3] =  
{  
  {10,5,3},  
  {15,20,25}  
};
```



- when initialising a two dimensional array the first dimension i.e. the **row** is optional, however the second dimension the **column** is a must.

```
int arr[][3] = {10,5,3,15,20,25};
```

Write a program to calculate sum of elements in rows of the array



```

main()
#define ROWS 50
#define COLS 50
main()
{
int arr1[ROWS][COLS], i, j, rows,
cols, sum;
printf("\n Enter number of rows :");
scanf("%d",&rows);
printf("\nEnter number of columns :");
scanf("%d", &cols);
for(i=0; i< rows; i++)
{
for(j = 0; j < cols; j++)
{
printf("\nEnter value :");
scanf("%d", &arr1[i][j]);
}
}
}

```

**p**

```

for(i=0; i< rows; i++)
{
sum = 0;
for(j = 0; j < cols; j++)
{
sum = sum + arr1[i][j];
}
printf("%d\t", sum);
printf("\n");
}
}

```



## Multidimensional arrays.

- C allows arrays of more than 2 dimensions. Such arrays are multidimensional arrays.
- The exact limit of the number of dimensions is dependant on the compiler.
- Multidimensional arrays are rarely required.
- **int** arr1 [2][3][4];
- A three dimensional array can be considered to be a two dimensional array in another array.
- The outermost array is an array which has two elements where each element itself is a two dimensional array whose dimensions are [3][4].





```
int arr1 [2][3][4] = {  
    {  
        {1, 1, 1, 1},  
        {2, 2, 2, 2},  
        {3, 3, 3, 3}  
    },  
    {  
        {5, 5, 5, 5},  
        {6, 6, 6, 6},  
        {7, 7, 7, 7},  
    }  
};
```



# Strings

- Strings
  - Arrays of characters enclosed within double quotes
    - All strings end with **null** (' \0 ')
    - Examples
      - **char string1[] = "hello";**
        - **Null** character implicitly added
        - **string1** has 6 elements
      - **char string1[] = { 'h', 'e', 'l', 'l', 'o', '\0' };**
    - Subscripting is the same
      - String1[ 0 ]** is 'h'
      - string1[ 2 ]** is 'l'



## Which is correct? Why?

1. `char color[3] = "RED";`
  2. `char color[] = "RED";`
- 1 can be corrected as
  - `char color[4] = "RED";`



## Reading and Writing Strings :

```
main()
{
char name[15];
printf("Enter your name :");
scanf("%s", name);
printf("Good Morning %s", name);
}
```

```
char name[ 80 ] ;
scanf ( "  %[ ^\n ] " , name ) ;
```



# Reading and Writing Strings : gets and puts

- The general form of a string variable is”

*char string\_name[size];*

**Eg: –           char name[ 15 ] ;**

```
main()
{
char name[25];
printf(“Enter your name :”);
gets(name);
puts(“Good Morning”);
puts(name);
}
```



# getchar() and putchar()

Program to read a line with the help  
of getchar()\*/

```
main()
{
char ch1, line_1[81];
int i;
printf("Enter a line of text. Press
Enter to end\n");
i = 0;
while ((ch1 = getchar()) != '\n')
{
line_1[i] = ch1;
i = i + 1;
}
```

```
line_1 [i] = '\0';
i = 0;
while((ch1 = line_1[i]) !='\0')
{
putchar(ch1);
i++;
}
}
```



# Additional programs for string manipulation

```
//copy one string to another
{
char str1[20], str2[20];
int i;
printf("Enter string :");
scanf("%s", str1);
for(i = 0;str1[i] !='\0'; i++)
str2[i] = str1[i];
str2[i] = '\0';
printf("\nThe copied string is %s", str2);
printf("\nThe length of the string is : %d", i);
}
```



```
/* String concatenation Program */  
main()  
{  
char str1[50], str2[25];  
int i, j ;  
printf("\nEnter first string :");  
scanf("%s",str1);  
printf("\nEnter second string :");  
scanf("%s", str2);  
i = 0;  
while(str1 [i] !='\0')  
i = i + 1;  
for(j = 0; str2[j] != '\0';j++)  
{  
str1[i] = str2[j];  
i = i + 1;  
}  
str1[i] = '\0';  
printf("\nConcatenated string is %s", str1);  
}
```





# String Handling functions

- C supports a large number of string handling functions in the standard library "string.h".

Function	Work of Function
<code>strlen()</code>	Calculates the length of string
<code>strcpy()</code>	Copies a string to another string
<code>strcat()</code>	Concatenates(joins) two strings
<code>strcmp()</code>	Compares two string
<code>strlwr()</code>	Converts string to lowercase
<code>strupr()</code>	Converts string to uppercase



- 1.strcat()- concatenates two given strings
- strcat ( str2, str1 ); – str1 is concatenated at the end of str2.

```
char str1[] = "This is ",
```

```
str2[] = "a pen";
```

```
strcat(str1,str2);
```

```
2. char a[20]="Program";
```

```
char b[20]={'P','r','o','g','r','a','m','\0'};
```

```
Strlen(a) ; //7
```

```
Strlen(b) ; //7
```



#### 4.strcpy()

```
char str1[10]= "awesome";
```

```
char str2[10]; char str3[10];
```

```
strcpy(str2, str1);
```

#### 5.strcmp()

### Return Value from strcmp()

Return Value	Remarks
0	if both strings are identical (equal)
negative	if the ASCII value of first unmatched character is less than second.
positive integer	if the ASCII value of first unmatched character is greater than second.



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

int main()
{
    char str1[] = "abcd", str2[] = "abCd", str3[] = "abcd";
    int result;

    // comparing strings str1 and str2
    result = strcmp(str1, str2);
    printf("strcmp(str1, str2) = %d\n", result);

    // comparing strings str1 and str3
    result = strcmp(str1, str3);
    printf("strcmp(str1, str3) = %d\n", result);

    return 0;
}
```

Output

```
strcmp(str1, str2) = 32
strcmp(str1, str3) = 0
```

The first unmatched character between string `str1` and `str2` is third character. The ASCII value of 'c' is 99 and the ASCII value of 'C' is 67. Hence, when strings `str1` and `str2` are compared, the return value is 32.



# Passing Arrays to Functions

- An entire array can be passed to a function as an argument.
- Arrays passed-by-reference
  - Functions can modify original array data
  - Value of name of array is address of first element
    - Function knows where the array is stored
    - Can change original memory locations
- Individual array elements passed-by-value
  - Like regular variables
  - **`square ( myArray[3] ) ;`**



# Passing Arrays to Functions

- Specify name without brackets
  - To pass array **myArray** to **myFunction**

```
int myArray[ 24 ];
```

```
myFunction( myArray, 24 );
```

- Array size usually passed, but not required



# Passing Arrays to Functions

- Functions taking arrays
  - Function prototype
    - `void modifyArray( int b[], int arraySize );`
    - `void modifyArray( int [], int );`
      - Names optional in prototype
    - Both take an integer array and a single integer
  - No need for array size between brackets
    - Ignored by compiler
  - If declare array parameter as **const**
    - Cannot be modified (compiler error)
    - `void doNotModify( const int [] );`



```

#include <stdio.h>

void modify(int a[]); /* function prototype
* /

main ( )
{
    int count, a[3]; /* array definition */
    printf("\nFrom main, before calling the
function:\n");
    for (count = 0; count <= 2; ++count)
    {
        a[count] = count + 1;
        printf('a[%d] = %d\n', count, a[count]);
    }
    modify(a);
    printf("\nFrom main, after calling the
function:\n");
    for (count = 0; count <=2; ++count)
        printf("a[%d] = %d\n", count, a[count]);
}

```

```

void modify ( int a [ ] )
{
    int count;

    printf("\nFrom the function,
after modifying the
values:\n");
    for (count = 0; count <= 2;
        ++count) {
        a[count] = -9;
        p r i n t f ( " a [ % d ] = %d\n",
count, a[count]);
    }
    return;
}

```





From main, before calling the function:

a[0] = 1

a[1] = 2

a[2] = 3

From the function, after modifying the values:

a[0] = -9

a[1] = -9

a[2] = -9

From main, after calling the function:

a[0] = -9

a[1] = -9

a[2] = -9



# Sorting Arrays

- Sorting data
  - Important computing application
  - Virtually every organization must sort some data
    - Massive amounts must be sorted
- Bubble sort (sinking sort)
  - Several passes through the array
  - Successive pairs of elements are compared
    - If increasing order (or identical), no change
    - If decreasing order, elements exchanged
  - Repeat these steps for every element



i = 0	j	0	1	2	3	4	5	6	7
	0	5	3	1	9	8	2	4	7
	1	3	5	1	9	8	2	4	7
	2	3	1	5	9	8	2	4	7
	3	3	1	5	9	8	2	4	7
	4	3	1	5	8	9	2	4	7
	5	3	1	5	8	2	9	4	7
	6	3	1	5	8	2	4	9	7
i = 1	0	3	1	5	8	2	4	7	9
	1	1	3	5	8	2	4	7	
	2	1	3	5	8	2	4	7	
	3	1	3	5	8	2	4	7	
	4	1	3	5	2	8	4	7	
	5	1	3	5	2	4	8	7	
i = 2	0	1	3	5	2	4	7	8	
	1	1	3	5	2	4	7		
	2	1	3	5	2	4	7		
	3	1	3	2	5	4	7		
	4	1	3	2	4	5	7		
i = 3	0	1	3	2	4	5	7		
	1	1	3	2	4	5			
	2	1	2	3	4	5			
	3	1	2	3	4	5			
i = 4	0	1	2	3	4	5			
	1	1	2	3	4				
	2	1	2	3	4				
i = 5	0	1	2	3	4				
	1	1	2	3					
i = 6	0	1	2	3					
		1	2						

```
void bubbleSort(int arr[], int n)
{
    int i, j ,temp;
    for (i = 0; i < n-1; i++)
        // Last i elements are already in place
        for (j = 0; j < n-i-1; j++)
            if (arr[j] > arr[j+1])
            { temp= arr[j];
              arr[j]=arr[j+1];
              arr[j+1]= temp;
```



```
#define MAX 20  
main()  
{  
int i, n, min, num[MAX];  
printf("\nHow many elements do you wish to enter ?:");  
scanf("%d", &n);  
printf("Enter elements of array :\n");  
for(i=0; i < n; i++)  
scanf("%d", &num[i]);
```



```
bubbleSort(num, n);  
printf(“Sorted array :\n”);  
for(i=0; i < n; i++)  
printf(“\n %d”, num[i]); }
```



# Case Study: Computing Mean, Median and Mode Using Arrays

- Mean
  - Average (sum/number of elements)
- Median
  - Number in middle of sorted list
  - 1, 2, 3, 4, 5 (3 is median)
  - If even number of elements, take average of middle two
- Mode
  - Number that occurs most often
  - 1, 1, 1, 2, 3, 3, 4, 5 (1 is mode)



# Searching Arrays: Linear Search and Binary Search

- Search array for a key value
- Linear search
- Linear search
  - **Problem:** Given an array `arr[]` of  $n$  elements, write a function to search a given element  $x$  in `arr[]`.

A simple approach is to do **linear search**, i.e

- a. Start from the leftmost element of `arr[]` and one by one compare  $x$  with each element of `arr[]`
- b. If  $x$  matches with an element, return the index.
- c. If  $x$  doesn't match with any of elements, return -1.





```

int search(int arr[], int n, int x)
{
    int i;
    for (i=0; i<n; i++)
        if (arr[i] == x)
            return i;
    return -1;
}

main()
{
    int i, n, pos, num[MAX];
    printf("\nHow many elements do you
wish to enter ?:");
    scanf("%d", &n);

```

```

printf("Enter elements of array
:\n");
for(i=0; i < n; i++)
    scanf("%d", &num[i]);
printf("enter the element to
searched");
scanf("%d",&x);
pos=search(num, n, x);
if(pos==-1)
    printf("elements is not in list");
else
    printf("elements is at position:
%d",pos);
}

```



# Searching Arrays: Linear Search and Binary Search

- Binary search
  - Only used with sorted arrays
  - Compare middle element with key
    - If equal, match found
    - If key < middle
      - Repeat search on first half of array
    - If key > middle
      - Repeat search on last half
  - Very fast
    - At most N steps, where  $2^N > \# \text{ of elements}$
    - 30 element array takes at most 5 steps
$$2^5 > 30$$



If searching for 23 in the 10-element array:

2	5	8	12	16	23	38	56	72	91
---	---	---	----	----	----	----	----	----	----

23 > 16,  
take 2<sup>nd</sup> half

L									H
2	5	8	12	16	23	38	56	72	91

23 < 56,  
take 1<sup>st</sup> half

					L				H
2	5	8	12	16	23	38	56	72	91

Found 23,  
Return 5

					L	H			
2	5	8	12	16	23	38	56	72	91



```

main( )
{
int a[100],i,n,x, mid, top, bot, c;
printf("enter the array size;");
scanf("%d",&n);
printf("enter the array elements");
for(i=1;i<=n;i++)
scanf("%d",&a[i]);
top=1; bot=n; c=0;
printf("enter the element to searched");
scanf("%d",&x);
while((top <=bot)&&(c==0))
{

```

```

mid=(top+bot-1)/2;
if(a[mid]<x)
top=mid+1;
else
if(a[mid]>x)
bot=mid-1;
else
c=1;
}
if(c==1)
printf("elements is at
position;%d",mid);
else
printf("elements is not in list");
}

```



- Implement a recursive function for binary search



# Selection sort

- The selection sort algorithm sorts an array by repeatedly finding the minimum element (considering ascending order) from unsorted part and putting it at the beginning. The algorithm maintains two subarrays in a given array.
  - 1) The subarray which is already sorted.
  - 2) Remaining subarray which is unsorted.
- In every iteration of selection sort, the minimum element (considering ascending order) from the unsorted subarray is picked and moved to the sorted subarray



# Selection sort- example

```
arr[] = 64 25 12 22 11
```

```
// Find the minimum element in arr[0...4]
```

```
// and place it at beginning
```

```
11 25 12 22 64
```

```
// Find the minimum element in arr[1...4]
```

```
// and place it at beginning of arr[1...4]
```

```
11 12 25 22 64
```

```
// Find the minimum element in arr[2...4]
```

```
// and place it at beginning of arr[2...4]
```

```
11 12 22 25 64
```

```
// Find the minimum element in arr[3...4]
```

```
// and place it at beginning of arr[3...4]
```

```
11 12 22 25 64
```

```

void selectionSort(int arr[], int n)
{
    int i, j, min_idx;
    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[j] < arr[min])
            {
                min = j;
            }
        }
        // Swap the found minimum element with the first element
        temp = arr[min];
        arr[min] = arr[i];
        arr[i] = temp;
    }
}

```

