UNIT-3

Topics to be covered in UNIT-3

Arrays:

- Definition and declaration
- Initialization
- Accessing elements of arrays
- Storing values in arrays,
- 1-D Arrays
- 2-D Arrays
- Character arrays
- Multidimensional arrays.

Function and arrays:

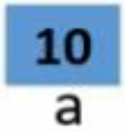
- passing individual elements to arrays
- passing 1-D array
- 2-D array to function

Applications:

- Linear search
- matrix addition
- subtraction
- multiplication
- transpose

Why we need Array Data Structure?

```
int a = 10;
```



- But we can **store only one value** in variable 'a'

What if we want to store 100 values???

Creating **100 different variables** to store **100 values** is **not good idea**

Solution is to use Array Data Structure

For Example:

Student '**S**'

Subject **A**	Subject **B**	Subject **C**	Subject **D**	Subject **E**
46	41	39	38	43

```
int a = 46;
int b = 41;
int c = 39;
int d = 38;
int e = 43;
```

```
int s[5] = {46, 41, 39, 38, 43};
```

46	41	39	38	43
0	1	2	3	4

Arrays in C:

- An **array** is a numbered collection of variables of the same type.
- An array is a **homogeneous** collection of data.
- An array represents a group of **related** data.
- The variables in an array **share** the same name and are distinguished from one another by their numbers or subscripts.
- We can **loop** through the variables of an array by using a variable for the subscript.
- The subscripts are always **0,1,…, size-1.**
- In memory all the data items are stored in **contiguous memory locations** one after the other

Array Characteristics:

An array has two distinct characteristics:

- An array is **ordered:** Data is grouped sequentially.
- An array is **homogenous:** Every value within the array must share the same data type.
- In other words, an int array can only hold ints, not doubles.

How to declare and define an array:

Creation of an consists two things: **Element Type** and **Array Size**.

Element Type: What kind of data an array can hold?
An array can hold any one of the following data:
integer, double, character data.

Array Size: How many elements an array can contain?
Once an array size is defined it cannot be changed at run-time.

```
type arrayname[size];
```

Using arrays in C:

In C, arrays can be classified based on how the data items are arranged for human understanding. Arrays are broadly classified into three categories,

1. **One** Dimensional Arrays
2. **Two** Dimensional Arrays
3. **Multi** Dimensional Arrays

One Dimensional Arrays:

- One dimensional array is a linear list consisting of related and similar data items.
- In memory all the data items are stored in contiguous memory locations one after the other.

Syntax for declaring One Dimensional Arrays:

elementType arrayName[size];

- Where **elementType** specifies data type of each element in the array, **arrayName** specifies name of the variable you are declaring and **size** specifies number of elements allocated for this array.

- To declare regular variables we just specify a data type and a unique name. **Example:** int number;

- To declare an array, we just add an array size.

  Example: **int temp[5];** //Creates an array of 5 integer elements.

  Example: **double stockprice[31];**

  //Creates an array of 31 double elements.

Initializing One Dimensional Arrays

- If array is not initialized it contain **garbage** values.

- **Types of array initializations:**

Option 1: Initializing **all** memory locations

int temp [5] = {75, 79, 82, 70, 68};

Option 2: Initialization **without** size

int temp [] = {75, 79, 82, 70, 68};

Option 3: **Partial** array initialization

int temp [5] = {75, 79, 82};

Option 4: Initializing an entire array with **zero**.

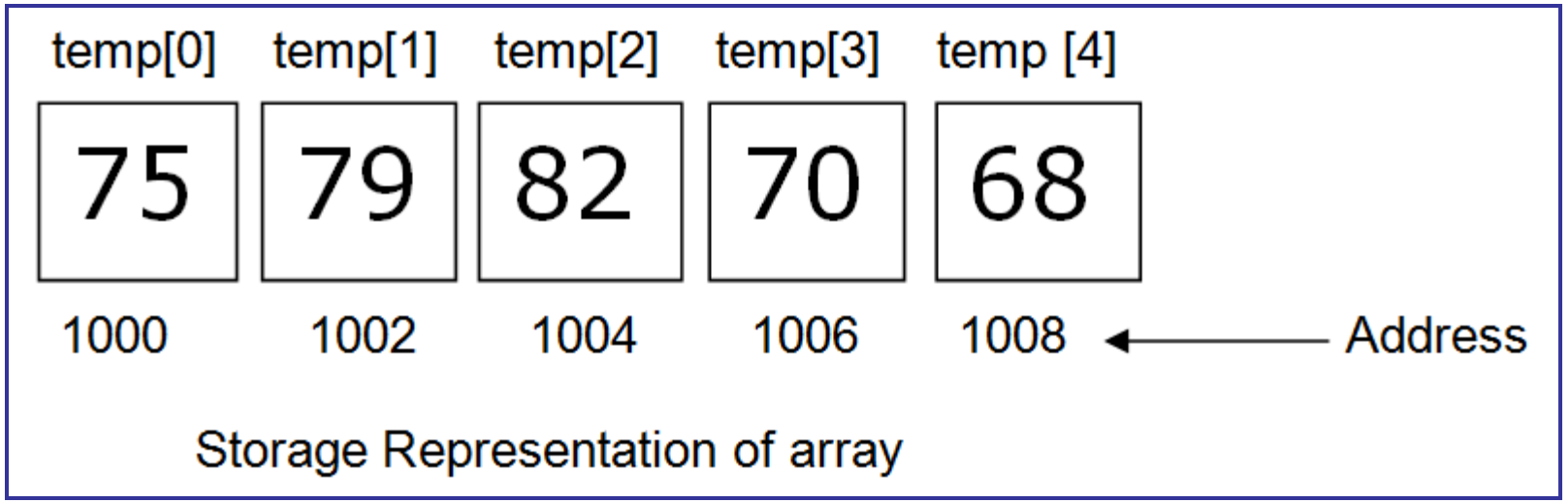
int temp [5] = {0};

Option 1: Initializing all memory locations:

- If you know all the data at compile time, you can specify all your data within brackets:

int temp [5] = {75, 79, 82, 70, 68};

- During compilation, 5 contiguous memory locations are reserved by the compiler for the variable temp and all these locations are initialized as shown below.



- If the size of integer is 2 bytes, 10 bytes will be allocated for the variable temp.

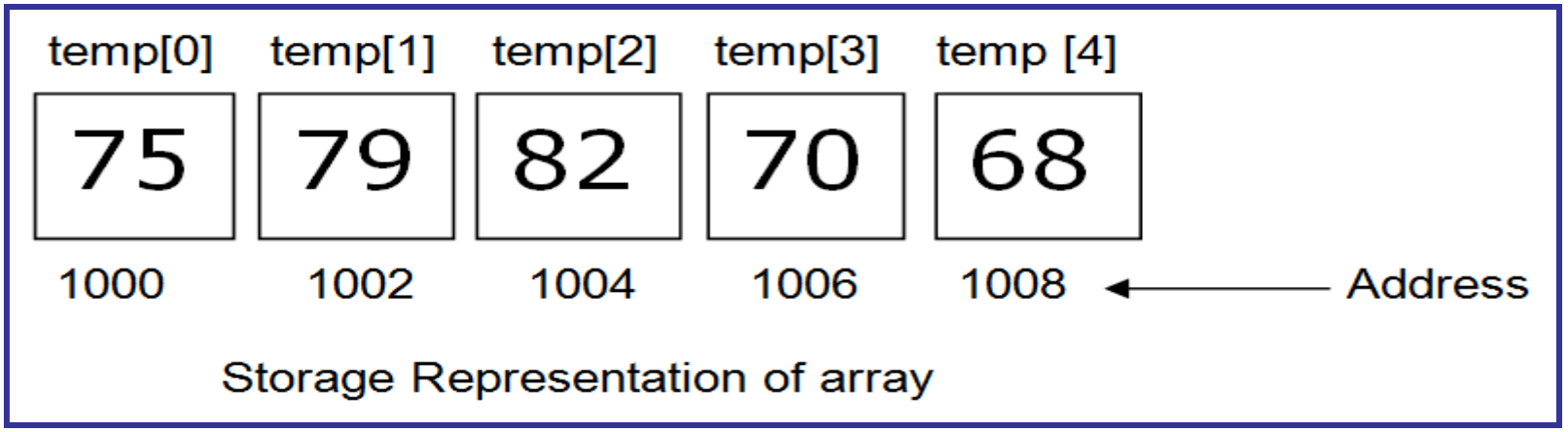
Option 2: Initialization without size:

- If you omit the size of an array, but specify an initial set of data, then the compiler will automatically determine the size of an array.

int temp [] = {75, 79, 82, 70, 68};

- In the above declaration, even though you have not specified exact number of elements to be used in array temp, the array size will be set with the total number of initial values specified.

- Here, the compiler creates an array of 5 elements. The array temp is initialized as shown below.



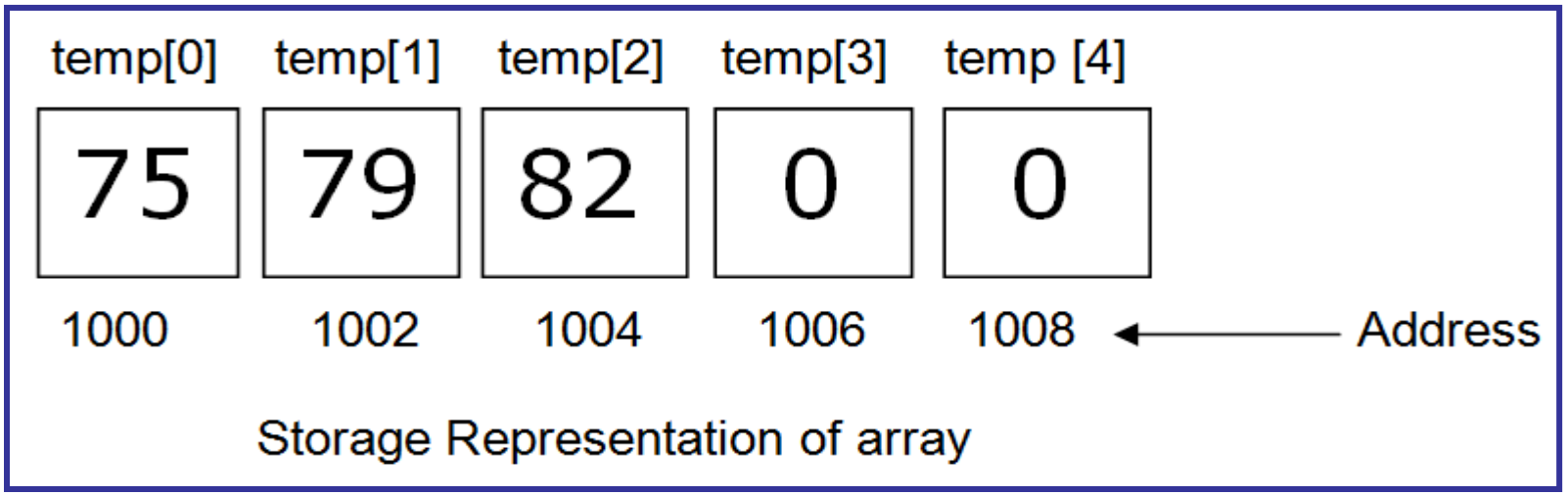
Option 3 Partial Array Initialization:

- If the number of values to be initialized is **less than the size of the array**, then the elements are initialized in the order from 0^{th} location.

- The remaining locations will be initialized to zero automatically.

int temp [5] = {75, 79, 82};

- Even though compiler allocates 5 memory locations, using the above declaration statement, the compiler initializes first three locations with 75, 70 and 82, and the next set of memory locations are automatically initialized to 0's by the compiler as shown below.

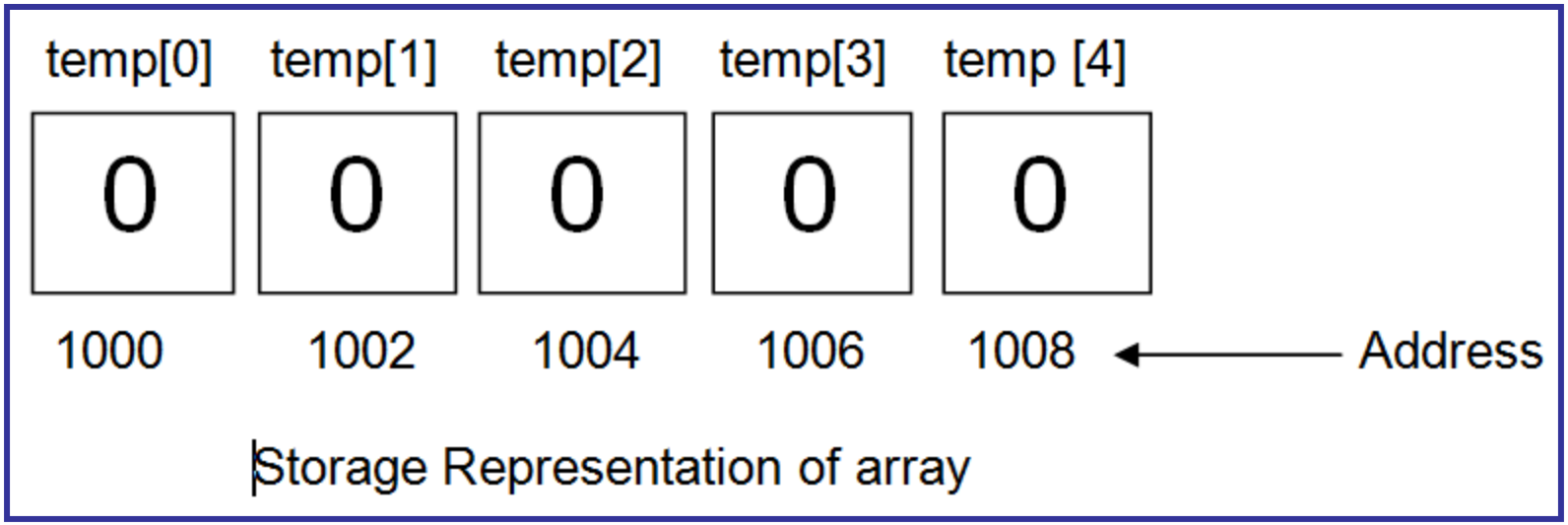


Option 4: Initializing an entire array with zero:

- If you do not know any data ahead of time, but you want to initialize everything to 0, just use 0 within { }. For example:

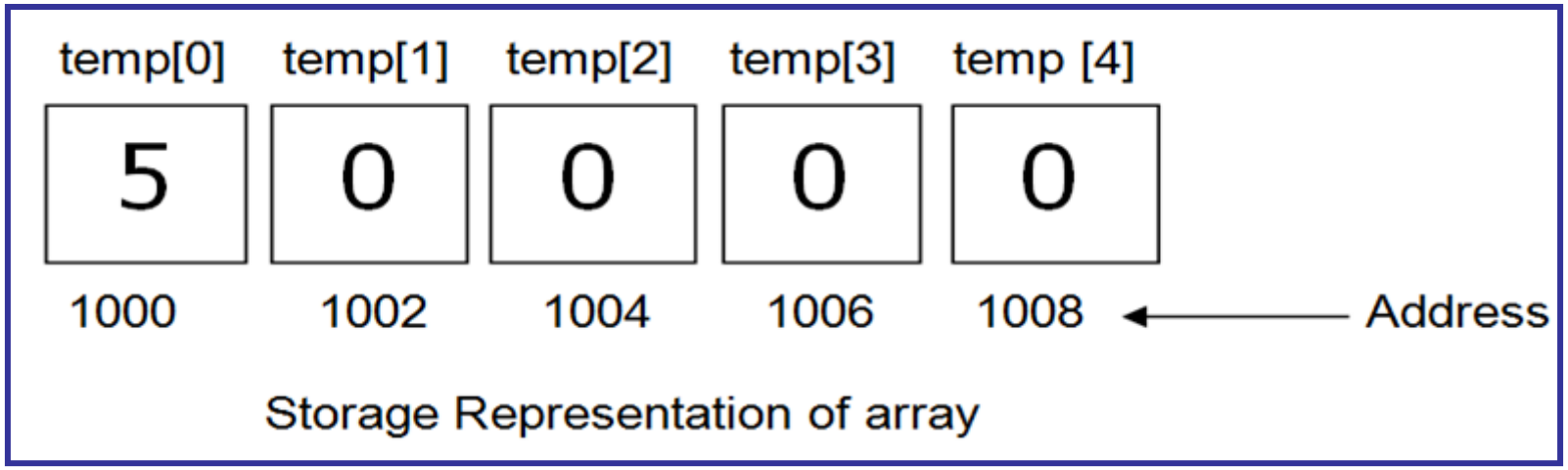
  int temp [5] = {0};

- This will initialize every element within the array to 0 as shown below.



Example:

int temp [5] = {5};



- The first value is supplied in the first element memory location, remaining all elements are placed with zero.

Accessing elements of one dimensional array:

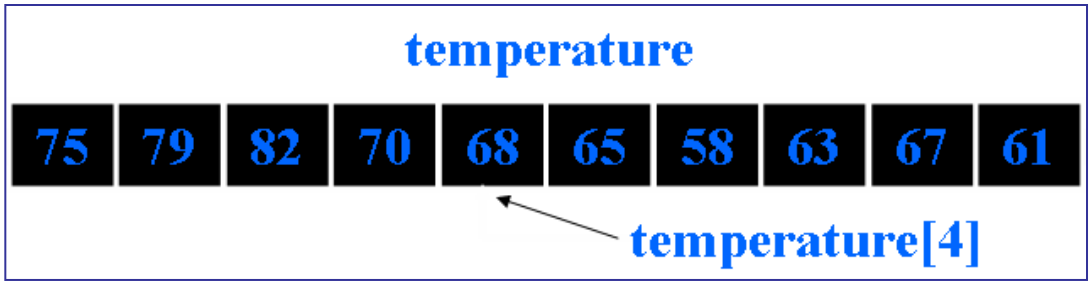
- You know how to declare and initialize an array. Now lets understand, how to access an array elements.

- To access an array element use **name** of the array with the **subscript** in **brackets**.

- Suppose you have an array called **temperature**, for storing temperature in a year.
- Then the subscripts would be 0,1,…,364.

- **For example to access temperature of fifth day:**

temperature [4]



Accessing elements of one dimensional array(Cont…)

- To **assign or store** the value 89 for the 150th day:

```
temperature [149] = 89;
```

- You can loop through the elements of an array by varying the subscript.
- To set all of the values to 0, say

```
for(i = 0;i < 365; i++)
    temperature[i] = 0;
```

- You can not use assignment statement directly with arrays.
- If a[] , b[] are two arrays then the assignment a=b is not valid.

- **C does not provide array bounds checking.**
- Hence, if you have

```
double stockPrice[5];
printf ("%d", stockPrice[10]);
```

- This will compile, but you have overstepped the bounds of your array.
- You may therefore get wrong value.

//Example for accessing array elements.

```
#include<stdio.h>
main() {
        int arr[5]={11,5,8,3,6};
        int i = 0;
        for(i=0;i<5;i++)
        {
                printf("%d",arr[i]);
        }
        int j = arr[4];
        printf("Value at 5th location is: %d",j);
}
```

- As we can see above, the 5th element of an array is accessed as '**arr[5]**'.

- **Note** that for an array declared as int arr[5].
- The five values are represented as: **arr[0] arr[1] arr[2] arr[3] arr[4]** and **not** **arr[1] arr[2] arr[3] arr[4] arr[5]**

- The first element of array always has a **subscript of '0'**.

//Example for array elements.

```
#include<stdio.h>
void main()
{
      int arr[6];
      int i = 0;
      printf(“enter values”);
      for(i=0;i<6;i++)
     {
       scanf(“%d”,&arr[i]);
       }
     printf(“array elements are:);
     for(i=0;i<6;i++)
     {
       printf(“%d”,arr[i]);
       }
}
```

Example for array elements

```
#include<stdio.h>
void main()
{
      int x[6], y[6];
      int i = 0;
      printf(“enter values”);
      for(i=0;i<6;i++)
     {
       scanf(“%d”,&x[i]);
       }
       for(i=0;i<6;i++)
       {   y[i]=x[i];}
     printf(“array elements are:);
     for(i=0;i<6;i++)
     {
       printf(“%d”,y[i]);
       }
```

```
#include<stdio.h>
#define  size 60
void main()
{
      int arr[size];
      int i = 0;
      printf(“enter values”);
      for(i=0;i<size;i++)
     {
       scanf(“%d”,&arr[i]);
       }
     printf(“array elements are:);
     for(i=0;i<size;i++)
     {
       printf(“%d”,arr[i]);
       }
}
```

//Program to calculate sum of all the array elements.

```
#include <stdio.h>
 void main()
{
        int a[10];
        int i, n, total=0;
        printf("\n Enter the size of the array : ");
        scanf("%d", &n);
        printf("\n Enter the elements of an array : ");
        for (i = 0; i < n ; i++)
                        scanf("%d",&a[i]);
        for (i = 0; i < size ; i++)
                total += a[i];
        printf("Sum of all array elements: %d", total);
}
```

Two Dimensional Arrays:

- Two-dimensional array are those type of array, which has finite number of rows and finite number of columns.

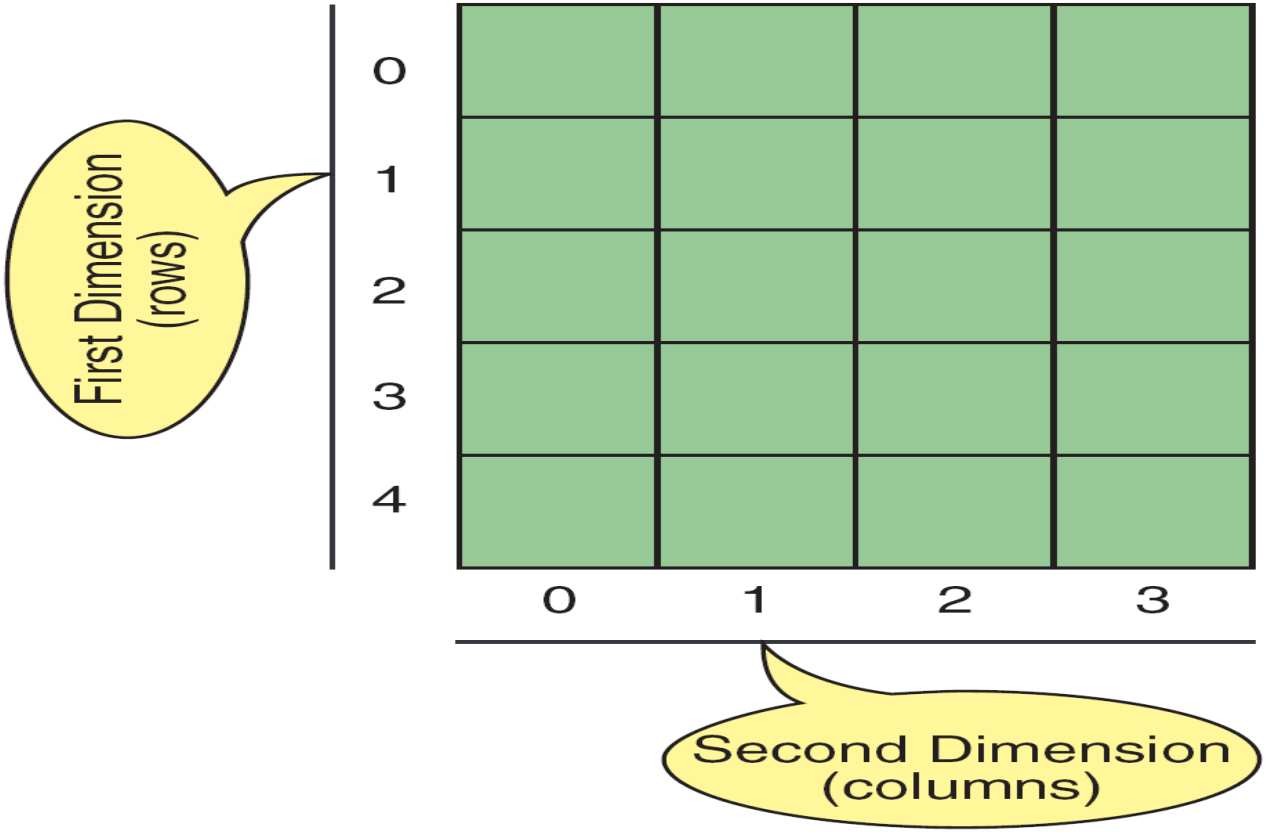
- An array of array is called a two-dimensional array and can be represented as a table with rows and columns.

is a 3 × 4 array
3 rows, 4 columns

- This is an array of size 3 names whose elements are arrays of size 4.

Declaration of Two Dimensional Arrays:

Syntax: **elementType arrayName [row_size][column_size];**



Two-dimensional Array representation

- The declaration form of 2-dimensional array is

  elementType arrayName [row_size][column_size];

- The elementType may be any valid type supported by C.
- The rule for giving the arrayName is same as the ordinary variable.
- The row_size and column_size should be an individual constant.

- The following declares a two-dimensional 3 by 3 array of integers and sets the first and last elements to be 10.

  int matrix [3][3];
  matrix[0][0] = 10;
  matrix[2][2] = 10;

	[0]	[1]	[2]
[0]	10		
[1]			
[2]			10

- In the declaration of two dimensional array the **column size** should be specified, so that it can arrange the elements in the form of rows and columns.
- Two-dimensional arrays in C are stored in "**row-major format**": the array is laid out contiguously, one row at a time.

Initialization of Two Dimensional Arrays:

- An array may be initialized at the time of declaration as follows:

```
char names [3][4]   =    {
                             {‘J’, 'o', 'h', 'n'},
                             {‘M’, 'a', 'r', 'y'},
                             {‘I’, 'v', 'a', 'n'}
                         };
```

- An integer array may be initialized to all zeros as follows

```
int nums [3][4] = {0};
```

- An integer array may be initialized to different values as follows

```
int nums [3][4] = {
                      {1,2,3,4},
                      {5,6,7,8},
                      {9,0,10,11}
                  };
```

- To access an element of a 2D array, you need to specify both the row and the column:

```
printf ("%d", nums[1][2]);
```

Example:

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

                     OR

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

                     OR

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

```
//Program to print sum of elements of a matrix.
#define M 3          /* Number of rows */
#define N 4          /* Number of columns */
main() {
,          int  a [ M ] [ N ], i, j, sum = 0;
           for ( i = 0; i < M; ++i ) {
                      for (j = 0; j < N, ++j ){
                                 scanf (%d", &a [i] [j]);
                      }
           }
           for ( i = 0; i < M; ++i ) {
                      for (j = 0; j < N, ++j ) {
                                 printf("a [ %d ] [ %d ] = %d     ", i, j, a[ i ] [ j ]);
                      }
                      printf ("\n");
           }
           for ( i = 0; i < M; ++i ) {
                      for (j = 0; j < N, ++j ) {
                                 sum += a[ i ] [ j ];
                      }
                      printf("\nsum = %d\n\n");
           }
 }
```

output:

```
a[0][0]=0  a[0][1]=1  a[0][2]=2  a[0][3]=3
a[1][0]=1  a[1][1]=2  a[1][2]=3  a[1][3]=4
a[2][0]=0  a[2][1]=3  a[2][2]=4  a[2][3]=5

sum=30
```

Multi Dimensional Arrays:

- C allows three or more dimensions. The exact limit is determined by the compile.

- The general form of multidimensional array is

  elementType arrayName [s1][s2][s3]…[sm];

- Where si is the size of the i^{th} dimension.

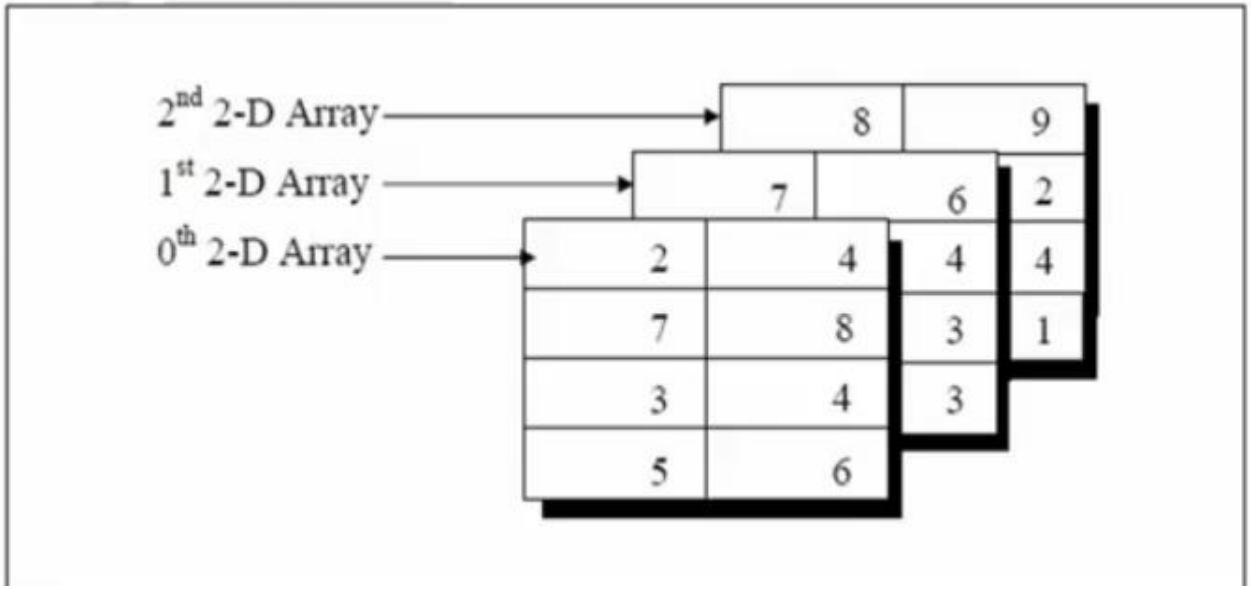
- Array declarations read right-to-left
- For Example: **int a[3][5][4];**

- It is represented as “an array of ten arrays of three arrays of two ints”
- In memory the elements are stored as shown in below figure.

Syntax for creating 3-D array

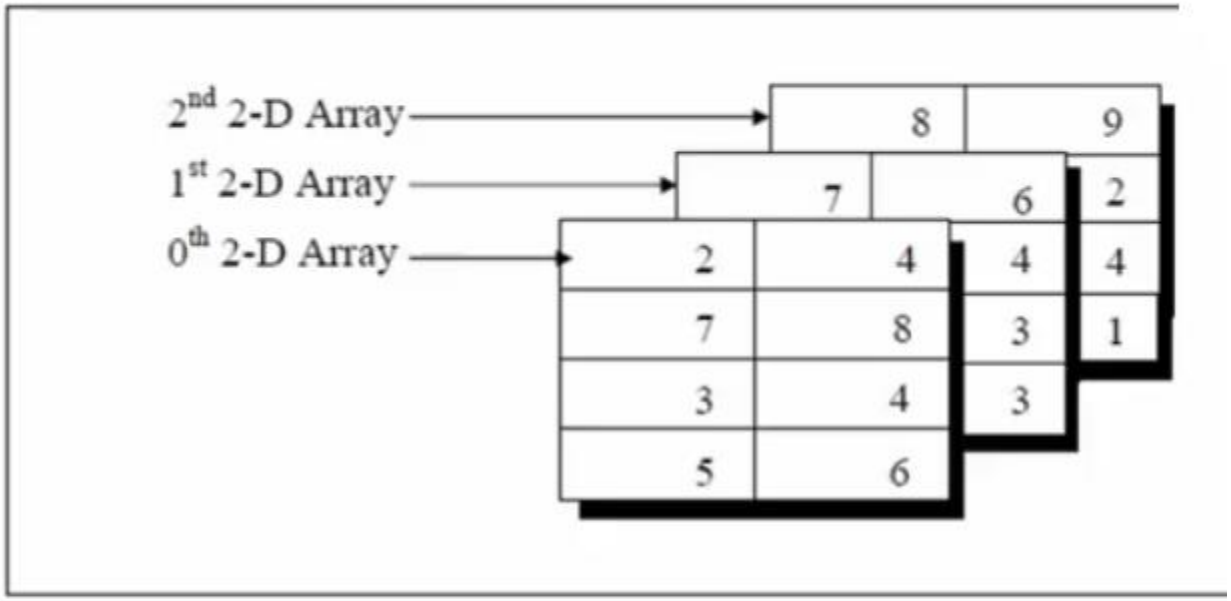
data_type array_name [No. of 2-d arrays][rows][columns];

No. of elements = No. of 2D Arrays X No. of Rows x No. of Columns



eg.

```
int s[3][4][2] = {
   {
      { 2, 4 },
      { 7, 8 },
      { 3, 4 },
      { 5, 6 }
   },
   {
      { 7, 6 },
      { 3, 4 },
      { 5, 3 },
      { 2, 3 }
   },
   {
      { 8, 9 },
      { 7, 2 },
      { 3, 4 },
      { 5, 1 },
   }
  };
```



printf("%d", s[1][0][1]); → 6

printf("%d", s[0][2][0]); → 3

Example:

```
int table[3][5][4] =  {
                                    {
                                            {000,001,002,003},
                                            {010,011,012,013},
                                            {020,021,022,023},
                                            {030,031,032,033},
                                            {040,041,032,043}
                                    },
                                    {
                                            {100,101,102,103},
                                            {110,111,112,113},
                                            {120,121,122,123},
                                            {130,131,132,133},
                                            {140,141,142,143}
                                    },
                                    {
                                            {200,201,202,203},
                                            {210,211,212,213},
                                            {220,221,222,223},
                                            {230,231,232,233},
                                            {240,241,242,243}
                                    }
                            };
```

Character Arrays and Strings:

- String is a sequence of characters.

- If ‘\0’ is present after a series of characters in an array, then that array becomes a string otherwise it is a character array.

Example:

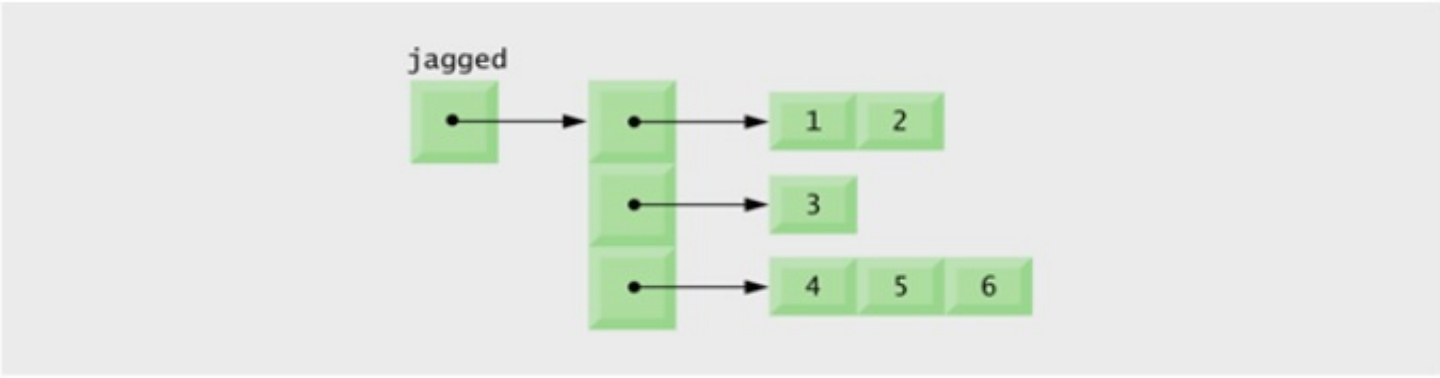
```
char arr[] = {'a', 'b', 'c'};          //This is an array

char arr[] = {'a', 'b', 'c', '\0'};    //This is a string
```

Ragged array

- **Ragged array**, also known as a **jagged array**
- Another type of multidimensional array, Jagged Array.
- It is an array of arrays in which the length of each array can differ.
- Example where this array can be used is to create a table in which the number of columns differ in each row.
- Say, if row1 has 2 columns, row2 has 1 columns then row3 can have 3 columns and so on.

Ragged array declaration and initialization

```
int jagged [3][10] = { {1,2},{3},(4,5,6}};
```



```
#include<stdio.h>
void main()
{
int i,j;
int a[10][10];
a[0][0]=1;
a[1][0]=5;
a[1][1]=9;
a[2][0]=14;
a[2][1]=20;
a[2][2]=35;
printf("a[0][0]= %d\n",a[0][0]);
printf("a[1][0]= %d\t",a[1][0]);
printf("a[1][1]= %d\n",a[1][1]);
printf("a[2][0]= %d\t",a[2][0]);
printf("a[2][1]= %d\t",a[2][1]);
printf("a[2][2]= %d",a[2][2]);
}
```

```
a[0][0]= 1
a[1][0]= 5      a[1][1]= 9
a[2][0]= 14     a[2][1]= 20     a[2][2]= 35
```

```
#include <stdio.h>
void main()
{
 int ra[3[5] = {
{10,20},{11,15,30,14},{12,17,19}};
 int i,j;
 printf(“array elements are”);
 for(i=0;i<3;i++)
 {
for(j=0;j<5&& ra[i][j]!=0;j++)
  {
 printf("%d\t", ra[i][j]);
}
 printf("\n");
 }
 }
```

```
10      20
11      15      30      14
12      17      19
```

Dynamic Array: **The dynamic arrays are the arrays which are allocated memory at the runtime and the memory is allocated using dynamic memory allocation functions(malloc ,calloc, realloc).**

Read and print array elements using Malloc

```
#include <stdio.h>
#include <stdlib.h>
int main()
{ int n, i, *ptr, sum = 0;
printf("Enter number of elements: ");
scanf("%d", &n);
ptr = (int*) malloc(n * sizeof(int));
if(ptr == NULL)
{
printf("Error! memory not allocated.");
 exit(0);
}
printf("Enter elements of array: ");
for(i = 0; i < n; ++i)
{              scanf("%d", ptr + i);              }
printf("array elements are");
for(i=0;i<n;i++)
printf(" %4d",*(ptr+i));
free(ptr);
return 0;                  }
```

Read and print array elements using Calloc

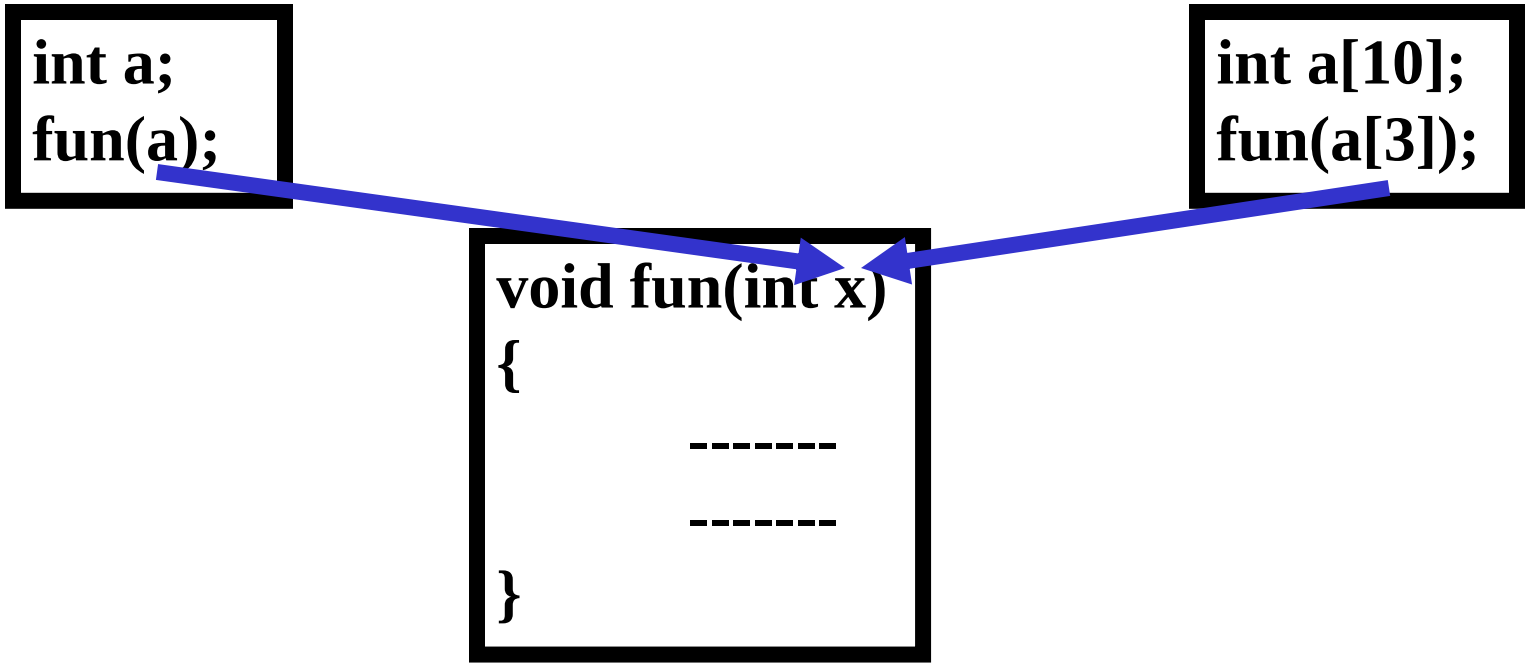
```
#include <stdio.h>
#include <stdlib.h>
int main()
{ int n, i, *ptr, sum = 0;
printf("Enter number of elements: ");
scanf("%d", &n);
ptr = (int*) calloc(n, sizeof(int));
if(ptr == NULL)
{
printf("Error! memory not allocated.");
 exit(0);
}
printf("Enter elements of array: ");
for(i = 0; i < n; ++i)
{              scanf("%d", ptr + i);              }
printf("array elements are")
for(i=0;i<n;i++)
printf(" %4d",*(ptr+i));
free(ptr);
return 0;                  }
```

Inter-function communication (Functions with Arrays):

- Like the values of variable, it is also possible to pass values of an array to a function.

- There are two types of passing an array to the function:
  - 1. Passing Individual Elements
  - 2. Passing the whole array

1. Passing Individual Elements:



```
//program for passing individual elements
#include<stdio.h>
void  print(int);
void main()
{
   int a[10], n ,i;
   printf("\n Enter the size of the array ");
   scanf("%d",&n);
   printf('\n Enter the elements of the array  :  ");
   for(i=0;i<n;i++)
          scanf("%d",&a[i]);
   for(i=0;i<n;i++)
          print(a[i]);   //fn call
 }
void  print(int x)
{
printf("%d",x);
}
```

2. Passing the whole array:

- To pass an array to a called function, it is sufficient to list the **name** of the array, **without** any subscripts, and the **size** of the array as arguments.

- For example, the function call **findMax(a, n);** will pass all the elements contained in the array a of size n.

- The called function expecting this must be appropriately defined.
- The findMax function header looks like: **int findMax(int x[], int size)**

- The pair of brackets informs the compiler that the argument x is an array of numbers. It is not necessary to specify the size of the array here.

- The function prototype takes of the form

  int findMax (int [], int);
  int findMax (int a [], int);

//Program to read an array of elements and find max value.

```
#include<stdio.h>
int findMax(int[],int);
void main()
{
   int a[10], n ,i , max;
   printf("\n Enter the size of the array ");
   scanf("%d",&n);
   printf('\n Enter the elements of the array  :  ");
   for(i=0;i<n;i++)
         scanf("%d",&a[i]);
   max=findMax(a, n);
   printf("\n The Maximum value =%d", max);
 }
```

```
int findMax(int x[],int size)
{
        int m,i;
        m=x[0];
        for(i=1;i<size; i++)
        {
            if(x[i]>m)
            {
                m=x[i];
            }
        }
        return m;
}
```

```
Enter the size of the array 5

Enter the elements of the array  :  10 4 56 44 12

The Maximum value =56
```

Passing two dimensional array to function:

The same concept is applied for passing a **multidimensional** array to a function.

- Here you need to specify the **column** size.
- For example to read two-dimensional array elements using functions, it takes of the form:

int readElements (int [] [5], int ,int);

```
#include<stdio.h>
int findmin(int a[][5],int,int);
void main()
{
   int a[5][5], r,c ,i , m;
   printf("\n Enter the size of the array ");
   scanf("%d%d",&r,&c);
   printf('\n Enter the elements of the array  :  ");
   for(i=0;i<r;i++)
   {
            for(j=0j<c;j++)
            scanf("%d",&a[i]);        }
   m=findmin(a, r,c);
   printf("\n The Maximum value =%d", max);
 }
```

```
int findmin(int a[][5],int r, int c)
{
      int min,i,j;
      min=a[0][0];
      for(i=0;i<r; i++)
      {
            for(j=0;j<c;j++)
            {
                  if(a[i][j]<min)
                     min=a[i][j];
              }
       }
            return min;
}
```

Applications of Array:

- Linear search
- Matrix addition
- Subtraction
- Multiplication
- Transpose

Array Applications:

- Array used to store the data or values of same data type
  Ex: int x[30]
- Array Used for Maintaining multiple variable names using single name
  we can read multiple values using single variable name, instead of using multiple varibles
- Array Can be Used for Sorting Elements and searching element
- Array Can Perform Matrix Operation
- Array Can be Used in CPU Scheduling
- Array Can be Used in Recursive Function

Advantages of array

Code Optimization: Less code is required, one variable can store numbers of value.
Easy to traverse data: By using array easily retrieve the data of array.
Easy to sort data: Easily sort the data using swapping technique
Random Access: With the help of array index you can randomly access any elements from array.

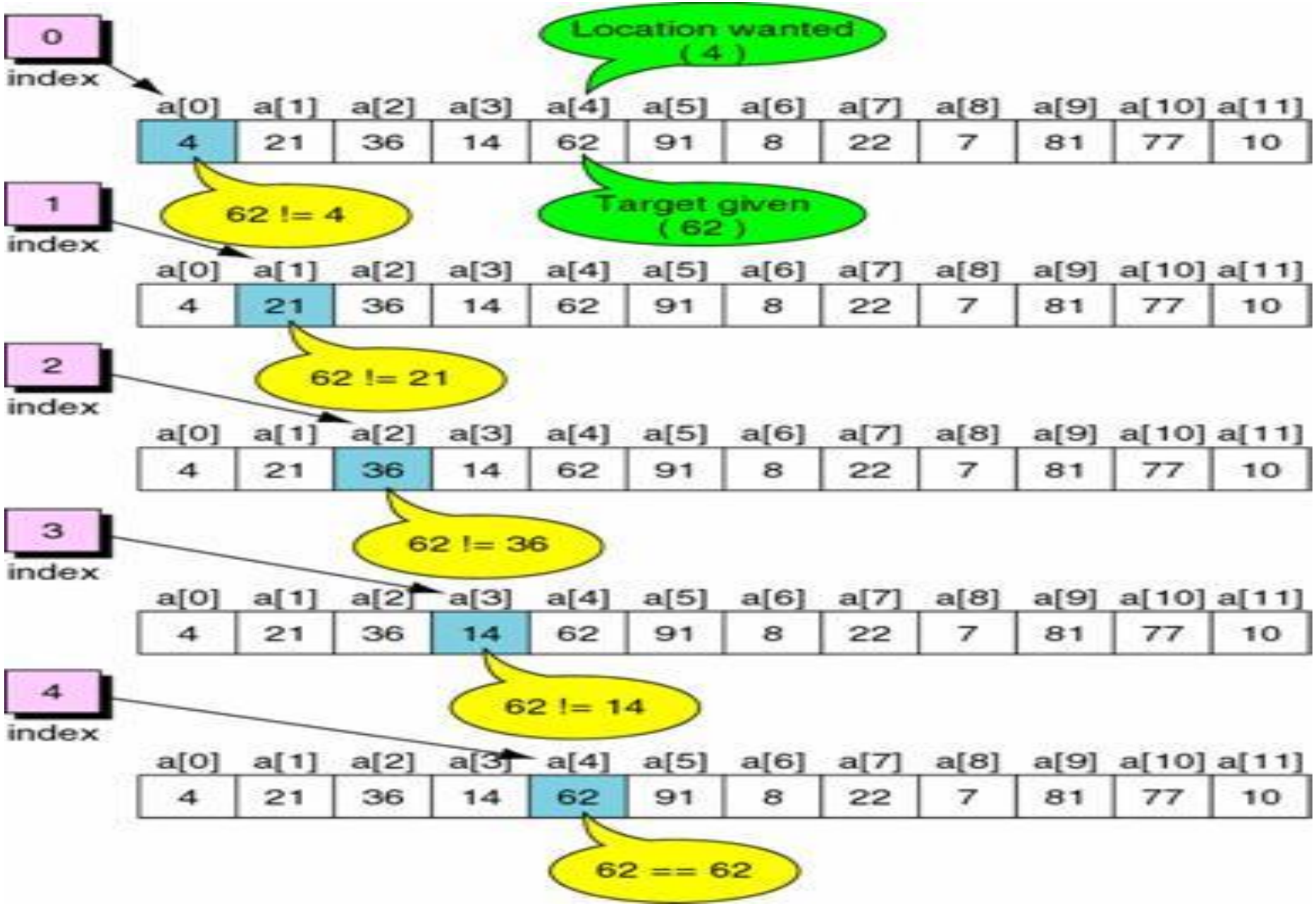
- ***Searching*** *is the process of finding a particular element in an array.*

Linear Search:

*A **linear or sequential search** of an array begins at the beginning of the array and continues until the item is found or the entire array has been searched.*

- *It is the basic searching technique.*
- *Very easy to implement*
- *The array DOESN'T have to be sorted.*
- *All array elements must be visited if the search fails.*
- *Could be very slow.*

Example: Successful Linear Search



```
//Program to find a number in the array using sequential search.
main() {
        int a[10],i,n,m,f=0;
        printf("Enter the size of an array: ");
        scanf("%d",&n);
        printf("Enter the elements of the array: ");
        for(i=0;i<=n-1;i++){
                scanf("%d",&a[i]);
        }
        printf("Enter the number to be search: ");
        scanf("%d",&m);
        for(i=0;i<=n-1;i++) {
                if(a[i]==m) {
                        f=1;
                        break;
                }
        }
        if(f==1)
                printf("The number is found");
        else
                printf("The number is  not in list");
}
```

Output:

```
Enter the size of an array: 5
Enter the elements of the array: 4 6 8 0 3
Enter the number to be search: 0
The number is found
```

Linear Search Program using functions

```
#include<stdio.h>
 int lsearch(int [], int, int);
 void main()
{
   int a[10], key, i, n, pos;
    printf("Input number of elements in array\n");
   scanf(“%d", &n);
    printf("Input %d numbers\n", n);
    for (i = 0; i < n; i++)
      scanf("%ld", &a[i]);
    printf("Input number to search\n");
   scanf("%d",&key);
    pos = lsearch(a, n, key);
    if (pos == -1)
      printf("%d is not present in array.\n", key);
   else
 printf("%d is present at location %d.\n", key,
pos+1);
   }
```

```
int lsearch(int a[], int n, int
find)
{
   int i;
    for (i = 0 ;i < n ; i++ )
   {
      if (a[i] == find)
         return i;
   }
    return  -1
                                    }
```

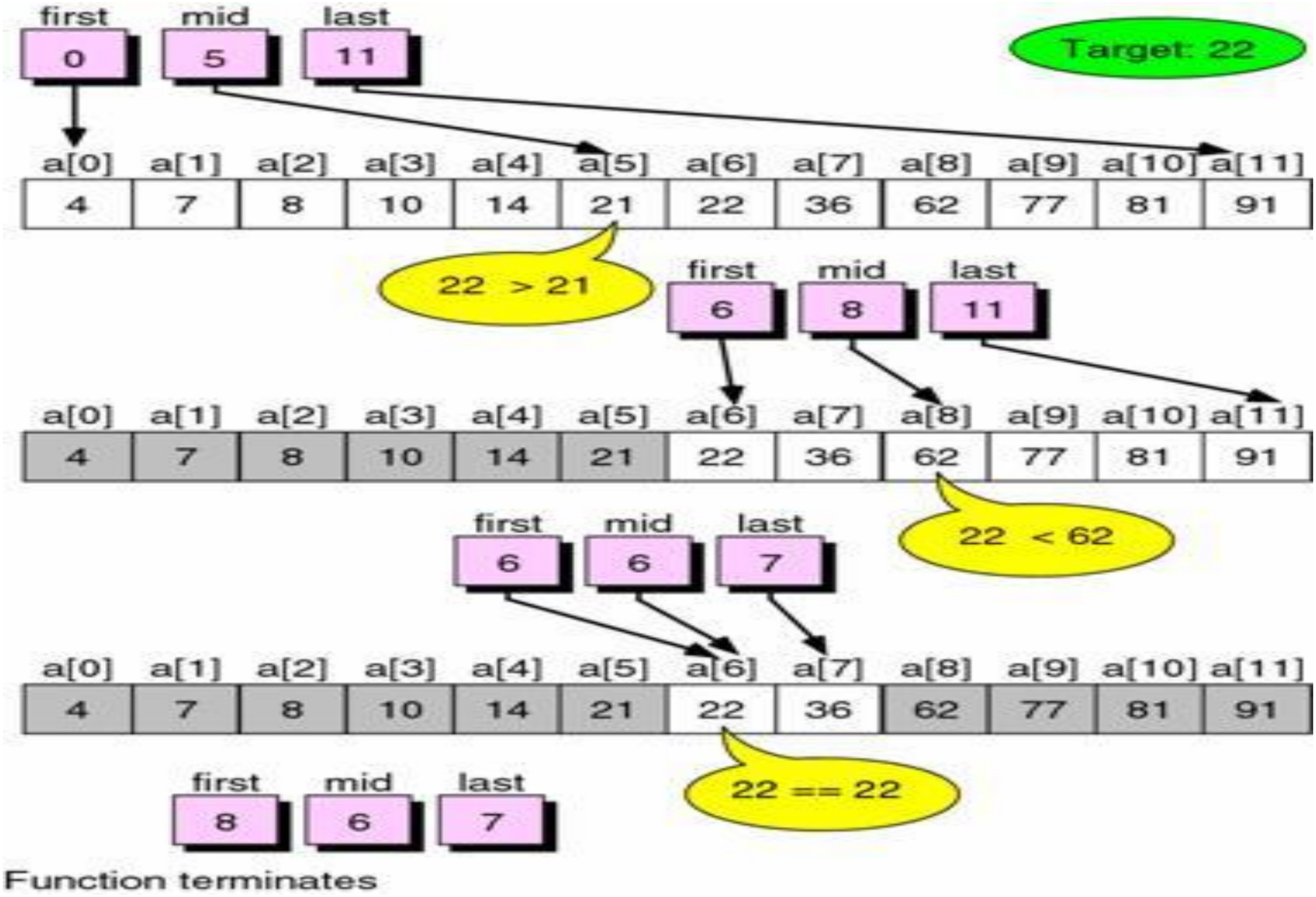
Binary Search: *The search starts at the center of a sorted array, if center is equal to target element search is successful otherwise it determines which half to continue to search on that basis.*

- *The algorithm starts searching with the **mid** element.*

  mid=(first + last)/2

  - *If the item is equal to mid then search is successful.*
  - *If the item is **less than** the mid element, it starts over searching the first half of the list.*
  - *If the item is **greater than** the mid element, it starts over searching the second half of the list.*
  - *It then continues halving the list until the item is found.*
- *Each iteration eliminates half of the remaining elements.*
- *It is **faster** than the linear search.*
- *It works only on SORTED array.*
- *Thus, there is a performance penalty for sorting the array.*

Example: Successful Binary Search



Non recursive program

```
#include <stdio.h>
int bsearch (int a[], int low, int high,  int key);
void main() {
            int a[15], key, pos,i,n;
printf(“Enter size of array”);
scanf(“%d”,&n);
printf("Enter %d elements in ascending or descending
order: \n “,n);
            for(i=0;i<n;i++)
                        scanf("%d",&a[i]);
            printf("Enter an element that is to be
searched: ");
            scanf("%d", &key);
            pos = bsearch(a, 0, n-1, key);
            if (pos = -1)
      printf("Sorry, target item was not found");
     else
    printf("\nTarget was found at index: %d ", pos);
```

```
Enter 8 elements in ascending or descending order:
1
2
3
4
5
6
7
8
Enter an element that is to be searched: 8

Target was found at index: 7
```

```
int bsearch (int a[], int low, int high, int key)
{
        int  mid;
        while (low <=high)
       {
            mid = ( low + high ) / 2;
            if ( key ==a[mid] )
            {
                return mid;
            }
            else if ( key < a[mid] )
                        high = mid - 1;
            else
                        low = mid + 1;
            }

            return -1;
}
```

- ***Big O Notation***
  - *Indicates, how hard an algorithm has to work to solve a problem.*

Time Complexities of Searching & Sorting Algorithms:

	Best Case	***Average Case***	***Worst Case***
Linear Search	***O(1)***	***O(n)***	***O(n)***
Binary Search	***O(1)***	***O(log n)***	***O(log n)***

//C program to print addition of two matrices.

```
 void main()
{
int a[5][5],b[5][5],c[5][5];
int i,j,k,r1,r2,c1,c2;
printf("Enter the dimensions of matrix A");
scanf("%d%d",&r1,&c1);
printf("Enter the dimensions of matrix B");
scanf("%d%d",&r2,&c2);
if(r1==r2&&c1==c2)
{
printf("Enter the elements of matrix A");
for(i=0;i<r1;i++){
      for(j=0;j<c1;j++){
            scanf("%d",&a[i][j]);
      }
}
printf("Enter the elements of matrix B");
for(i=0;i<r2;i++){
      for(j=0;j<c2;j++)            {
            scanf("%d",&b[i][j]);
      }
}
for(i=0;i<r1;i++) {
      for(j=0;j<c1;j++){
            c[i][j]=a[i][j]+b[i][j];
            }
      }

 printf("The resultant matrix C\n");
 for(i=0;i<r1;i++) {
      for(j=0;j<c2;j++) {
                        printf("%3d",c[i][j]);
            }
      printf("\n");
}
} //if close
else
{
printf("addition is not possible");
}
}//main
```

//C program to print multiplication of two matrices.

```
void main()
{
int a[5][5],b[5][5],c[5][5]={0};
int i,j,k,r1,r2,c1,c2;
printf("Enter the dimensions of matrix A");
scanf("%d%d",&r1,&c1);
printf("Enter the dimensions of matrix B");
scanf("%d%d",&r2,&c2);
if(c1==r2)
{
printf("Enter the elements of matrix A");
for(i=0;i<r1;i++) {
    for(j=0;j<c1;j++) {
        scanf("%d",&a[i][j]);
    }
}
printf("Enter the elements of matrix B");
for(i=0;i<r2;i++) {
    for(j=0;j<c2;j++) {
        scanf("%d",&b[i][j]);
    }
}
for(i=0;i<r1;i++) {
    for(j=0;j<c2;j++){
        for(k=0;k<c1;k++) {

        c[i][j]=c[i][j]+a[i][k]*b[k][j];
        }
    }
}
printf("The resultant matrix C\n");
for(i=0;i<r1;i++) {
    for(j=0;j<c2;j++) {
                printf("%3d",c[i][j]);
        }
    printf("\n");
}
}//if close
else{
printf("Invalid");
}
}//main
```

```
Enter the dimensions of matrix A 2 2
Enter the dimensions of matrix B 2 3

Enter the elements of matrix A
1 0
0 1
Enter the elements of matrix B
1 2 3
4 5 6
The resultant matrix C
1 2 3
4 5 6
```

Write a program to find transpose of matrix

```
void   main()
{
    int a[5][5],[5][5];
    int i,j,r,c;
    printf("\n\t Enter the order of the matrix A: ");
    scanf("%d%d",&r,&c);
    for(i=0;i<r;i++)
    {
        for(j=0;j<c ; j++)
        scanf("%d",&a[i][j]);

    }
    for(i=0;i<r;i++)
    {
        printf("\n");
        for(j=0;j<c;j++)
            printf("\t  %d",a[i][j]);
        }
```

```
for(i=0;i<c;i++)
{
    for(j=0;j<r;j++)
        b[i][j]=a[j][i];
}
printf("Transpose matrix");
for(i=0;i<c;i++)
{
    printf("\n");
    for(j=0;j<r;j++)
        printf("\t  %d",b[i][j]);
            }
}
```