Module 3:
Introduction to Arrays

Contents

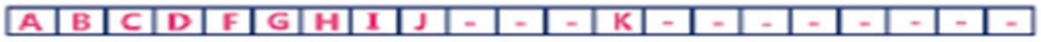
- One dimensional array
- Multidimensional array
- Declaration and Initialization of Arrays
- Reading and Displaying arrays
- Character Arrays and String

3.1 Arrays

An array is a sequence of data item of homogeneous values (same type).

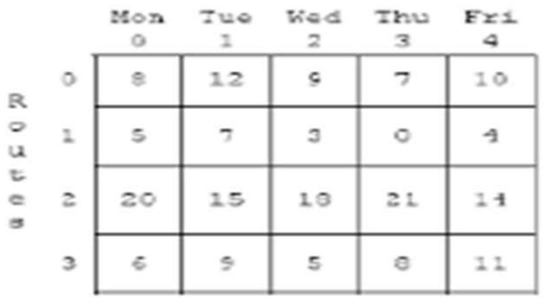
Arrays are of two types:

1. One dimensional arrays (1D array)
2. Multidimensional arrays (2D, #D, etc.)



A	B	C	D	F	G	H	I	J	-	-	-	K	-	-	-	-	-	-	-	-

1D array



Routes		Mon 0	Tue 1	Wed 2	Thu 3	Fri 4
	0	8	12	9	7	10
	1	5	7	3	0	4
	2	20	15	18	21	14
	3	6	9	5	8	11

2D array

Arrays

- An array is defined as an ordered set of similar data items.
- All the data items of an array are stored in consecutive memory locations in memory.
- The elements of an array are of same data type and each item can be accessed using the same name.

Declaration of an array:-

- We know that all the variables are declared before the are used in the program.
- Similarly, an array must be declared before it is used. During declaration, the size of the array has to be specified.
- The size used during declaration of the array informs the compiler to allocate and reserve the specified memory locations.

Arrays

Syntax:- data_type array_name[n];

where, n is the number of data items (or) index(or) dimension.

0 to (n-1) is the range of array.

Example:

```
int a[5];
float x[10];
```

Arrays

Initialization of Arrays:-
The different types of initializing arrays:

1. At Compile time
(i) Initializing all specified memory locations.
(ii) Partial array initialization
(iii) Initialization without size.
(iv) String initialization.

2. At Run Time

Arrays

1. Compile Time Initialization

- We can initialize the elements of arrays in the same way as the ordinary variables when they are declared.
- The general form of initialization of arrays is
  Type array-name[size]={ list of values};

Arrays

(i) Initializing all specified memory locations:- Arrays can be initialized at the time of declaration when their initial values are known in advance.

- Array elements can be initialized with data items of type int, char etc.

Ex:- int a[5]={10,15,1,3,20};

```
float b[3]={0.2,2.1,4.5};
printf("b[0]=%f\tb[1]=%f\tb[2]=%f",b[0],b[1],b[2]);
```

During compilation, 5 contiguous memory locations are reserved by the compiler for the variable a and all these locations are initialized as shown in figure.

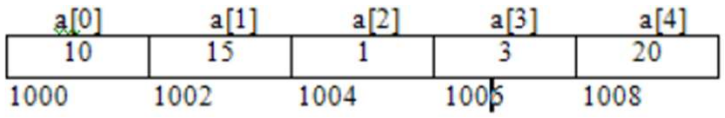


Fig: Initialization of int Arrays

Ex:-

int a[3]={9,2,4,5,6}; //error: no. of initial values are more than the size of array.

Arrays

1. Compile Time Initialization

(ii) Partial array initialization:- Partial array initialization is possible in c language. If the number of values to be initialized is less than the size of the array , then the elements will be initialized to zero automatically.

Ex:-

int a[5]={10,15};

Eventhough compiler allocates 5 memory locations, using this declaration statement; the compiler initializes first two locations with 10 and 15, the next set of memory locations are automatically initialized to 0's by compiler as shown in figure.

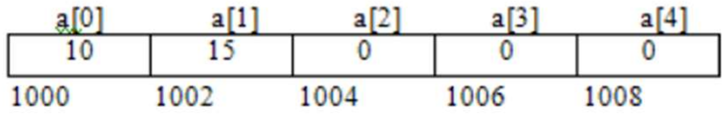


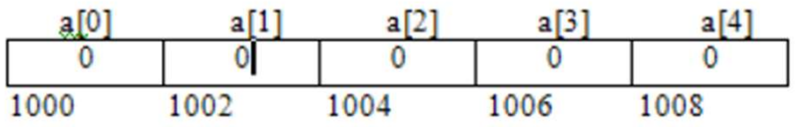
Fig: Partial Array Initialization

Arrays

1. Compile Time Initialization

Initialization with all zeros:-
Ex:-

int a[5]={0};



Arrays

1. Compile Time Initialization

(iii) Initialization without size:- Consider the declaration along with the initialization.
Ex:-
char b[]={'C','O','M','P','U','T','E','R'};

In this declaration, eventhough we have not specified exact number of elements to be used in array b, the array size will be set of the total number of initial values specified. So, the array size will be set to 8 automatically. the array b is initialized as shown in figure.

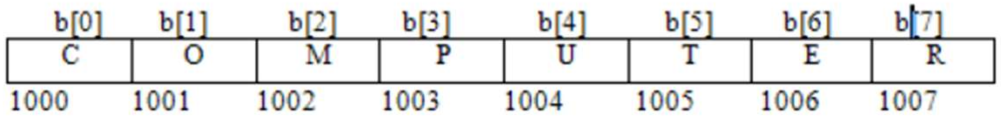


Fig: Initialization without size

Ex:- int ch[]={1,0,3,5} // array size is 4

Arrays

1. Compile Time Initialization

(iv) Array initialization with a string: - Consider the declaration with string initialization.
Ex:-
char b[]="COMPUTER";
The array b is initialized as shown in figure.

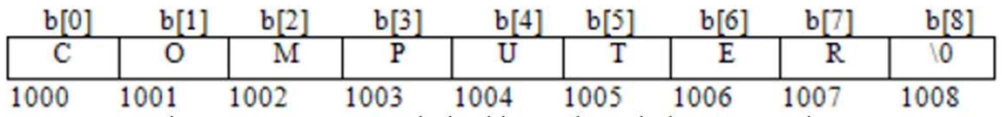


Fig: Array Initialized with a String

Eventhough the string "COMPUTER" contains 8 characters, because it is a string. It always ends with null character. So, the array size is 9 bytes (i.e., string length 1 byte for null character).
Ex:-
char b[9]="COMPUTER";
char b[8]="COMPUTER";

Arrays

1. Compile Time Initialization

(iv) Array initialization with a string: - Consider the declaration with string initialization.

Ex:-

char b[]="COMPUTER";

The array b is initialized as shown in figure.

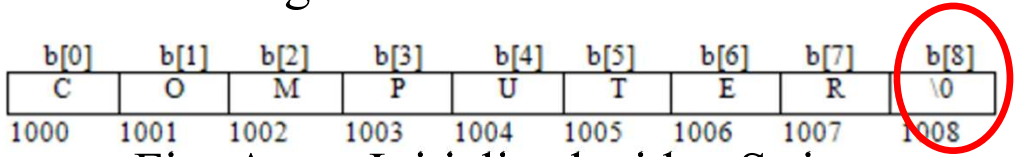


Fig: Array Initialized with a String

Eventhough the string "COMPUTER" contains 8 characters, because it is a string. It always ends with null character. So, the array size is 9 bytes (i.e., string length 1 byte for null character).

Ex:-

char b[9]="COMPUTER"; // correct

char b[8]="COMPUTER"; // wrong

Arrays

Task

- Consider an array of integer, float and character with **Compile Time Initialization**
- Display all elements in an array
- Display any particular value/ character from the array

x[0]= x[1]= x[2]= x[3]=

And as

x[0]=

x[1]=…

Arrays

2. Run Time Initialization

An array can be explicitly initialized at run time. This approach is usually applied for initializing large arrays.

Ex:- scanf can be used to initialize an array.

Int x[3];

Scanf(“%d%d%d”,&x[0],&x[1],&x[2]);

The above statements will initialize array elements with the values entered through the key board.

TASK

Consider an array of 4 elements and display using scanf and printf as

x[0]=   x[1]=   x[2]=   x[3]=

And as

x[0]=

x[1]=…

Arrays

2. Run Time Initialization

OR

```
float Sum[150];
For(i=0;i<100;i=i+1)
{
If(i<50)
Sum[i]=0.0;
Else
Sum[i]=1.0;
}
```

Arrays

2. Run Time Initialization

OR

```
float Sum[150];
For(i=0;i<100;i=i+1)
{
If(i<50)
Sum[i]=0.0;
Else
Sum[i]=1.0;
}
```

The first 50 elements of the array sum are initialized to 0 while the remaining 50 are initialized to 1.0 at run time.

Arrays

2 D arrays

An array consisting of two subscripts is known as two-dimensional array. These are often known as array of the array. In two dimensional arrays the array is divided into rows and columns,. These are well suited to handle the table of data. In 2-D array we can declare an array as :

Declaration:-

Data_type array_name[row_size][column_size] = {{list of first row elements},
{list of second row elements},….
{list of last row elements}};

Ex:- int arr[3][3]; **arr[row][col]**

where first index value shows the number of the rows and second index value shows the no. of the columns in the array.

Initialization:-

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

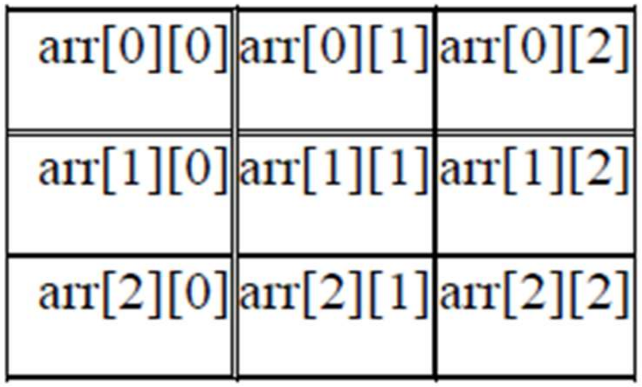
Arrays

2 D arrays: Memory storage

These are stored in the memory as given below.

1. Row-Major order Implementation
2. Column-Major order Implementation

In Row-Major Implementation of the arrays, the arrays are stored in the memory in terms of the row design, i.e. first the first row of the array is stored in the memory then second and so on. Suppose we have an array named arr having 3 rows and 3 columns then it can be stored in the memory in the following manner :



arr[0][0]	arr[0][1]	arr[0][2]
arr[1][0]	arr[1][1]	arr[1][2]
arr[2][0]	arr[2][1]	arr[2][2]

Arrays

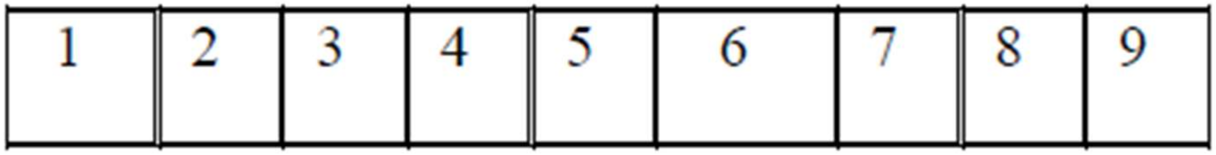
2 D arrays: Memory storage: Row-Major Implementation

int arr[3][3];

Thus an array of 3*3 can be declared as follows :

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

and it will be represented in the memory with row major implementation as follows :



1	2	3	4	5	6	7	8	9

Arrays

2 D arrays: Memory storage: Column-Major Implementation

The arrays are stored in the memory in the term of the column design, i.e. the first column of the array is stored in the memory then the second and so on.

int arr[3][3];

Thus an array of 3*3 can be declared as follows :

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

and it will be represented in the memory with column major implementation as follows :

Arrays

2 D arrays: Memory storage: Column-Major Implementation

```
int arr[3][3];
```

Thus an array of 3*3 can be declared as follows :

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

1	**2**	**3**
4	**5**	**6**
7	**8**	**9**

and it will be represented in the memory with column major implementation as follows :

1	4	7	2	5	8	3	6	9

Arrays

2 D arrays:

To initialize values for variable length arrays we can use scanf statement & loop constructs.

```
int arr[5][5];
for (i=0 ; i<3; i++)
for(j=0;j<3;j++)
scanf("%d",&arr[i][j]);
```

Task:
Consider 2-D array, display the 2-D array and also display any particular element or specified by user from it

Arrays

2 D arrays:

To initialize values for variable length arrays we can use scanf statement & loop constructs.

```
int arr[5][5];
for (i=0 ; i<3; i++)
for(j=0;j<3;j++)
scanf("%d",&arr[i][j]);
```

Task:

Consider 2-D array, display the 2-D array and also display any particular element

Arrays

Multidimensional #D arrays:

Syntax:

Data_type arrat_name[size1][size2][size3]------[sizeN];

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

10, 11, 12,
13, 14, 15,
16, 17, 18,

19, 20, 21,
22, 23, 24,
25, 26, 27 };
```

/* here we have divided array into grid for sake of convenience as in above declaration we have created 3 different grids, each have rows and columns */

If we want to access the element the in 3-D array we can do it as follows :

```
printf("%d",&arr[2][2][2]);
```

Output=?

Arrays

Multidimensional #D arrays:

Syntax:

Data_type arrat_name[size1][size2][size3]------[sizeN];

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

10, 11, 12,
13, 14, 15,
16, 17, 18,

19, 20, 21,
22, 23, 24,
25, 26, 27 };
```

/* here we have divided array into grid for sake of convenience as in above declaration we have created 3 different grids, each have rows and columns */

If we want to access the element the in 3-D array we can do it as follows :

printf("%d",a[2][2][2]);

Output=?

27

Array indexing start with zero

Arrays

Task

Addition of 2-D array (2 * 2)
Display both the array
Display added array

Task: Addition of 2-D array (2 * 2)

```
#include<stdio.h>
void main()
{
int i,j;
int a[2][2]={1,2,3,4};
int b[2][2]={1,2,3,4};
int c[2][2];

for(i=0;i<2;i++)
{
for(j=0;j<2;j++)
{
printf("%2d",a[i][j]);
}
printf("\n\n");
}
for(i=0;i<2;i++)
{
for(j=0;j<2;j++)
{
printf("%2d",b[i][j]);
}
printf("\n\n");
}


printf("\n The addition of given two matrices is\n");
for(i=0;i<2;i++)
{
for(j=0;j<2;j++)
{
c[i][j]=a[i][j]+b[i][j];
}
}
for(i=0;i<2;i++)
{
for(j=0;j<2;j++)
{
printf("%2d",c[i][j]);
}
printf("\n\n");
}
getch();
}
```

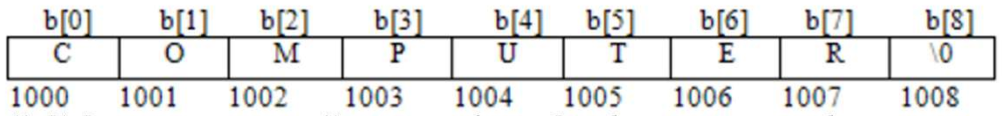
3.2 Character Arrays and String

Array declaration and initialization with a character/string: - Consider the declaration with string initialization.

Ex:-

char b[]="COMPUTER";

The array b is initialized as shown in figure.



Eventhough the string "COMPUTER" contains 8 characters, because it is a string. It always ends with null character. So, the array size is 9 bytes (i.e., string length 1 byte for null character).

Ex:-

char b[9]="COMPUTER";

char b[8]="COMPUTER";

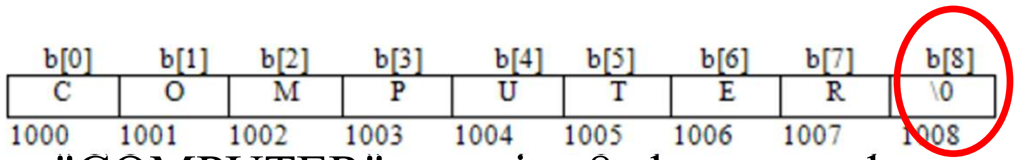
Arrays

Array declaration and initialization with a character/string: - Consider the declaration with string initialization.

Ex:-

char b[]="COMPUTER";

The array b is initialized as shown in figure.



Eventhough the string "COMPUTER" contains 8 characters, because it is a string. It always ends with null character. So, the array size is 9 bytes (i.e., string length 1 byte for null character).

Ex:-

char b[9]="COMPUTER"; // correct

char b[8]="COMPUTER"; // wrong

Arrays

Task

- Consider an array of integer, float and **character** with **Compile Time Initialization**
- Display all elements in an array
- Display any particular value/ character from the array

x[0]= x[1]= x[2]= x[3]= And as
x[0]=
x[1]=…

```
#include <stdio.h>
int main ()
{
  char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
  printf("Greeting message: %s\n", greeting );
  return 0;
}
```

Output?

Arrays

Task

- Consider an array of integer, float and **character** with **Compile Time Initialization**
- Display all elements in an array
- Display any particular value/ character from the array

x[0]= x[1]= x[2]= x[3]= And as

x[0]=

x[1]=…

```
#include <stdio.h>
int main ()
{
Message:Hello
   char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
   printf("Greeting message: %s\n", greeting );
   return 0;
}
```

Output?

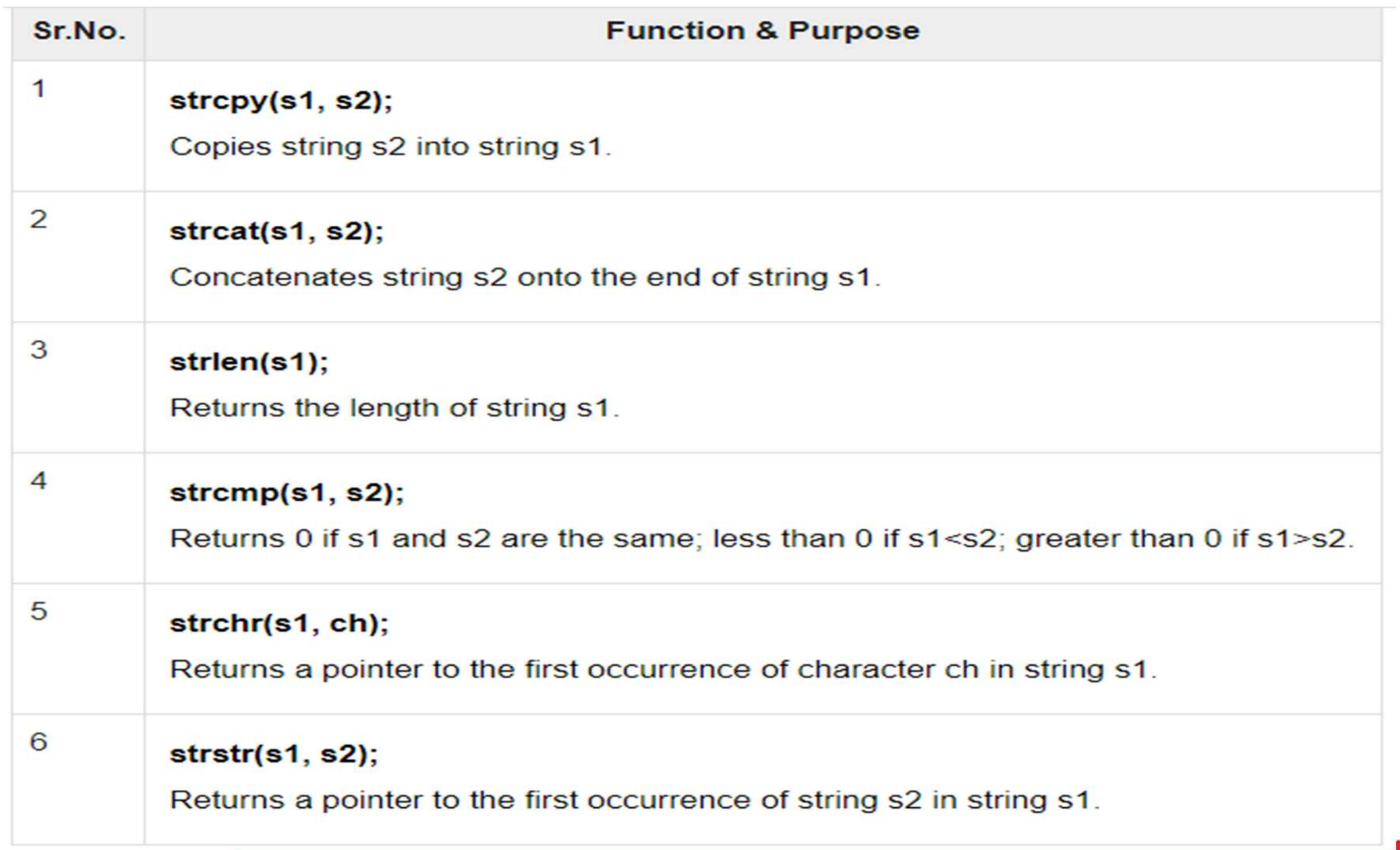
Greeting

Arrays

```
#include <stdio.h>
#include <string.h>
int main()
{
   /* String Declaration*/
   char nickname[20];
   printf("Enter your Nick name:");

   /* I am reading the input string and storing it in
nickname
    * Array name alone works as a base address of array
so
    * we can use nickname instead of &nickname here
    */
   scanf("%s", nickname);

   /*Displaying String*/
   printf("%s",nickname);

   return 0;
}
```

```
#include <stdio.h>
#include <string.h>
int main()
{
   /* String Declaration*/
   char nickname[20];

   /* Console display using puts */
   puts("Enter your Nick name:");

   /*Input using gets*/
   gets(nickname);

   puts(nickname);

   return 0;
}
```

Arrays: String Handling Functions



Sr.No.	Function & Purpose
1	**strcpy(s1, s2);** Copies string s2 into string s1.
2	**strcat(s1, s2);** Concatenates string s2 onto the end of string s1.
3	**strlen(s1);** Returns the length of string s1.
4	**strcmp(s1, s2);** Returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 if s1>s2.
5	**strchr(s1, ch);** Returns a pointer to the first occurrence of character ch in string s1.
6	**strstr(s1, s2);** Returns a pointer to the first occurrence of string s2 in string s1.

Arrays: String Handling Functions

```
#include <stdio.h>
#include <string.h>
int main () {

  char str1[12] = "Hello";
  char str2[12] = "World";
  char str3[12];
  int  len ;

  /* copy str1 into str3 */
  strcpy(str3, str1);
  printf("strcpy( str3, str1) :  %s\n", str3 );

  /* concatenates str1 and str2 */
  strcat( str1, str2);
  printf("strcat( str1, str2):   %s\n", str1 );

 /* total lenghth of str1 after
concatenation */
  len = strlen(str1);
  printf("strlen(str1) :  %d\n", len );

  return 0;
}
```

Output?

Arrays: String Handling Functions

```
#include <stdio.h>
#include <string.h>
int main () {

  char str1[12] = "Hello";
  char str2[12] = "World";
  char str3[12];
  int  len ;

  /* copy str1 into str3 */
  strcpy(str3, str1);
  printf("strcpy( str3, str1) :  %s\n", str3 );

  /* concatenates str1 and str2 */
  strcat( str1, str2);
  printf("strcat( str1, str2):   %s\n", str1 );

  /* total length of str1 after
concatenation */
   len = strlen(str1);
   printf("strlen(str1) :  %d\n", len );

   return 0;
}
```

Output?

```
strcpy( str3, str1) :  Hello
strcat( str1, str2):   HelloWorld
strlen(str1) :  10
```

Arrays: String Handling Functions

strlen vs sizeof

```
#include <stdio.h>
#include <string.h>
int main()
{
    char str1[20] = "BeginnersBook";
    printf("Length of string str1: %d",
strlen(str1));
    printf(“Size of string str1: %d", sizeof(str1));
    return 0;
}
```

Output?

Arrays: String Handling Functions

strlen vs sizeof

```
#include <stdio.h>
#include <string.h>
int main()
{
    char str1[20] = "BeginnersBook";
    printf("Length of string str1: %d",
strlen(str1));
    printf(“Size of string str1: %d",
sizeof(str1));
    return 0;
}
```

Output?

strlen(str1) returned value 13.
sizeof(str1) would return value 20 as the array size is 20 (see the first statement in main function).

Arrays: String Handling Functions

String function – strcmp, strncmp

- If string1 < string2 OR string1 is a substring of string2 then it would result in a negative value. If string1 > string2 then it would return positive value.
- If **string1 == string2** then you would get **0(zero)** when you use this function for compare strings.

String function – strcmp, strncmp

```
#include <stdio.h>
#include <string.h>
int main()
{
    char s1[20] = "BeginnersBook";
    char s2[20] = "BeginnersBook.COM";
    if (strcmp(s1, s2) ==0)
    {
      printf("string 1 and string 2 are equal");
    }else
     {
       printf("string 1 and 2 are different");
     }
    return 0;
}
```

```
#include <stdio.h>
#include <string.h>
int main()
{
    char s1[20] = "BeginnersBook";
    char s2[20] = "BeginnersBook.COM";
    /* below it is comparing first 8 characters of s1 and s2*/
    if (strncmp(s1, s2, 8) ==0)
    {
       printf("string 1 and string 2 are equal");
    }else
    {
       printf("string 1 and 2 are different");
    }
    return 0;
}
```

Arrays: String Handling Functions

String function – strcmp, strncmp

```
#include <stdio.h>
#include <string.h>
int main()
{
    char s1[20] = "BeginnersBook";
    char s2[20] = "BeginnersBook.COM";
    if (strcmp(s1, s2) ==0)
    {
      printf("string 1 and string 2 are equal");
    }else
     {
       printf("string 1 and 2 are different");
     }
    return 0;
}
```

string 1 and 2 are different

```
#include <stdio.h>
#include <string.h>
int main()
{
    char s1[20] = "BeginnersBook";
    char s2[20] = "BeginnersBook.COM";
    /* below it is comparing first 8 characters of s1 and s2*/
    if (strncmp(s1, s2, 8) ==0)
    {
       printf("string 1 and string 2 are equal");
    }else
    {
       printf("string 1 and 2 are different");
    }
    return 0;
}
```

string 1 and 2 are equal

Arrays: String Handling Functions

String function – strchr & strstr

```
#include <stdio.h>
#include <string.h>
int main () {
   char str[] = "This is just a String";
   char ch = 's';
   char *p;
   p = strchr(str, ch);
   printf("String starting from %c is: %s",
ch, p);
   return 0;
}
```

```
#include <stdio.h>
#include <string.h>
int main () {
   char str[20] = "Hello, how are you?";
   char searchString[10] = "you";
   char *result;

   /* This function returns the pointer of the first
occurrence
    * of the given string (i.e. searchString)     */

   result = strstr(str, searchString);
   printf("The substring starting from the given string:
%s", result);
   return 0;
}
```

String function – strchr & strstr

```
#include <stdio.h>
#include <string.h>
int main () {
   char str[] = "This is just a String";
   char ch = 's';
   char *p;
   p = strchr(str, ch);
   printf("String starting from %c is: %s", ch,
p);
   return 0;
}
```

String starting from s is: s is just a String

```
#include <stdio.h>
#include <string.h>
int main () {
   char str[20] = "Hello, how are you?";
   char searchString[10] = "you";
   char *result;

   /* This function returns the pointer of the first occurrence
    * of the given string (i.e. searchString)     */

   result = strstr(str, searchString);
   printf("The substring starting from the given string: %s",
result);
   return 0;
}
```

The substring starting from the given string: you ?

Arrays: String Handling Functions

String function – strchr & strstr

find the first occurrence of a small string (the **"needle"**) within a larger string (the **"haystack"**)

```
int main () {
  char haystack[20] = "TutorialsPoint";
  char needle[10] = "Point";
  char *ret;

  ret = strstr(haystack, needle);

  printf("The substring is: %s\n", ret);

  return(0);
}
```

```
int main () {
  char haystack[20] = "TutorialsPoint";
  char needle[10] = “Tutorials";
  char *ret;

  ret = strstr(haystack, needle);

  printf("The substring is: %s\n", ret);

  return(0);
}
```

Arrays: String Handling Functions

String function – strchr & strstr

```
int main () {
  char haystack[20] = "TutorialsPoint";
  char needle[10] = "Point";
  char *ret;

  ret = strstr(haystack, needle);

  printf("The substring is: %s\n", ret);

  return(0);
}
```

The substring is: Point

```
int main () {
  char haystack[20] = "TutorialsPoint";
  char needle[10] = “Tutorials";
  char *ret;

  ret = strstr(haystack, needle);

  printf("The substring is: %s\n", ret);

  return(0);
}
```

The substring is: TutorialsPoint

Arrays: String Handling Functions

String function – strchr & strstr

The strstr() function returns a pointer to the start of "TutorialsPoint". The printf function then uses this pointer to print the string from that point to the end, which is the entire string, "TutorialsPoint".

```
The substring is: Point
```

```
The substring is: TutorialsPoint
```

Arrays: String Handling Functions

String function – strchr & strstr

The algorithm itself (strstr()) is identical in both cases. The difference lies in the **arguments** passed to the function, which changes the **result of the search** and, consequently, the **output**. The first code finds a match in the middle of the string, while the second finds a match at the beginning.

Write a program to find duplicate element in array

Use two arrays: one to store the input numbers (arr) and another (freq) to keep track of whether an element has already been counted.

Algorithm

Initialization:

The program first prompts the user to enter the size of the array, n.

It declares two integer arrays: arr of size n to store the user-inputted numbers, and freq of size n to store the frequency of each element.

A counter variable count is initialized to 0 to store the final count of duplicate elements.

Input and freq Array Setup:

The code uses a for loop to read n elements from the user into the arr array. Simultaneously, it initializes all elements of the freq array to -1. This value acts as a flag, indicating that the element at the corresponding index in arr has not yet been processed or counted.

Write a program to find duplicate element in array

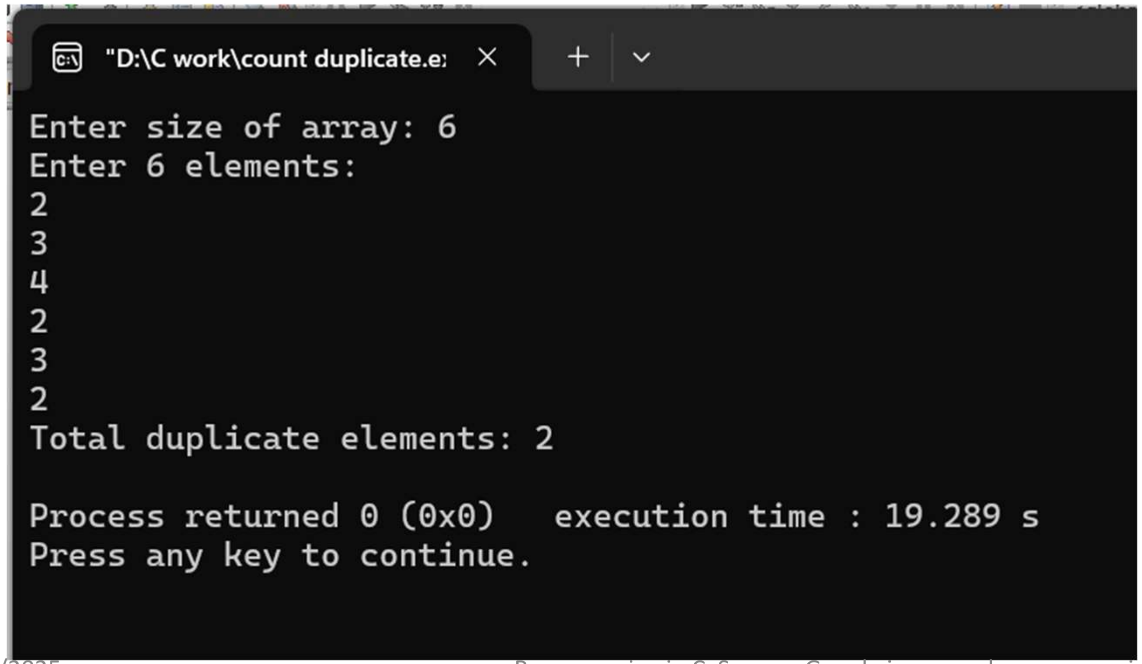
Counting Duplicates:
The core of the algorithm is a **nested for loop**.
The **outer loop** iterates through each element of the arr array, from index i = 0 to n-1.
Inside the outer loop, there's a check: if (freq[i] == -1). This condition ensures that the algorithm only processes an element that hasn't been counted as a duplicate before.

The **inner loop** starts at j = i + 1 and iterates to the end of the array. Its purpose is to compare the element arr[i] with all subsequent elements arr[j].

If a match is found (arr[i] == arr[j]), it means a duplicate has been found. The duplicateCount for arr[i] is incremented, and the freq[j] is set to 0. Setting freq[j] to 0 is a crucial step; it marks the duplicate element at index j as "visited" or "counted," preventing it from being processed again by the outer loop. After the inner loop completes, if duplicateCount for the element at index i is greater than 1, it means that arr[i] had at least one duplicate.

If duplicates are found, the count of total duplicate elements is incremented, and freq[i] is set to duplicateCount (or any value other than -1, marking it as processed).

Output:
Finally, the program prints the total number of duplicate elements stored in the count variable.
This approach is effective because it uses a auxiliary array (freq) to efficiently mark and skip elements that have already been identified as duplicates, avoiding redundant comparisons and ensuring each unique duplicate element is counted only once.



```
Enter size of array: 6
Enter 6 elements:
2
3
4
2
3
2
Total duplicate elements: 2

Process returned 0 (0x0)   execution time : 19.289 s
Press any key to continue.
```

```
#include <stdio.h>
int main()
{
    int n, i, j, count = 0;
    printf("Enter size of array: ");
    scanf("%d", &n);
    int arr[n], freq[n];
    printf("Enter %d elements: \n", n);
 for (i = 0; i < n; i++)
        {
        scanf("%d", &arr[i]);
        freq[i] = -1;
        }
    for (i = 0; i < n; i++)
    {
        int duplicateCount = 1;
        if (freq[i] == -1)
        {
            for (j = i + 1; j < n; j++)
            {
                if (arr[i] == arr[j])
                {
                    duplicateCount++;
                    freq[j] = 0;
                }
            }
            if (duplicateCount > 1)
            {
                count++;
                freq[i] = duplicateCount;
            }

        }
    }
    printf("Total duplicate elements: %d\n", count);
    return 0;
}
```