Module 3: Introduction to Arrays

Syllabus

3.1	Arrays: Introduction to One Dimensional Arrays, Multidimensional Arrays, Declaration and Initialization of Arrays, Reading and Displaying arrays
3.2	Character Arrays and Strings: Introduction, Declaring and Initializing String Variables, Reading Character and Writing Character, Reading and Writing Strings, various operation on strings, Implementation of string handling operations (from scratch)

Arrays

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

These similar elements could be all ints, or all floats, or all chars, etc.

Usually, the array of characters is called a ‘string’, whereas an array of ints or floats is called simply an 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 they 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.

Types of array

- Single-Dimensional Arrays (1-D Arrays)
- Multi-Dimensional Arrays

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];

Syntax:-data_type array_name[size1][size2]...[sizeN];

Example:

int matrix[3][4];

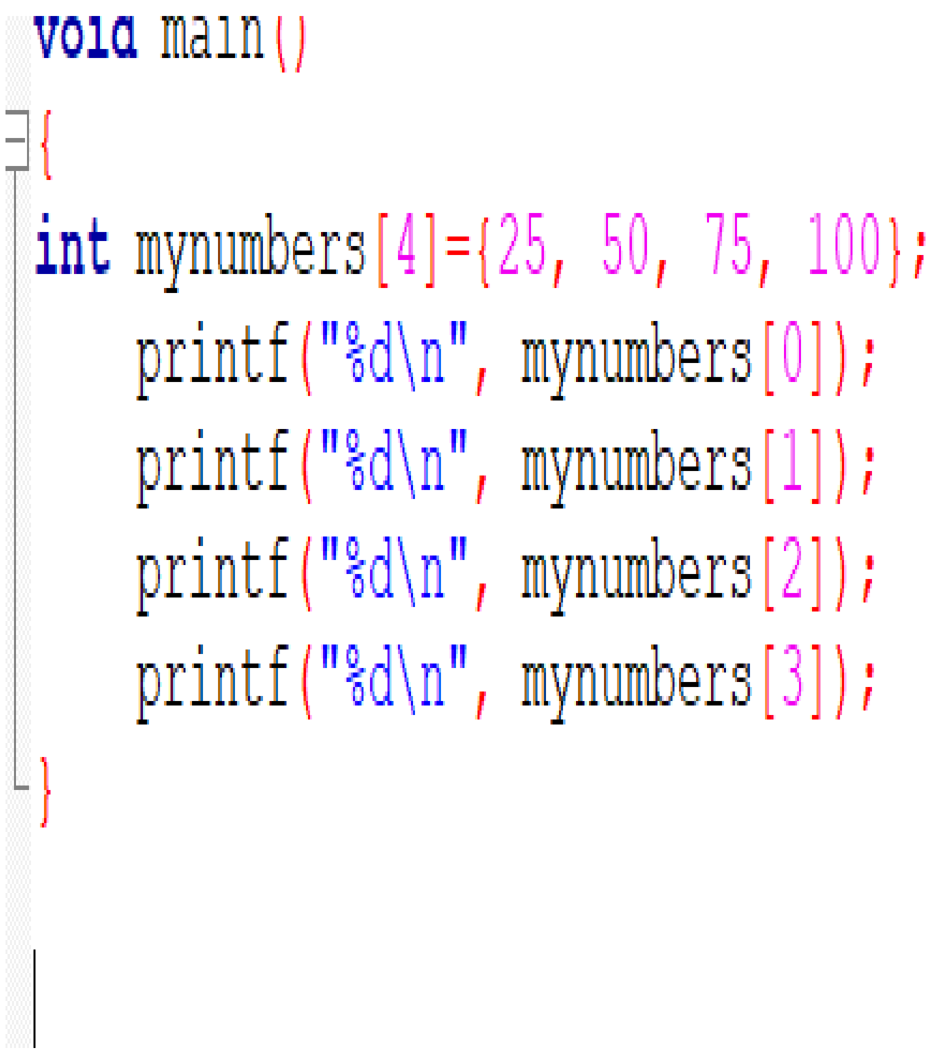
int x[3][4] 2D array

	Column 1	**Column 2**	**Column 3**	**Column 4**
ROW 1	X[0][0]	X[0][1]	X[0][2]	X[0][3]
ROW 2	X[1][0]	X[1][1]	X[1][2]	X[1][3]
ROW 3	X[2][0]	X[2][1]	X[2][2]	X[2][3]

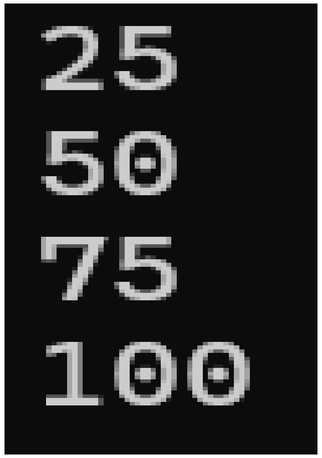
Array elements are always stored in contiguous memory locations.
Suppose we are having an array num[]={24,25,29,12,17}

24	25	29	12	17
6422016	6422020	6422024	6422028	6422032

Access the Elements of an Array

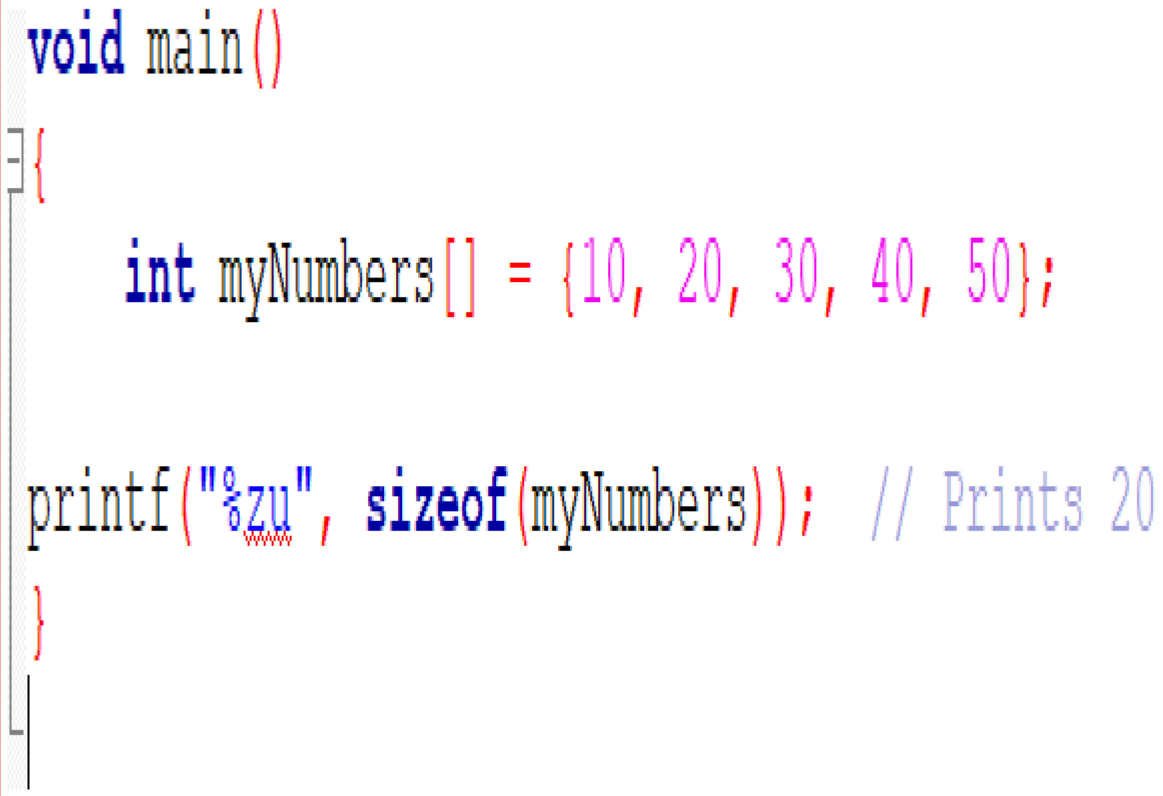


```c
void main()
{
int mynumbers[4]={25, 50, 75, 100};
    printf("%d\n", mynumbers[0]);
    printf("%d\n", mynumbers[1]);
    printf("%d\n", mynumbers[2]);
    printf("%d\n", mynumbers[3]);
}
```



```
25
50
75
100
```

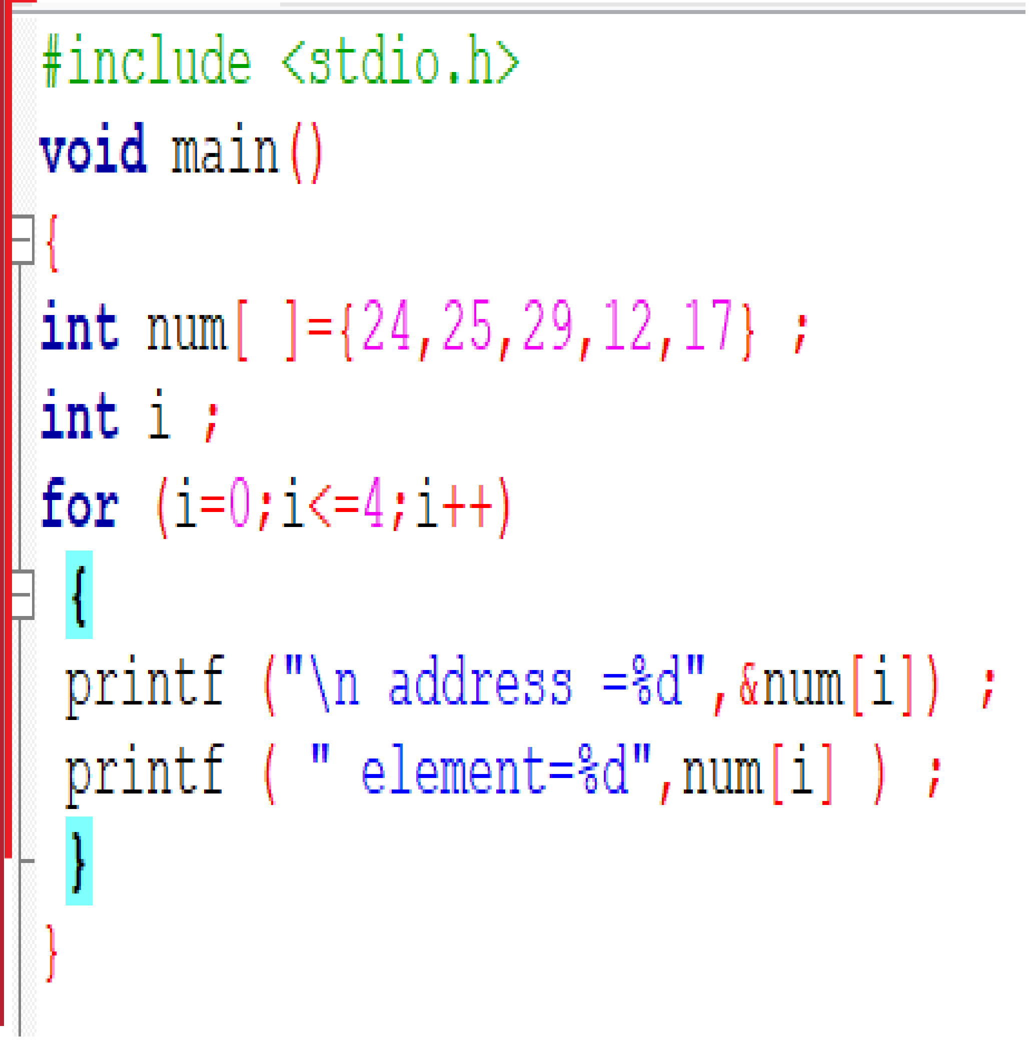
Get the size of an array



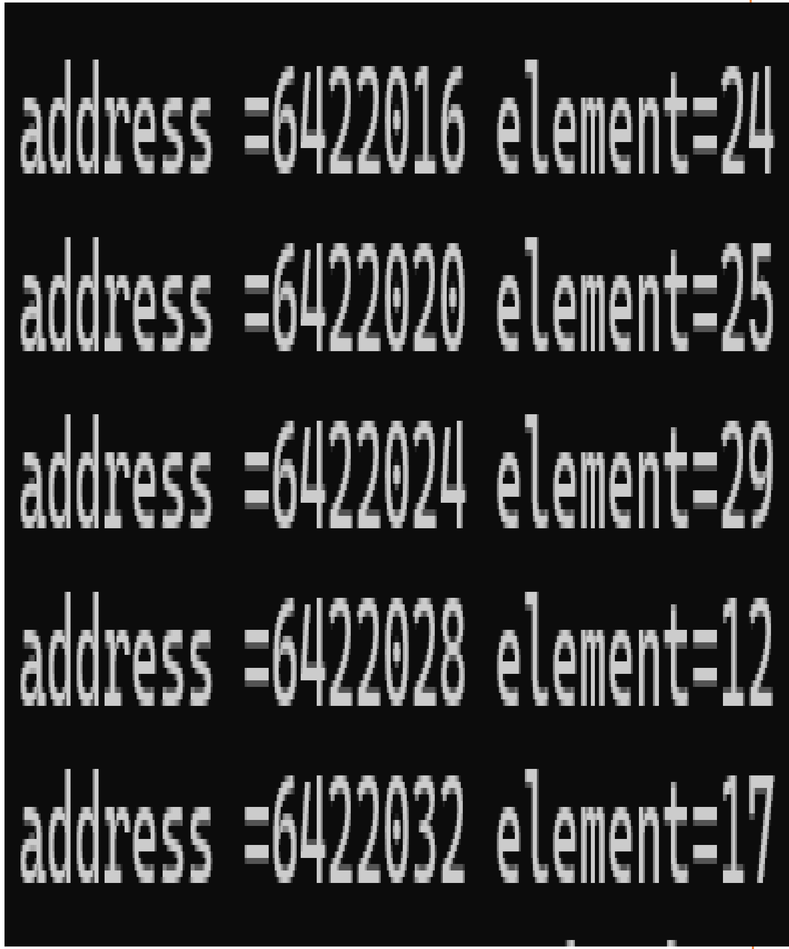
```
void main()
{
    int myNumbers[] = {10, 20, 30, 40, 50};

printf("%zu", sizeof(myNumbers));  // Prints 20
}
```

Note: an int type is usually 4 bytes, so from the example above, 4 x 5 (*4 bytes x 5 elements*) = **20 bytes**.

- Example:1

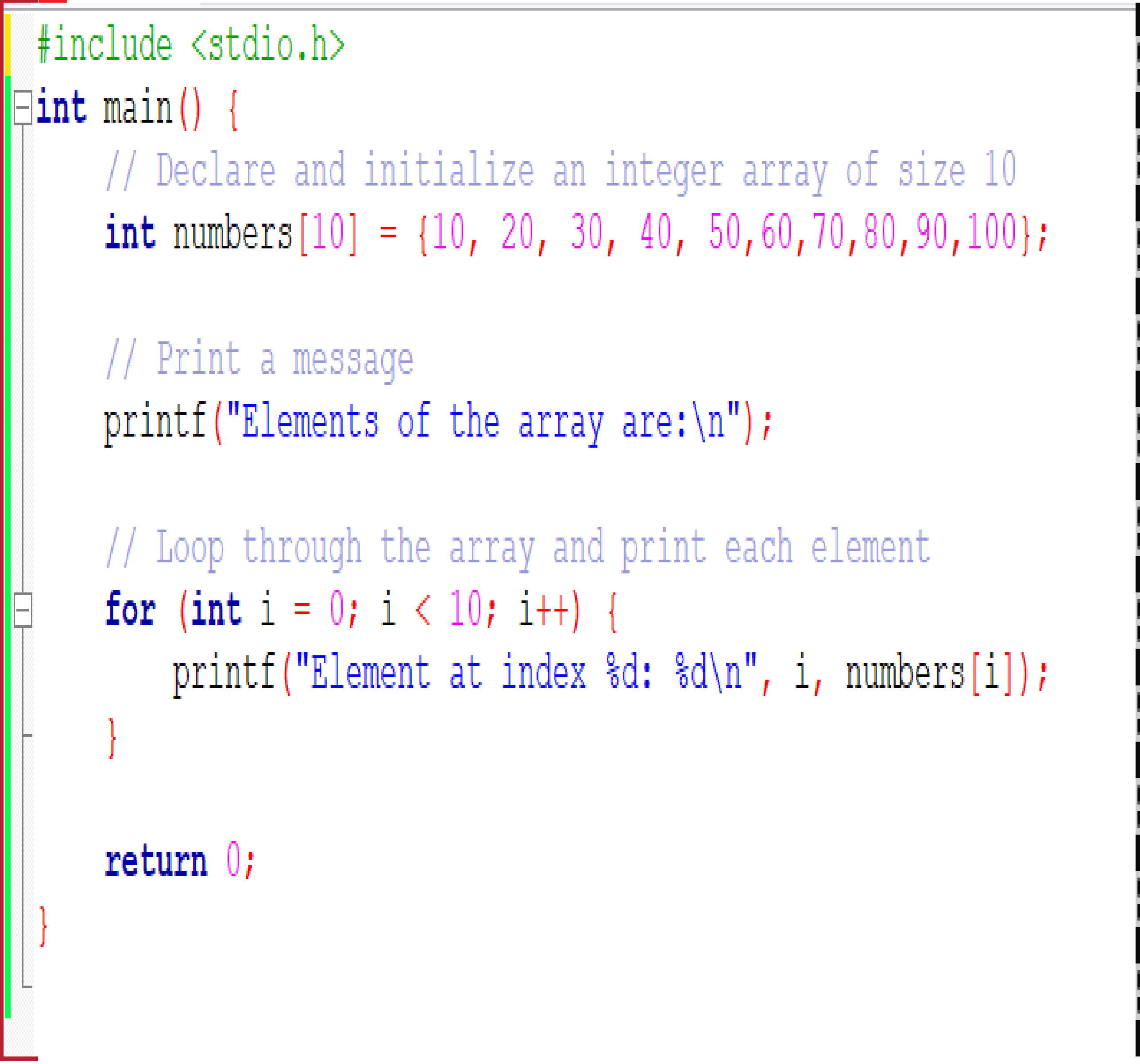


```
#include <stdio.h>
void main()
{
int num[ ]={24,25,29,12,17} ;
int i ;
for (i=0;i<=4;i++)
 {
 printf ("\n address =%d",&num[i]) ;
 printf ( " element=%d",num[i] ) ;
 }
}
```

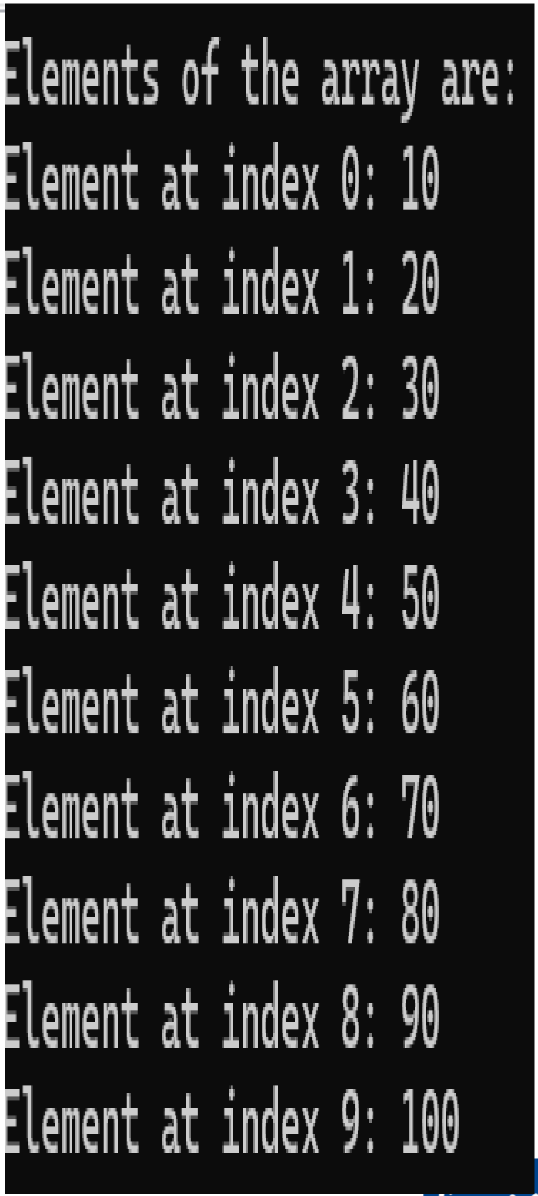


```
address =6422016 element=24
address =6422020 element=25
address =6422024 element=29
address =6422028 element=12
address =6422032 element=17
```

Example:2

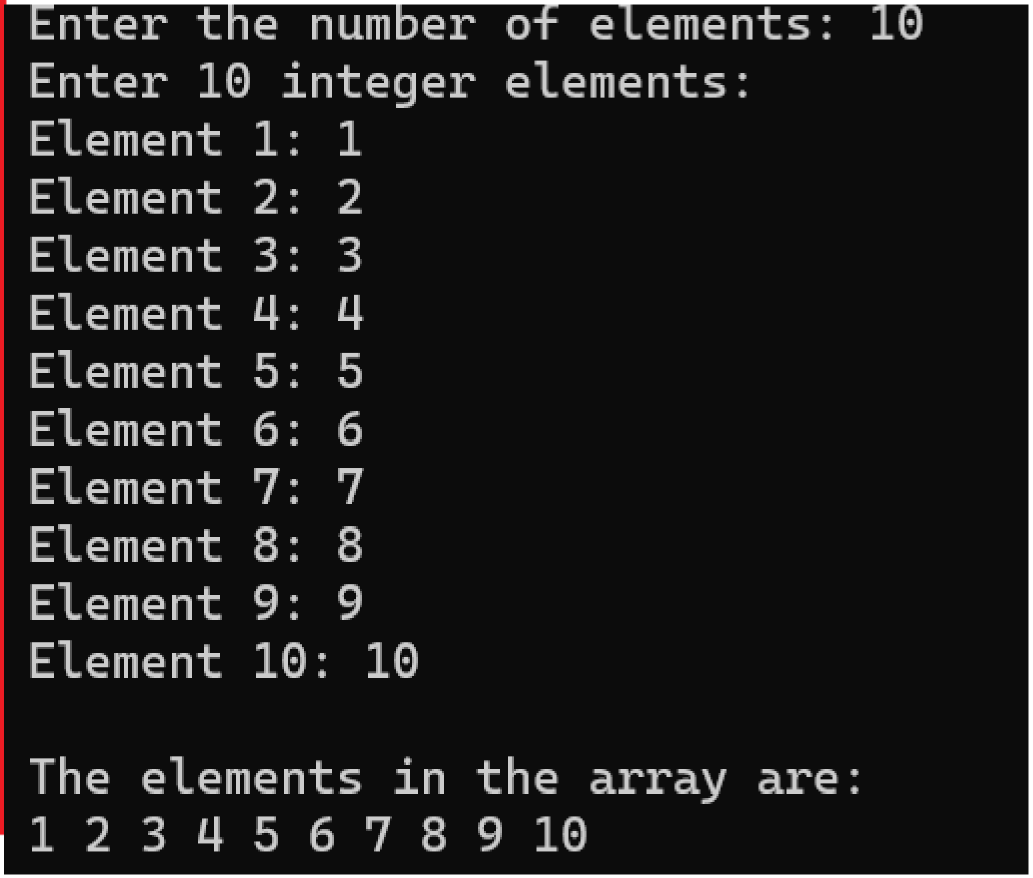


```c
#include <stdio.h>
int main() {
    // Declare and initialize an integer array of size 10
    int numbers[10] = {10, 20, 30, 40, 50,60,70,80,90,100};

    // Print a message
    printf("Elements of the array are:\n");

    // Loop through the array and print each element
    for (int i = 0; i < 10; i++) {
        printf("Element at index %d: %d\n", i, numbers[i]);
    }

    return 0;
}
```

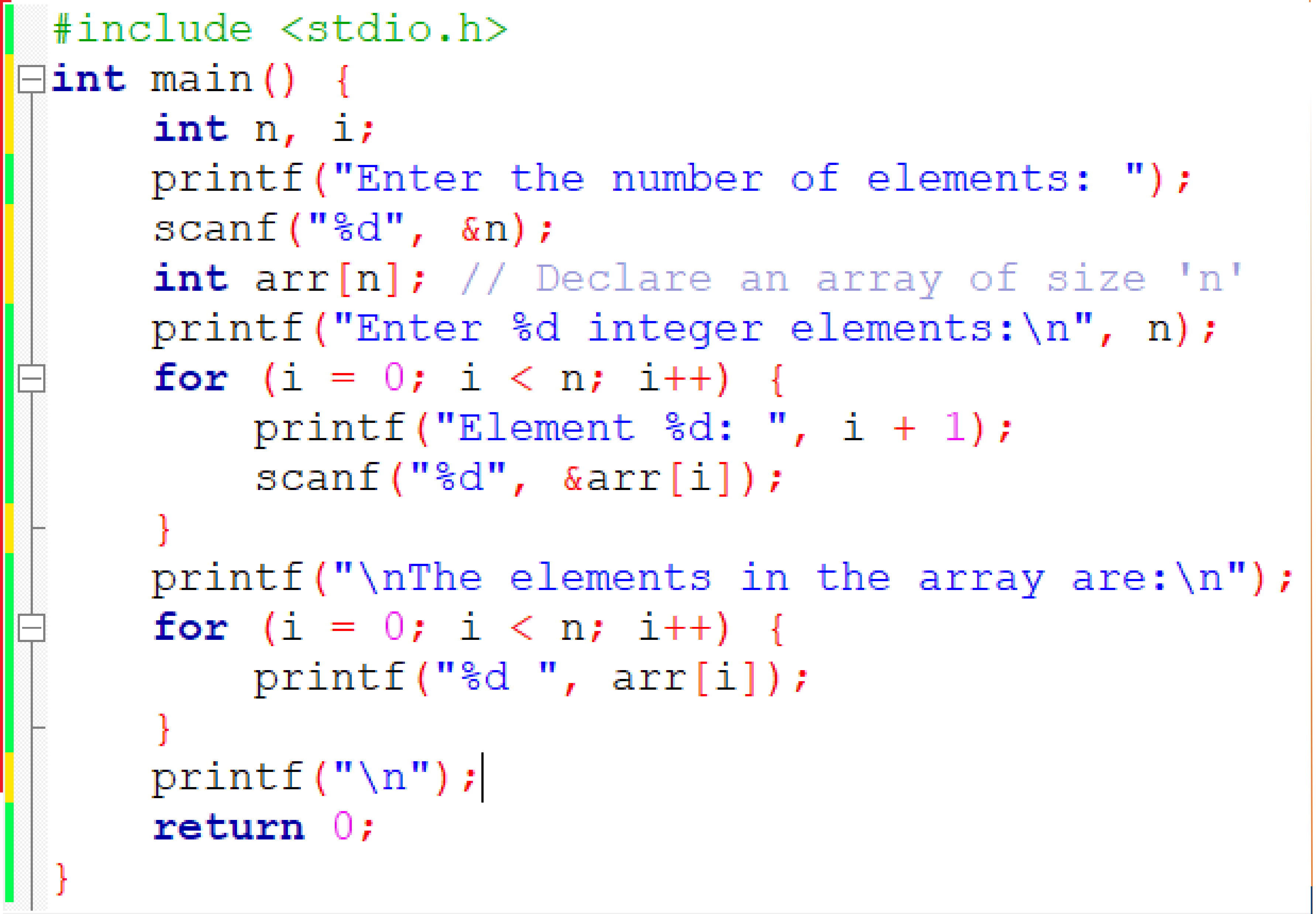


```
Elements of the array are:
Element at index 0: 10
Element at index 1: 20
Element at index 2: 30
Element at index 3: 40
Element at index 4: 50
Element at index 5: 60
Element at index 6: 70
Element at index 7: 80
Element at index 8: 90
Element at index 9: 100
```

Example3: Write a c program to the above mentioned output

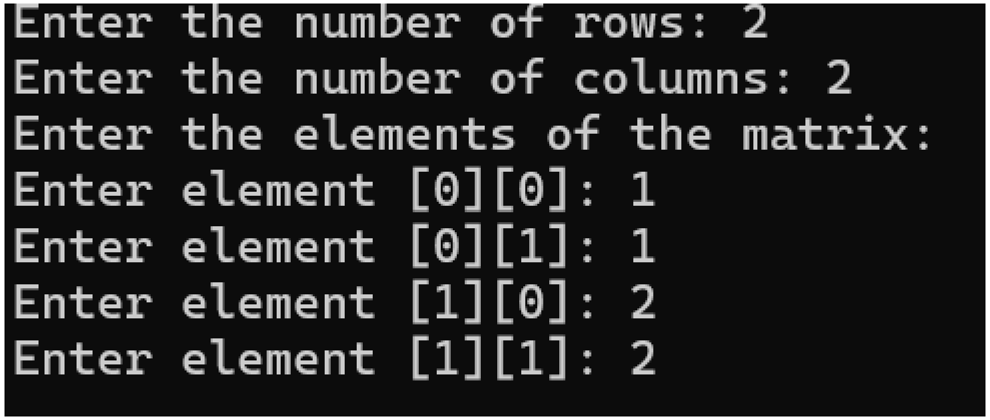


```
Enter the number of elements: 10
Enter 10 integer elements:
Element 1: 1
Element 2: 2
Element 3: 3
Element 4: 4
Element 5: 5
Element 6: 6
Element 7: 7
Element 8: 8
Element 9: 9
Element 10: 10

The elements in the array are:
1 2 3 4 5 6 7 8 9 10
```



```
#include <stdio.h>
int main() {
    int n, i;
    printf("Enter the number of elements: ");
    scanf("%d", &n);
    int arr[n]; // Declare an array of size 'n'
    printf("Enter %d integer elements:\n", n);
    for (i = 0; i < n; i++) {
        printf("Element %d: ", i + 1);
        scanf("%d", &arr[i]);
    }
    printf("\nThe elements in the array are:\n");
    for (i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
    return 0;
}
```

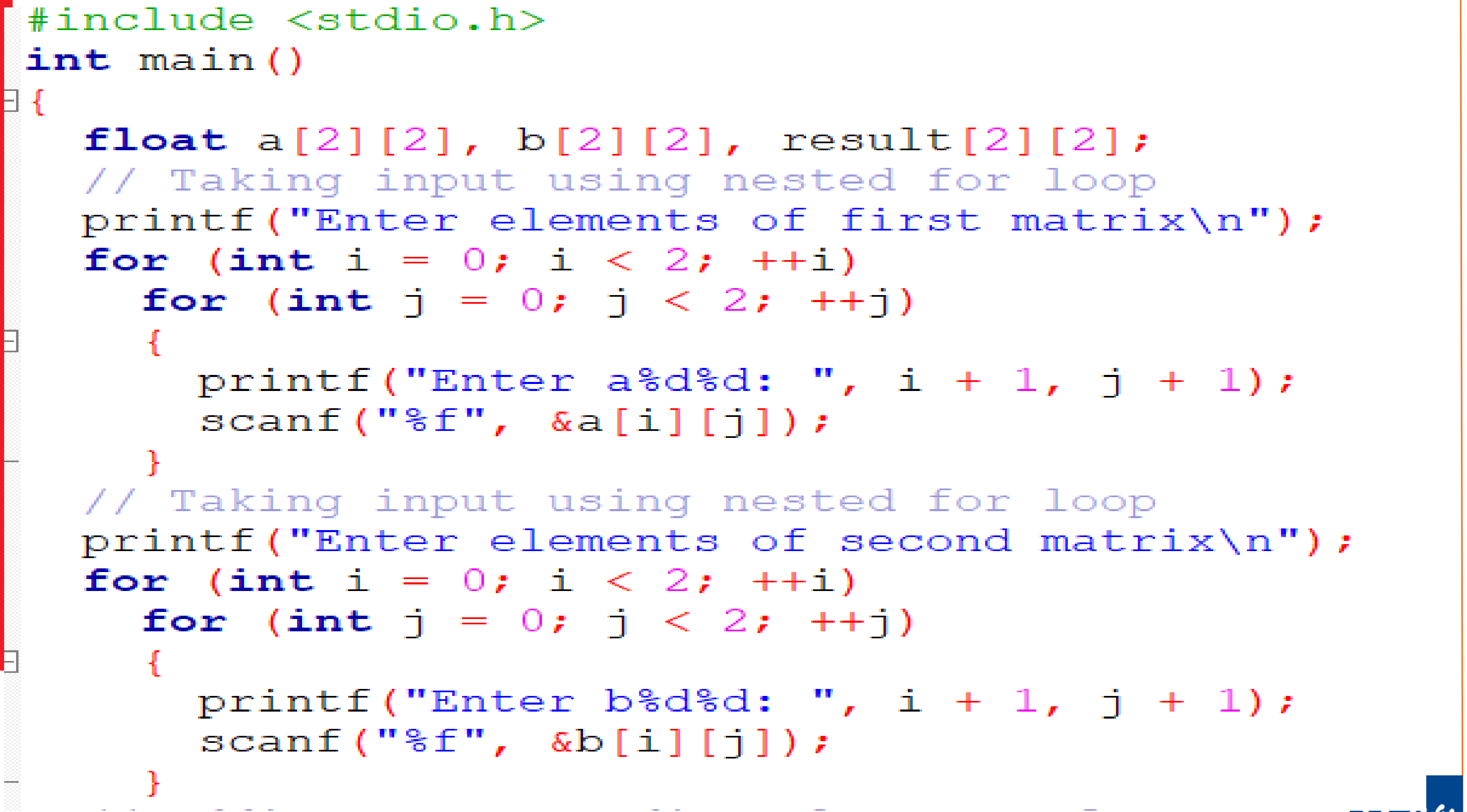
Example: 4



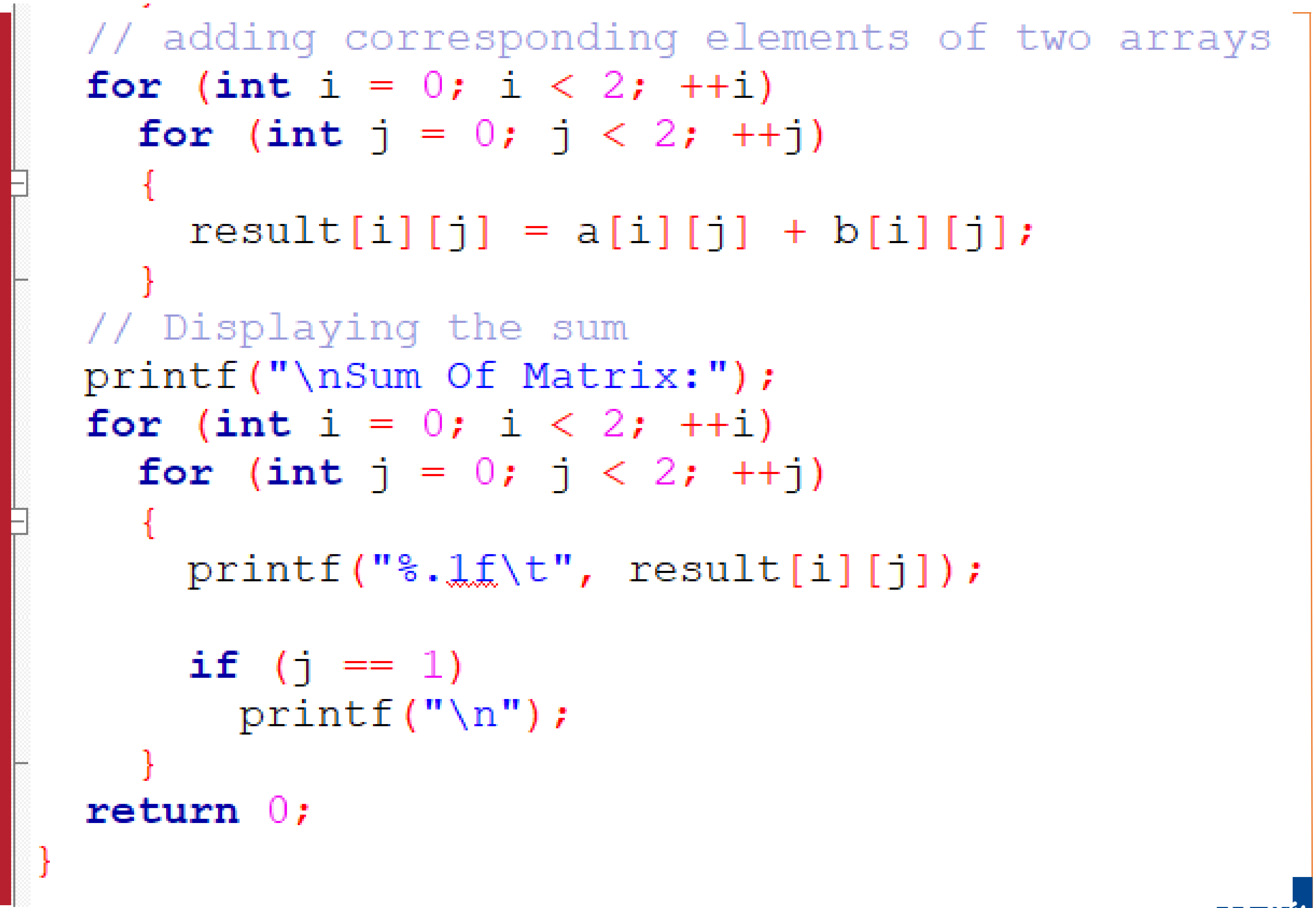
```
Enter the number of rows: 2
Enter the number of columns: 2
Enter the elements of the matrix:
Enter element [0][0]: 1
Enter element [0][1]: 1
Enter element [1][0]: 2
Enter element [1][1]: 2
```

```
#include <stdio.h>
int main() {
    int rows, cols, i, j;

    // Get the dimensions of the array from the user
    printf("Enter the number of rows: ");
    scanf("%d", &rows);
    printf("Enter the number of columns: ");
    scanf("%d", &cols);
 // Declare a 2D array
    int matrix[rows][cols];

    // Read elements into the 2D array
    printf("Enter the elements of the matrix:\n");
    for (i = 0; i < rows; i++) {
        for (j = 0; j < cols; j++) {
            printf("Enter element [%d][%d]: ", i, j);
            scanf("%d", &matrix[i][j]);
        }

    }
}
```

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



```c
#include <stdio.h>
int main()
{
   float a[2][2], b[2][2], result[2][2];
   // Taking input using nested for loop
   printf("Enter elements of first matrix\n");
   for (int i = 0; i < 2; ++i)
     for (int j = 0; j < 2; ++j)
     {
       printf("Enter a%d%d: ", i + 1, j + 1);
       scanf("%f", &a[i][j]);
     }
   // Taking input using nested for loop
   printf("Enter elements of second matrix\n");
   for (int i = 0; i < 2; ++i)
     for (int j = 0; j < 2; ++j)
     {
       printf("Enter b%d%d: ", i + 1, j + 1);
       scanf("%f", &b[i][j]);
     }
```



```
    // adding corresponding elements of two arrays
    for (int i = 0; i < 2; ++i)
      for (int j = 0; j < 2; ++j)
      {
        result[i][j] = a[i][j] + b[i][j];
      }
    // Displaying the sum
    printf("\nSum Of Matrix:");
    for (int i = 0; i < 2; ++i)
      for (int j = 0; j < 2; ++j)
      {
        printf("%.1f\t", result[i][j]);

        if (j == 1)
          printf("\n");
      }
    return 0;
  }
```

Strings in C

Group of characters can be stored in a character array.

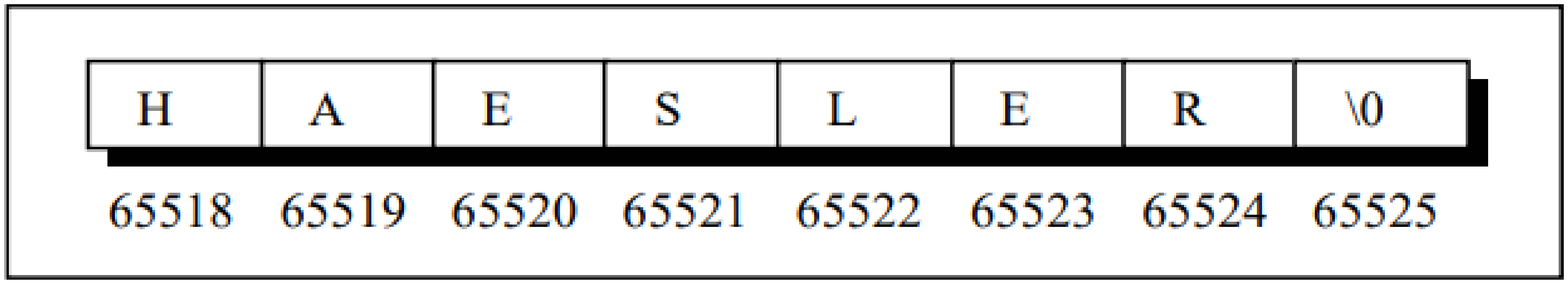
Character arrays are many a time also called strings.

A string constant is a **one-dimensional** array of characters terminated by a null (‘\0’).

For example

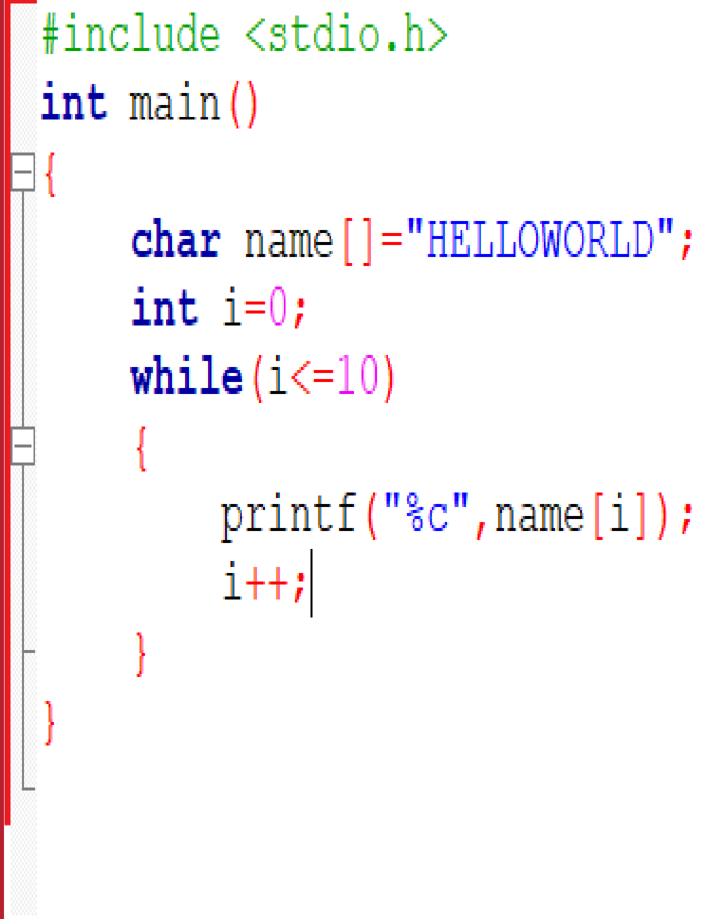
char name[ ] = { 'H', 'A', 'E', 'S', 'L', 'E', 'R', '**\0**' } ;

each character in the array occupies **one byte** of memory and the last character is always ‘\0’

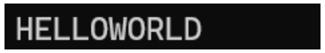


char name[ ] = "HAESLER" ; Note that, in this declaration ‘\0’ is not necessary. C inserts the null character automatically.

Program: To print string

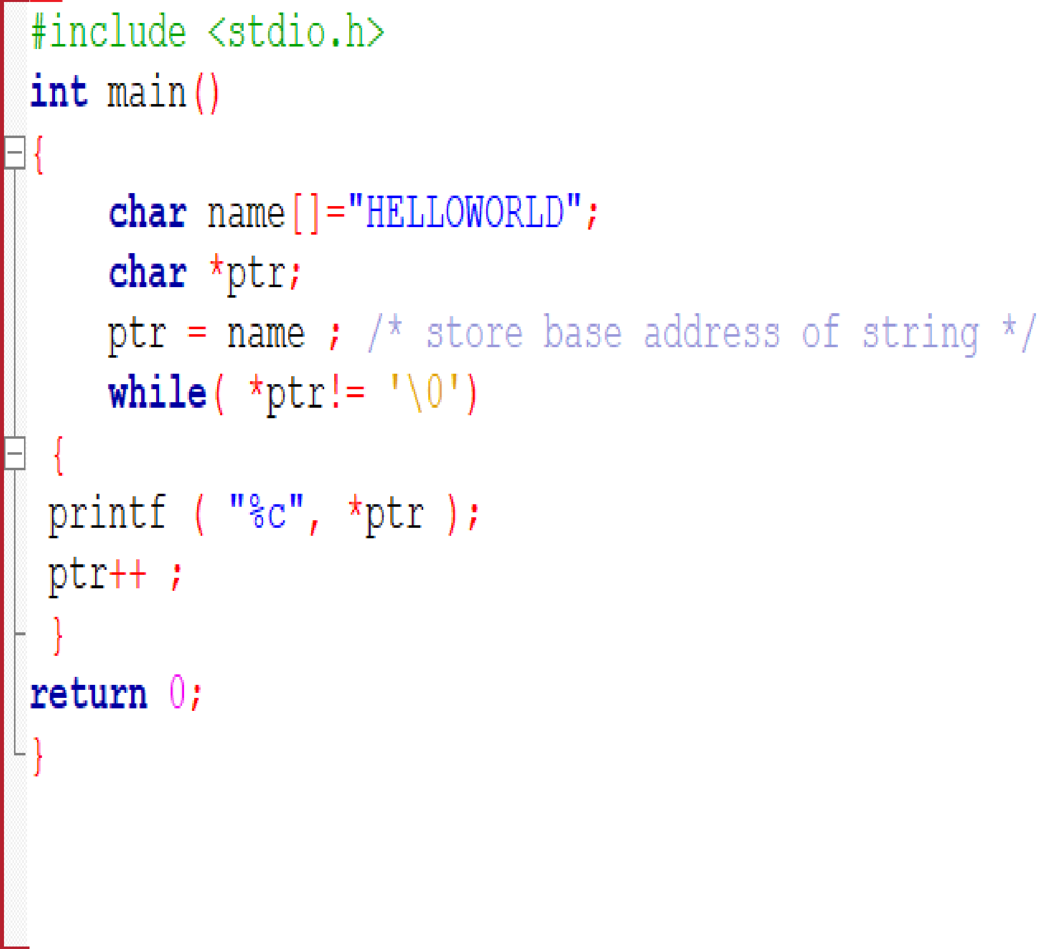


```c
#include <stdio.h>
int main()
{
    char name[]="HELLOWORLD";
    int i=0;
    while(i<=10)
    {
        printf("%c",name[i]);
        i++;
    }
}
```

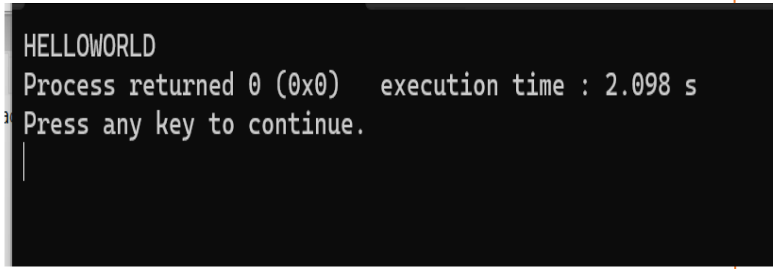


```
HELLOWORLD
```

Use of pointers

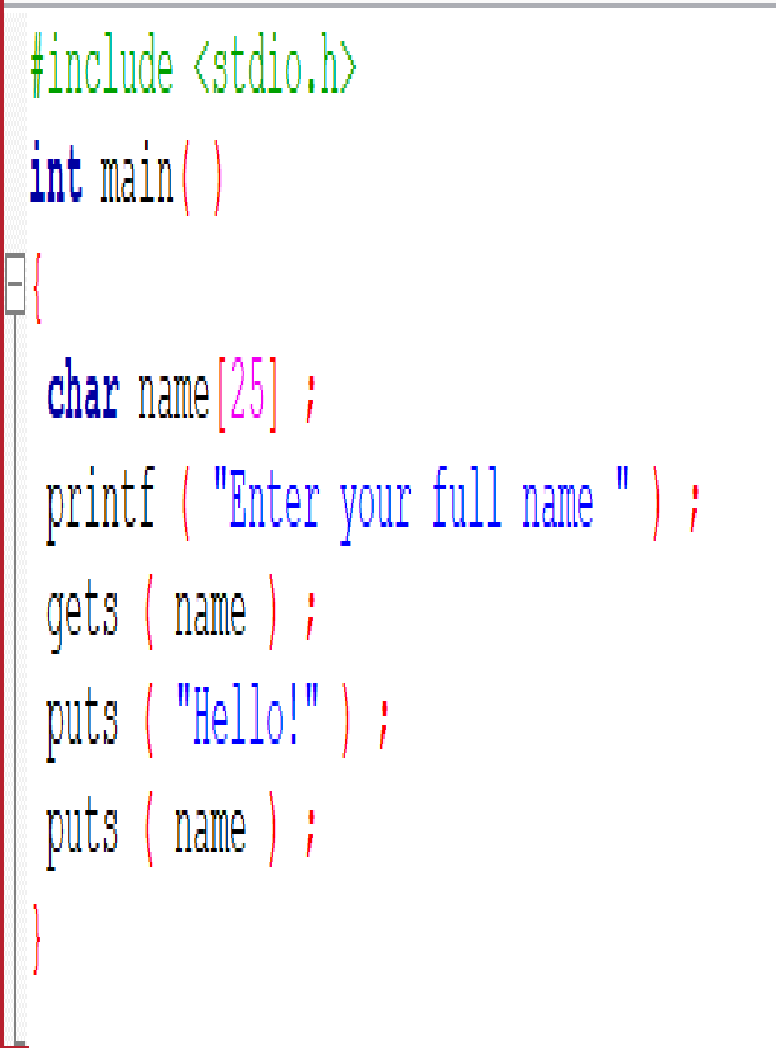


```
#include <stdio.h>
int main()
{
    char name[]="HELLOWORLD";
    char *ptr;
    ptr = name ; /* store base address of string */
    while( *ptr!= '\0')
 {
 printf ( "%c", *ptr );
 ptr++ ;
 }
return 0;
}
```

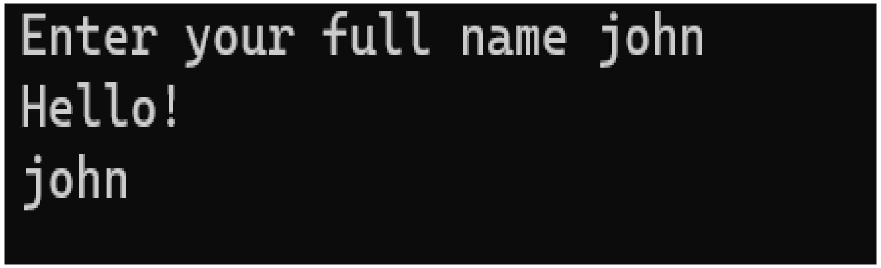


```
HELLOWORLD
Process returned 0 (0x0)   execution time : 2.098 s
Press any key to continue.
```

Example: get, put



```c
#include <stdio.h>
int main( )
{
 char name[25] ;
 printf ( "Enter your full name " ) ;
 gets ( name ) ;
 puts ( "Hello!" ) ;
 puts ( name ) ;
}
```



```
Enter your full name john
Hello!
john
```

String handling functions

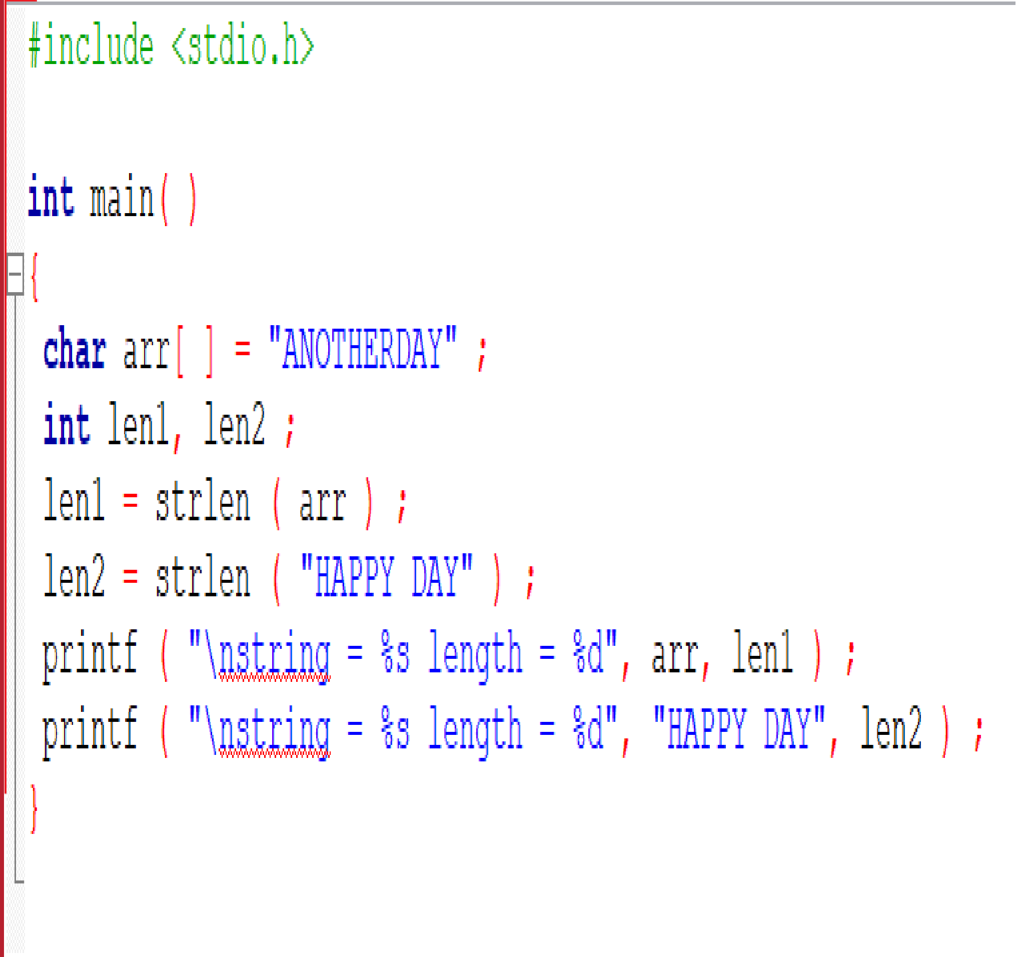
Function	Use
strlen	finds length of a string
strlwr	converts a string to lowercase
strupr	Converts a string to uppercase
strcat	Appends one string at the end of another
strncat	Appends first n characters of a string at the end of another
strcpy	Copies a string into another
strncpy	copies first n characters of one string into another
strcmp	Compares two strings
strncmp	compares first n characters of two strings
stricmp	Compares two strings without regard to case (identical to strcmpi)

String handling functions

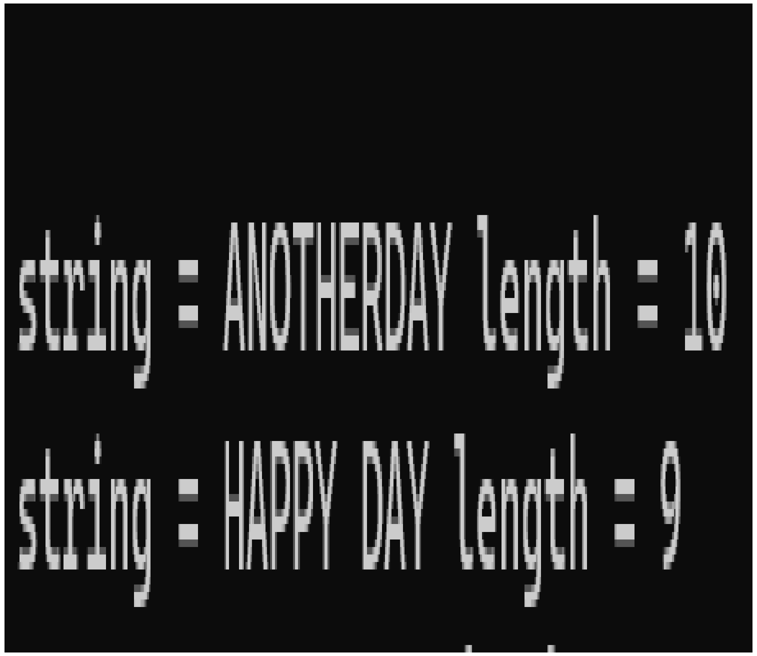
strnicmp	Compares first n characters of two strings without regard to case
strdup	Duplicates a string
strchr	finds first occurrence of a given character in a string
strrchr	finds last occurrence of a given character in a string
strstr	finds first occurrence of a given string in another string
strset	Sets all characters of string to a given character
strnset	Sets first n characters of a string to a given character
strrev	Reverses string

strlen( ) :

counts the number of characters present in a string.



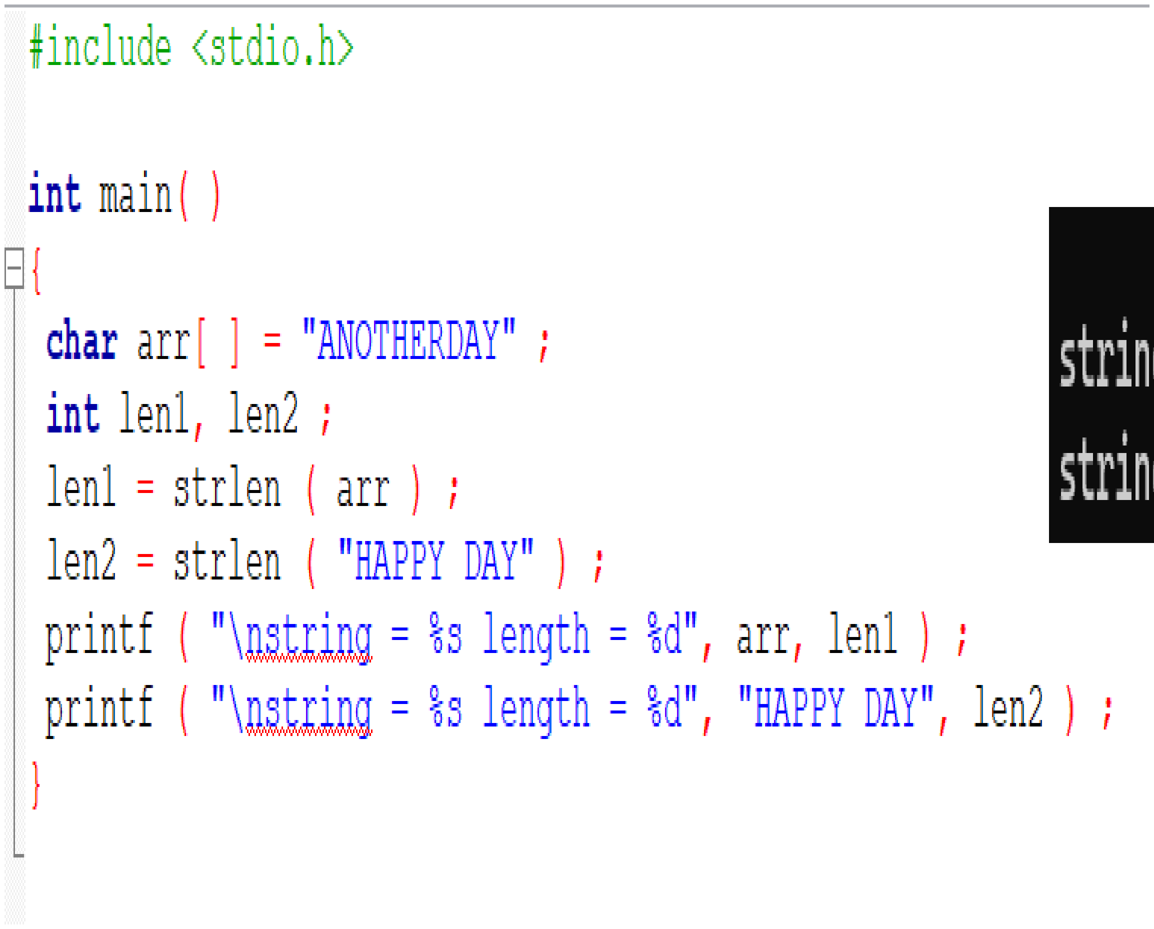
```
#include <stdio.h>

int main( )
{
 char arr[ ] = "ANOTHERDAY" ;
 int len1, len2 ;
 len1 = strlen ( arr ) ;
 len2 = strlen ( "HAPPY DAY" ) ;
 printf ( "\nstring = %s length = %d", arr, len1 ) ;
 printf ( "\nstring = %s length = %d", "HAPPY DAY", len2 ) ;
}
```



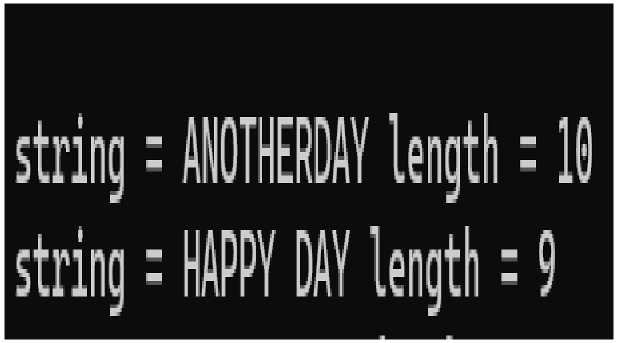
```
string = ANOTHERDAY length = 10
string = HAPPY DAY length = 9
```

strcpy( ) :

counts the number of characters present in a string.



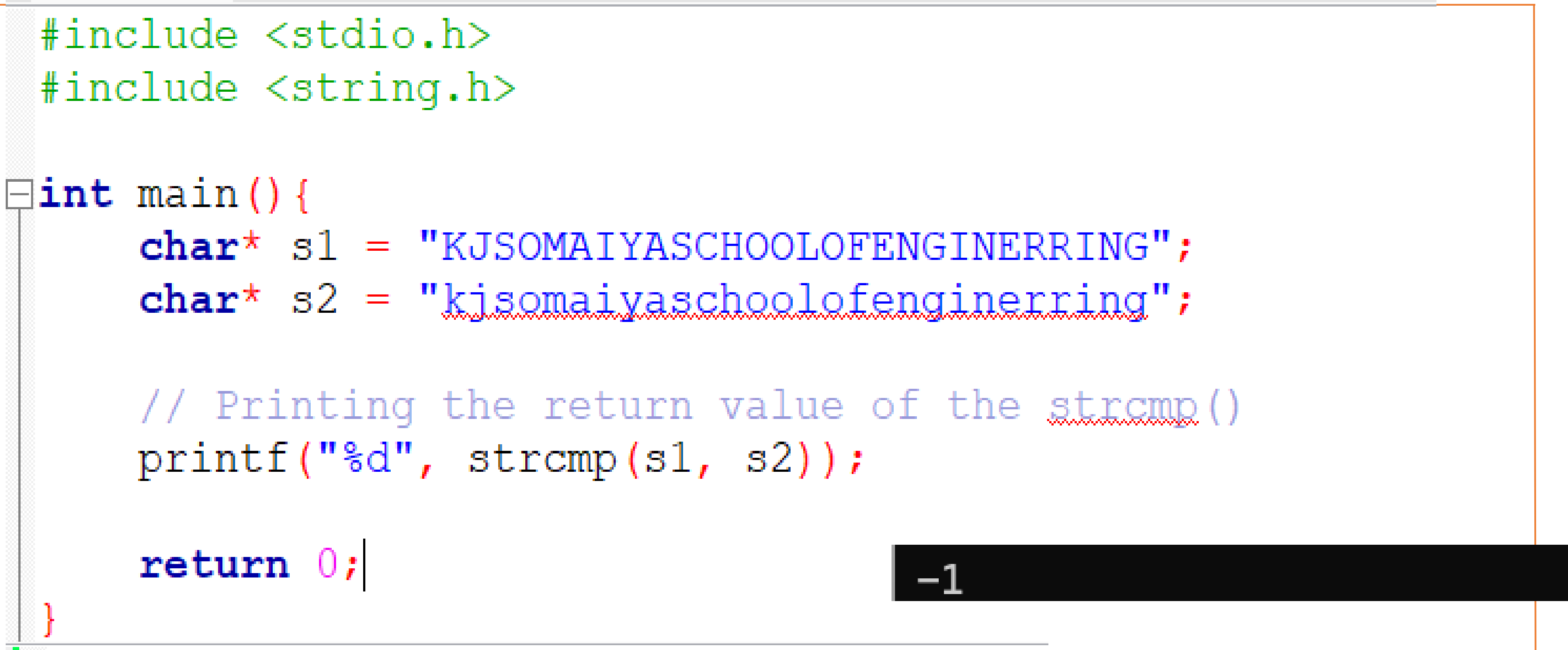
```
#include <stdio.h>

int main( )
{
 char arr[ ] = "ANOTHERDAY" ;
 int len1, len2 ;
 len1 = strlen ( arr ) ;
 len2 = strlen ( "HAPPY DAY" ) ;
 printf ( "\nstring = %s length = %d", arr, len1 ) ;
 printf ( "\nstring = %s length = %d", "HAPPY DAY", len2 ) ;
}
```



```
string = ANOTHERDAY length = 10
string = HAPPY DAY length = 9
```

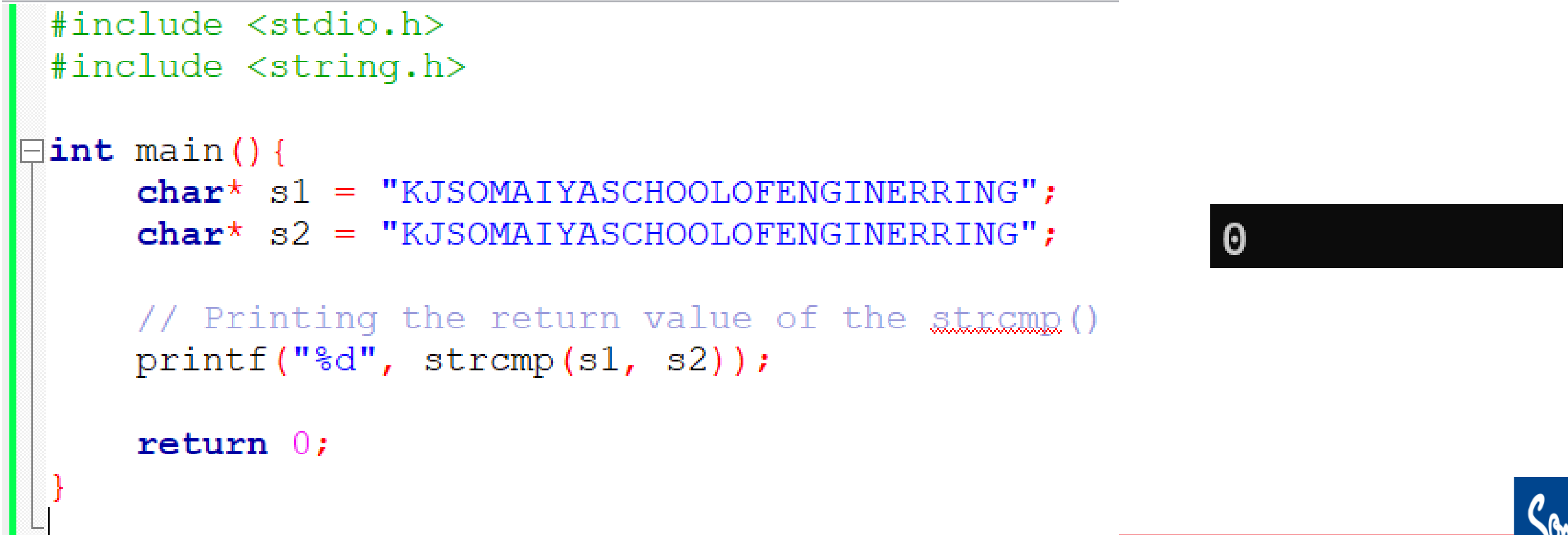
strcmp( )

- This is a function which compares two strings to find out whether they are same or different.
- used to compare two null-terminated strings (also known as C-strings) lexicographically.
- It performs a case-sensitive comparison of the strings character by character based on their ASCII values.
- ASCII values are different for lower case and upper case
- **Syntax**
- int strcmp(const char *str1, const char *str2);



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

int main(){
    char* s1 = "KJSOMAIYASCHOOLOFENGINERRING";
    char* s2 = "kjsomaiyaschoolofenginerring";

    // Printing the return value of the strcmp()
    printf("%d", strcmp(s1, s2));

    return 0;
}
```

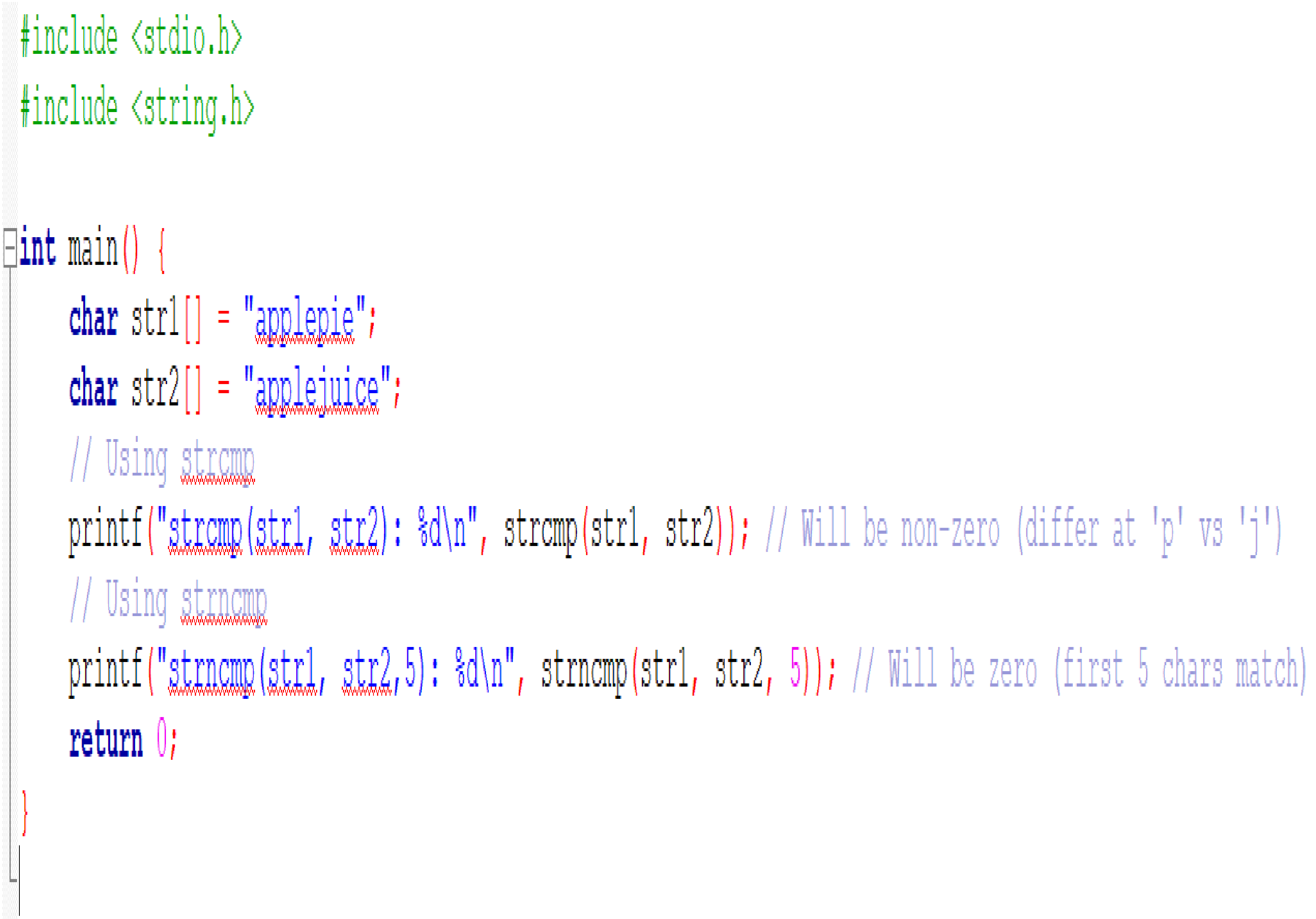
```
-1
```



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

int main(){
    char* s1 = "KJSOMAIYASCHOOLOFENGINERRING";
    char* s2 = "KJSOMAIYASCHOOLOFENGINERRING";

    // Printing the return value of the strcmp()
    printf("%d", strcmp(s1, s2));

    return 0;
}
```

```
0
```

- Strncmp

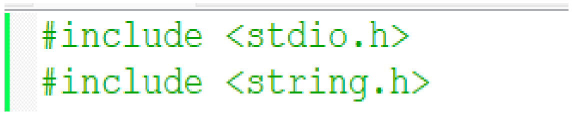


```
#include <stdio.h>
#include <string.h>

int main() {
    char str1[] = "applepie";
    char str2[] = "applejuice";
    // Using strcmp
    printf("strcmp(str1, str2): %d\n", strcmp(str1, str2)); // Will be non-zero (differ at 'p' vs 'j')
    // Using strncmp
    printf("strncmp(str1, str2,5): %d\n", strncmp(str1, str2, 5)); // Will be zero (first 5 chars match)
    return 0;
}
```

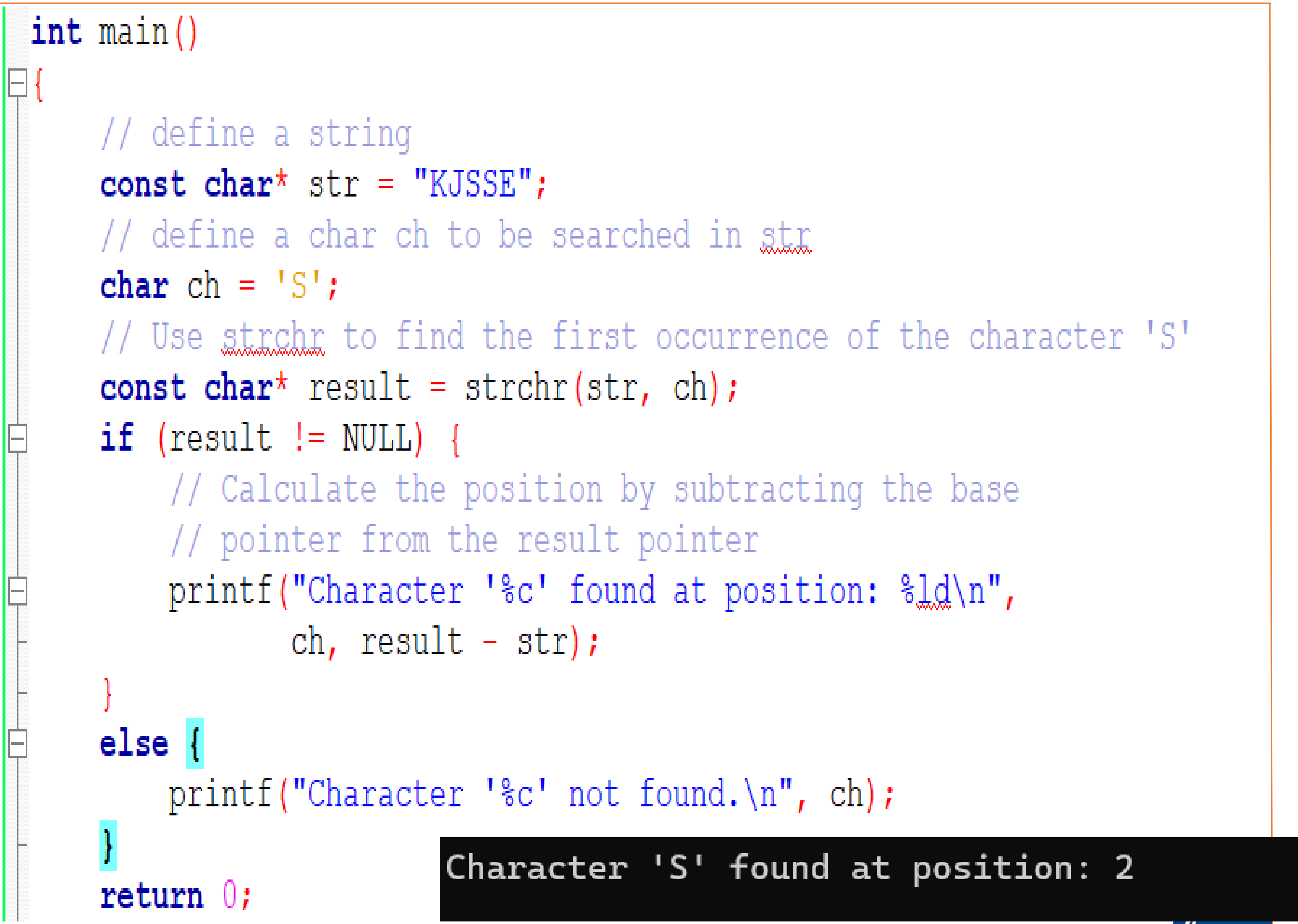
- **Stricmp**
- It compares *string1* and *string2* without sensitivity to case. All alphabetic characters in the two arguments *string1* and *string2* are converted to lowercase before the comparison.
- The function operates on null-ended strings. The string arguments to the function are expected to contain a null character (\0) marking the end of the string.

strchr

- It is used to find the first occurrence of a character in a string.
- It checks whether the given character is present in the given string.
- If the character is found, it returns the pointer to its first occurrence otherwise, it returns a null pointer, indicating the character is not found in the string.
- **Syntax of strchr() in C**
- char *strchr(**const** char *str, int ch);
- Make sure header files are added



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

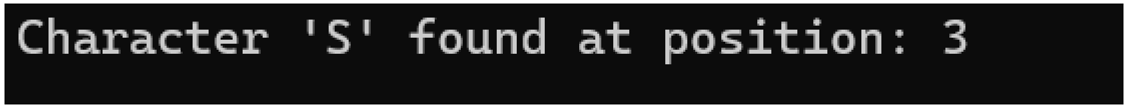


```c
int main()
{
    // define a string
    const char* str = "KJSSE";
    // define a char ch to be searched in str
    char ch = 'S';
    // Use strchr to find the first occurrence of the character 'S'
    const char* result = strchr(str, ch);
    if (result != NULL) {
        // Calculate the position by subtracting the base
        // pointer from the result pointer
        printf("Character '%c' found at position: %ld\n",
                  ch, result - str);
    }
    else {
        printf("Character '%c' not found.\n", ch);
    }
    return 0;
```

```
Character 'S' found at position: 2
```

- strrchr
- The **strrchr()** function in C locates the last occurrence of a character in a string and returns a pointer to it.
- **Syntax :**
- char* strrchr(char* str, int chr);

```c
int main()
{
    // define a string
    const char* str = "KJSSE";
    // define a char ch to be searched in str
    char ch = 'S';
    // Use strchr to find the first occurrence of the character 'S'
    const char* result = strrchr(str, ch);
    if (result != NULL) {
        // Calculate the position by subtracting the base
        // pointer from the result pointer
        printf("Character '%c' found at position: %ld\n",
                  ch, result - str);
    }
    else {
        printf("Character '%c' not found.\n", ch);
    }
    return 0;
```



```
Character 'S' found at position: 3
```

strstr

- This function takes two strings **s1** and **s2** as arguments and finds the first occurrence of the string **s2** in the string **s1**.

Syntax

- char ***strstr** (const char **s1*, const char **s2*);
- **s1**: This is the main string to be examined.
- **s2**: This is the sub-string to be searched in string.

- **Return Value**
- This function returns **a pointer** point to the first character of the found *s2* in *s1* otherwise a null pointer if *s2* is not present in *s1*.
- If s2 points to an empty string, s1 is returned.

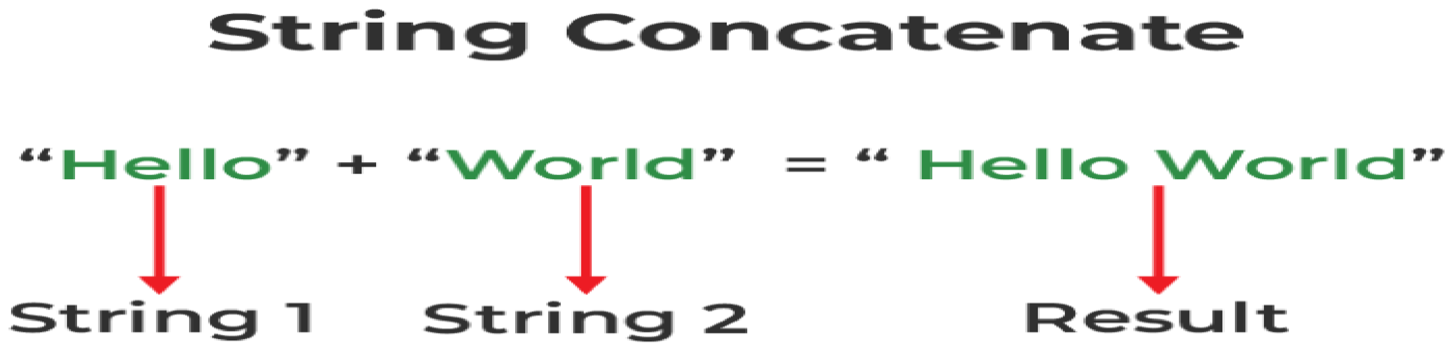
```
#include <string.h>
int main()
{
    // Take any two strings
    char s1[] = "Another beautiful day";
    char s2[] = "day";
    char* p;
    // Find first occurrence of s2 in s1
    p = strstr(s1, s2);
    // Prints the result
    if (p) {
        printf("String found\n");
    }
    else
        printf("String not found\n");
    return 0;
}
```

```
String found
```

Strcat

- The strcat() function in C is used to concatenate (join) two strings. It appends the source string to the end of the destination string, modifying the destination string. This function is declared in the string.h header file.
- **Syntax:**

char *strcat(char *destination, const char *source);

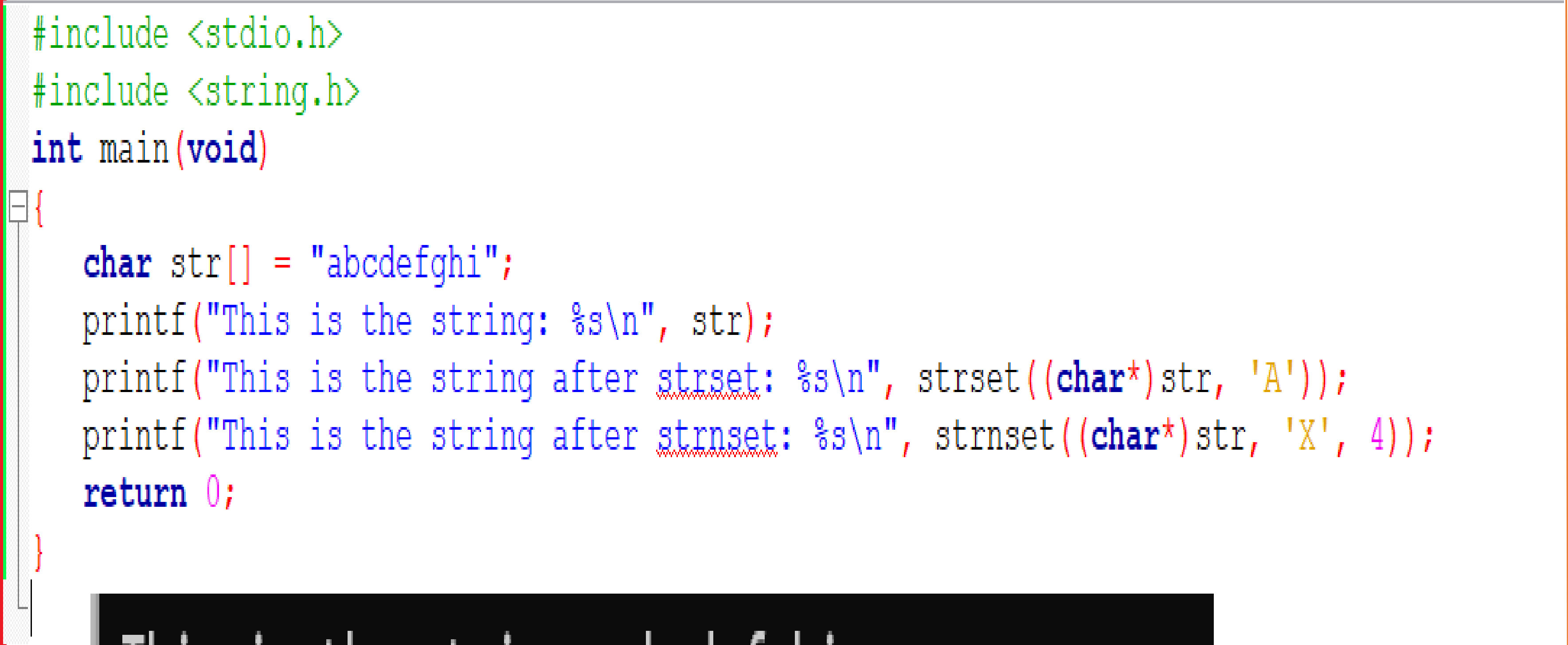


Strset and strnset

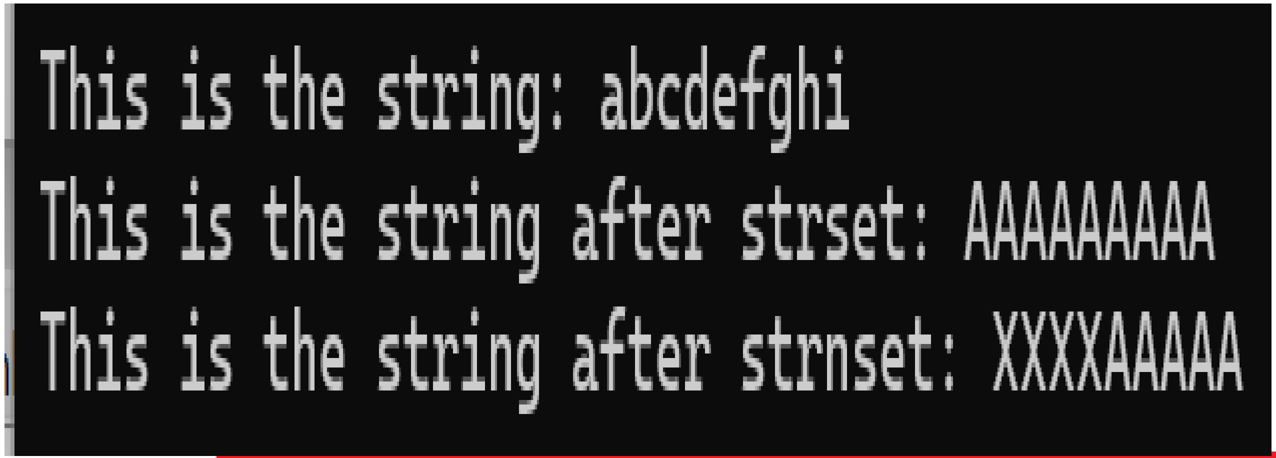
- Set the characters in given string

- **Syntax**
- #include <string.h>
- char *strnset(char *string, int c, size_t n);
- char *strset(char *string, int c);

- **Example**

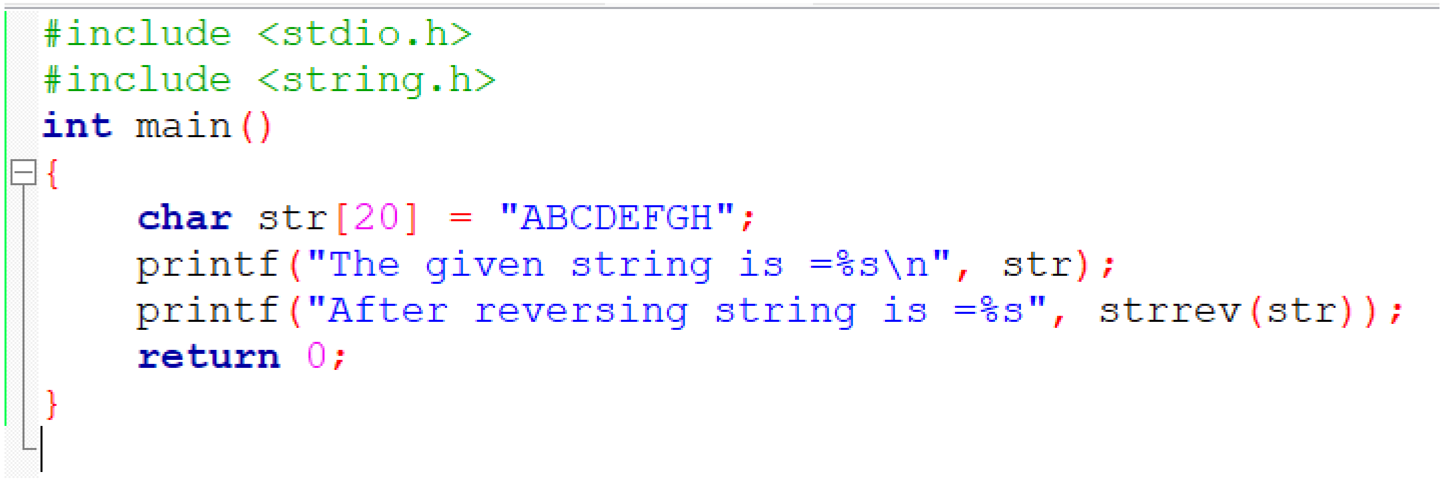


```
#include <stdio.h>
#include <string.h>
int main(void)
{
    char str[] = "abcdefghi";
    printf("This is the string: %s\n", str);
    printf("This is the string after strset: %s\n", strset((char*)str, 'A'));
    printf("This is the string after strnset: %s\n", strnset((char*)str, 'X', 4));
    return 0;
}
```

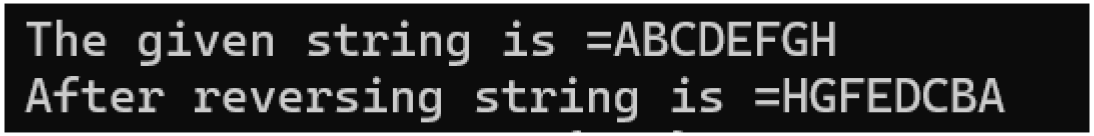


```
This is the string: abcdefghi
This is the string after strset: AAAAAAAAA
This is the string after strnset: XXXXAAAAA
```

- **strrev**
- The **strrev()** function is a built-in function in C and is defined in **string.h** header file.
- The strrev() function is used to reverse the given string.



```
#include <stdio.h>
#include <string.h>
int main()
{
    char str[20] = "ABCDEFGH";
    printf("The given string is =%s\n", str);
    printf("After reversing string is =%s", strrev(str));
    return 0;
}
```



```
The given string is =ABCDEFGH
After reversing string is =HGFEDCBA
```

Exercise

Consider a string named mystring1="HELLO"

mystring2="EVERYONE"

Find the length of each string

Copy the string

Compare the strings

Concatenate the strings

Reverse the strings