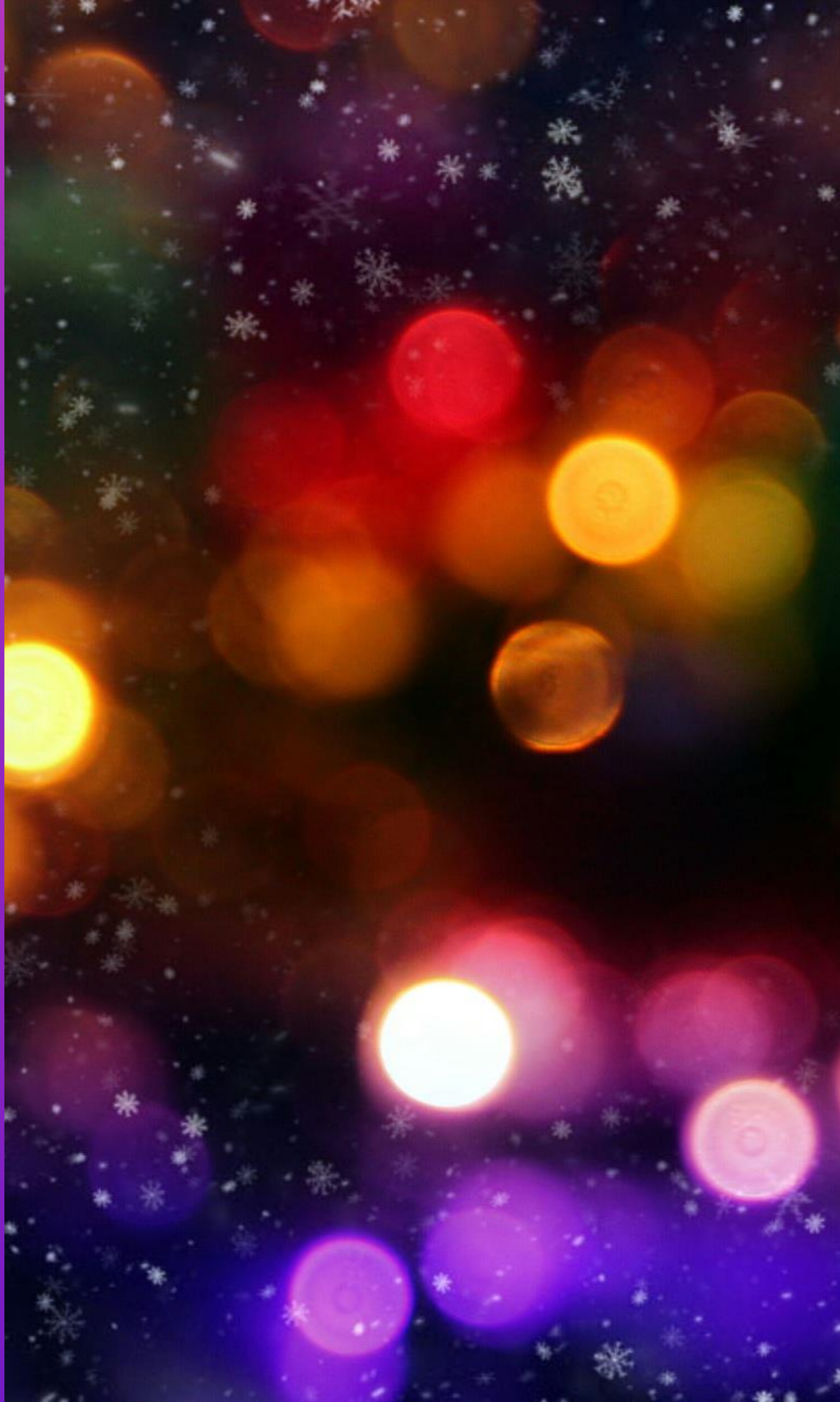
MNNIT COMPUTER CODING CLUB

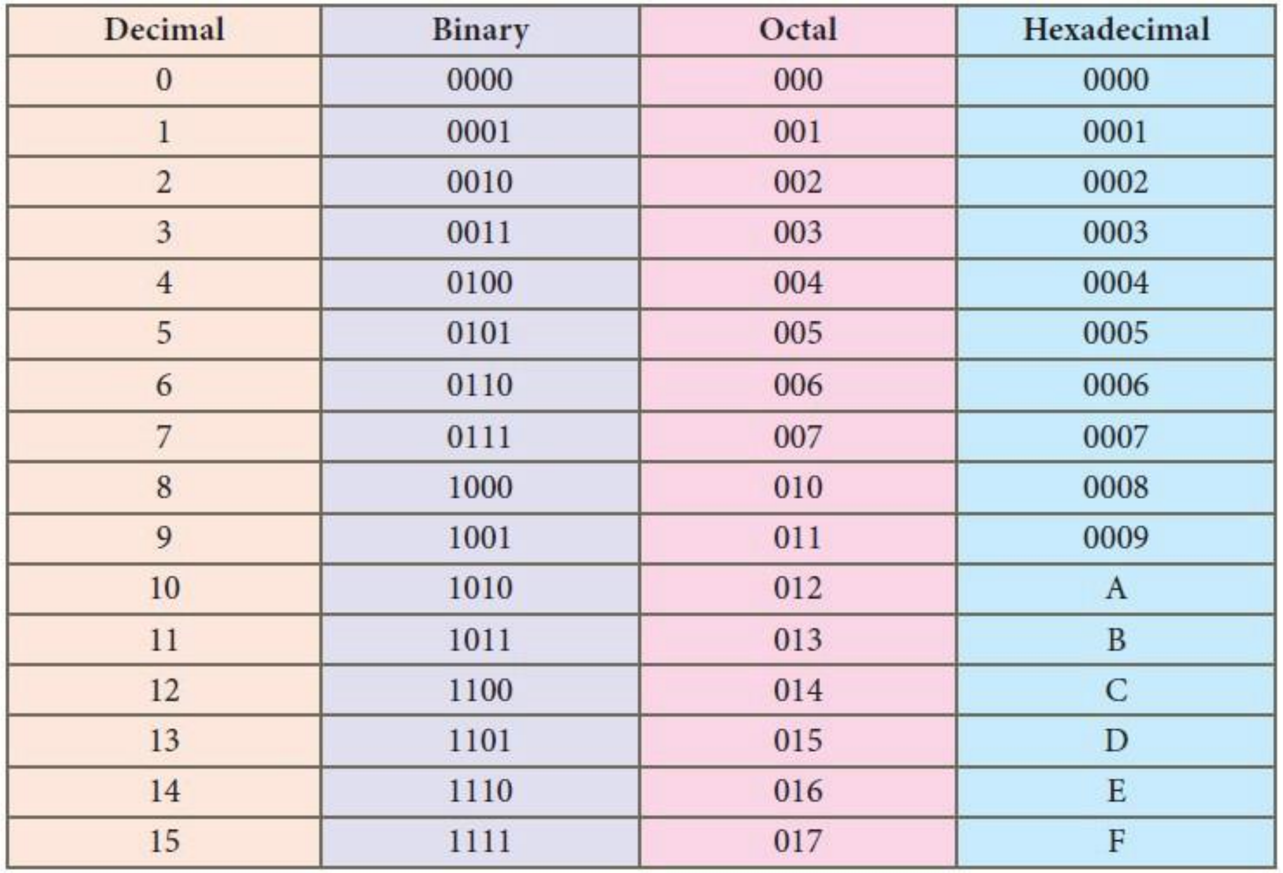
CLASS-8

BASICS OF C



NUMBER SYSTEMS

- Binary
  - Base 2
  - Digits used : 0, 1
- Decimal
  - Base 10
  - Digits used : 0 to 9
- Octal
  - Base 8
  - Digits used : 0 to 7
- Hexadecimal
  - Base 16
  - Digits used: 0 to 9, Letters used : A- F
- Base n



Decimal	Binary	Octal	Hexadecimal
0	0000	000	0000
1	0001	001	0001
2	0010	002	0002
3	0011	003	0003
4	0100	004	0004
5	0101	005	0005
6	0110	006	0006
7	0111	007	0007
8	1000	010	0008
9	1001	011	0009
10	1010	012	A
11	1011	013	B
12	1100	014	C
13	1101	015	D
14	1110	016	E
15	1111	017	F

DECIMAL TO OTHER BASE SYSTEM

- **Step 1** – Divide the decimal number to be converted by the value of the new base.
- **Step 2** – Get the remainder from Step 1 as the rightmost digit (least significant digit) of the new base number.
- **Step 3** – Divide the quotient of the previous divide by the new base.
- **Step 4** – Record the remainder from Step 3 as the next digit (to the left) of the new base number.
- Repeat Steps 3 and 4, getting remainders from right to left, until the quotient becomes zero in Step 3.
- The last remainder thus obtained will be the Most Significant Digit (MSD) of the new base number.

OTHER BASE SYSTEM TO DECIMAL SYSTEM

- **Step 1** – Determine the column (positional) value of each digit (this depends on the position of the digit and the base of the number system).
- **Step 2** – Multiply the obtained column values (in Step 1) by the digits in the corresponding columns.
- **Step 3** – Sum the products calculated in Step 2. The total is the equivalent value in decimal.

BITWISE OPERATORS

- Bitwise AND (&)
- Bitwise OR (|)
- Bitwise XOR (^)
- One's Complement (~)
- Bitwise Left Shift (<<)
- Bitwise Right Shift (>>)

BITWISE OPERATORS

- Bitwise AND (&)

Bit of operand1	Bit of operand2	Resulting Bit
0	0	0
0	1	0
1	0	0
1	1	1

```
a                 0000 1010          = 10 (Decimal)
b                 0001 1100          = 28 (Decimal)
_____________________________________________________
a & b             0000 1000          =  8 (Decimal)
```

BITWISE OPERATORS

- Bitwise OR (|)

Bit of operand1	Bit of operand2	Resulting Bit
0	0	0
0	1	1
1	0	1
1	1	1

```
a          0000 1010     = 10 (Decimal)
b          0001 1100     = 28 (Decimal)
--------------------------------------------
a | b      0001 1110     = 30 (Decimal)
```

BITWISE OPERATORS

- Bitwise XOR (^)

Bit of operand1	Bit of operand2	Resulting Bit
0	0	0
0	1	1
1	0	1
1	1	0

```
a                       0000 1010          = 10 (Decimal)
b                       0001 1100          = 28 (Decimal)
_____________________________________________________________
a ^ b                   0001 0110          = 22 (Decimal)
```

BITWISE OPERATORS

- One's Complement (~)

Bit of operand1	Resulting Bit
0	1
1	0

```
a               0000 1010        = 10 (Decimal)
__________________________________________________
~a              1111 0101        = 245 (Decimal)
```

*For 8-bit number, remember integer is either 32 bits or 64 bits depending on the OS

BITWISE OPERATORS

- Bitwise Left Shift (<<)

a	0001 1010 1000 0001	a << 5	00011 0101 0000 001**0 0000**
a	6,785 (Decimal)		

00000 New bits added
00011 Lost bits
0101 0000 001**0 0000** = 20,512 (Decimal)

BITWISE OPERATORS

- Bitwise Right Shift (>>)

a 0001 1010 1000 0001

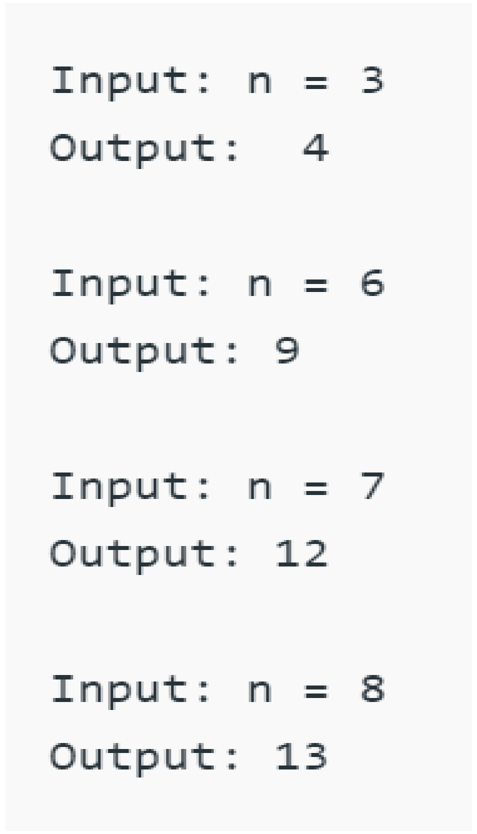
a >> 5 **0000 0**000 1101 0100 00001

00000 New bits added

00001 Lost bits

0000 0000 1101 0100 = 212 (Decimal)

Given **N**, if we write all numbers from **1** to **N** (both inclusive) in binary what is the count of 1s I have written.
For example, if N=3,
I will write down:
1
10
11
Therefore, a total of 4 ones.



```
Input: n = 3
Output:  4

Input: n = 6
Output: 9

Input: n = 7
Output: 12

Input: n = 8
Output: 13
```

Few problems you can try yourself :

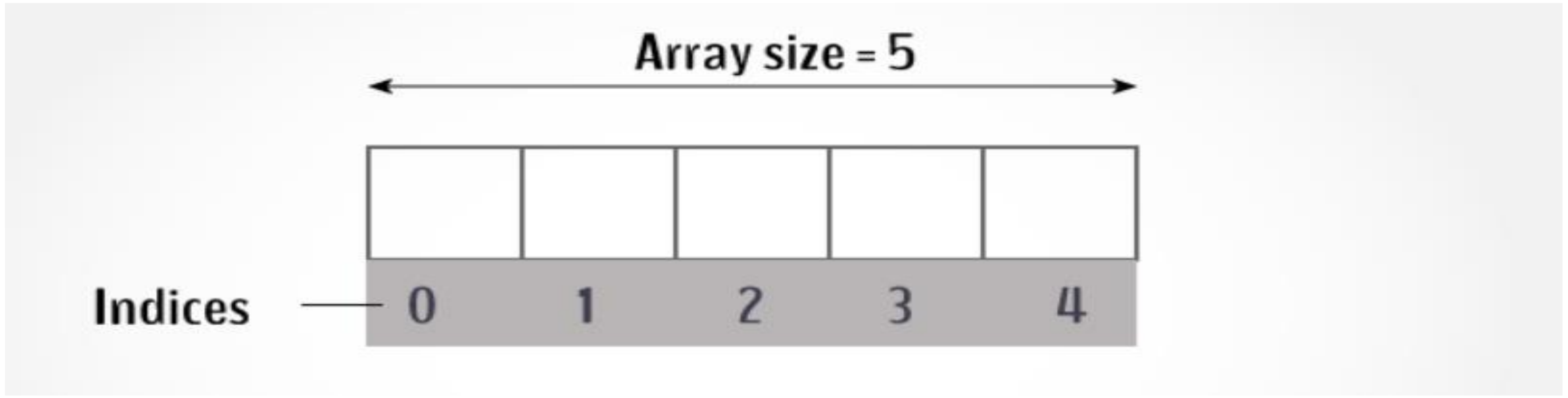
https://www.hackerrank.com/challenges/sum-vs-xor/problem

https://codeforces.com/problemset/problem/1208/A

https://codeforces.com/problemset/problem/1421/A

1D ARRAY

- An array in C is a collection of homogenous items stored at contiguous memory locations.
- The elements of the array share the same variable name but each element has its own unique index number.
- An array can be of any type. For example: `int`, `float`, `char` etc. If an array is of type `int` then it's elements must be of type `int` only.
- We can use normal variables (v1, v2, v3, ..) when we have a small number of objects, but if we want to store a large number of instances, it becomes difficult to manage them with normal variables. The idea of an array is to represent many instances in one variable.



DECLARING 1D ARRAY

dataType arrayName[arraySize];

For example, `float mark[5];`

- Here, we declared an array, mark, of floating-point type. And its size is 5. Meaning, it can hold 5 floating-point values.
- It's important to note that the size and type of an array cannot be changed once it is declared.

INITIALIZING 1D ARRAY

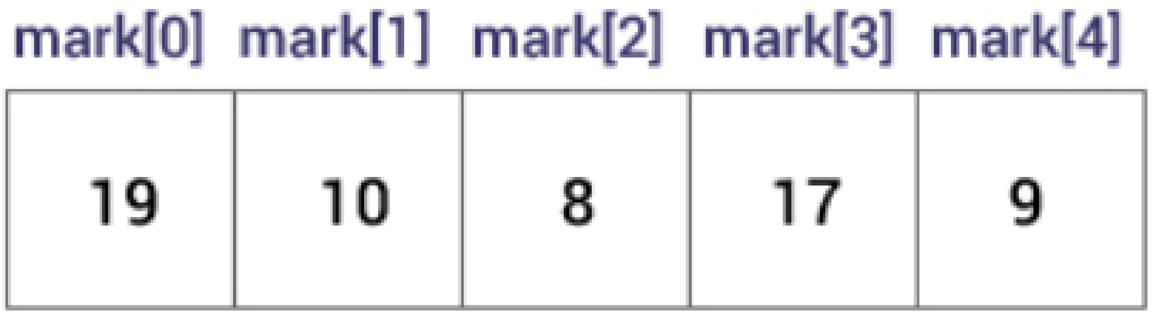
It is possible to initialize an array during declaration. For example,

```
int mark[5] = {19, 10, 8, 17, 9};
```

An array can also be initialized like

```
int mark[] = {19, 10, 8, 17, 9};
```

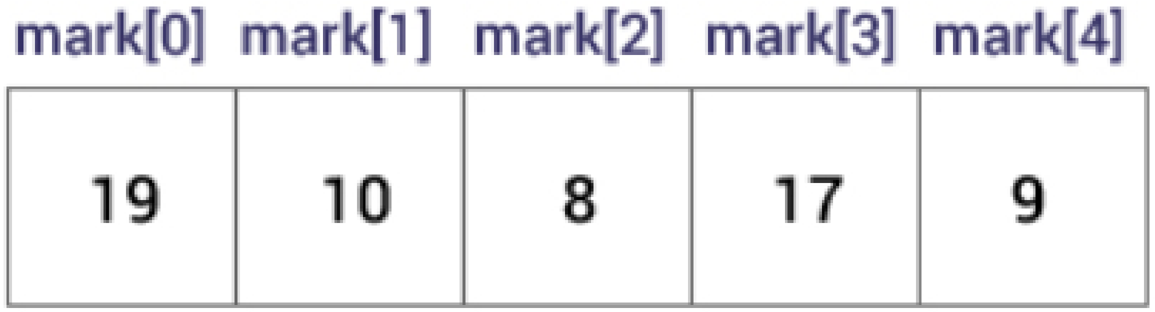
Here, we haven't specified the size. However, the compiler knows its size is 5 as we are initializing it with 5 elements.



ACCESSING 1D ARRAY ELEMENTS

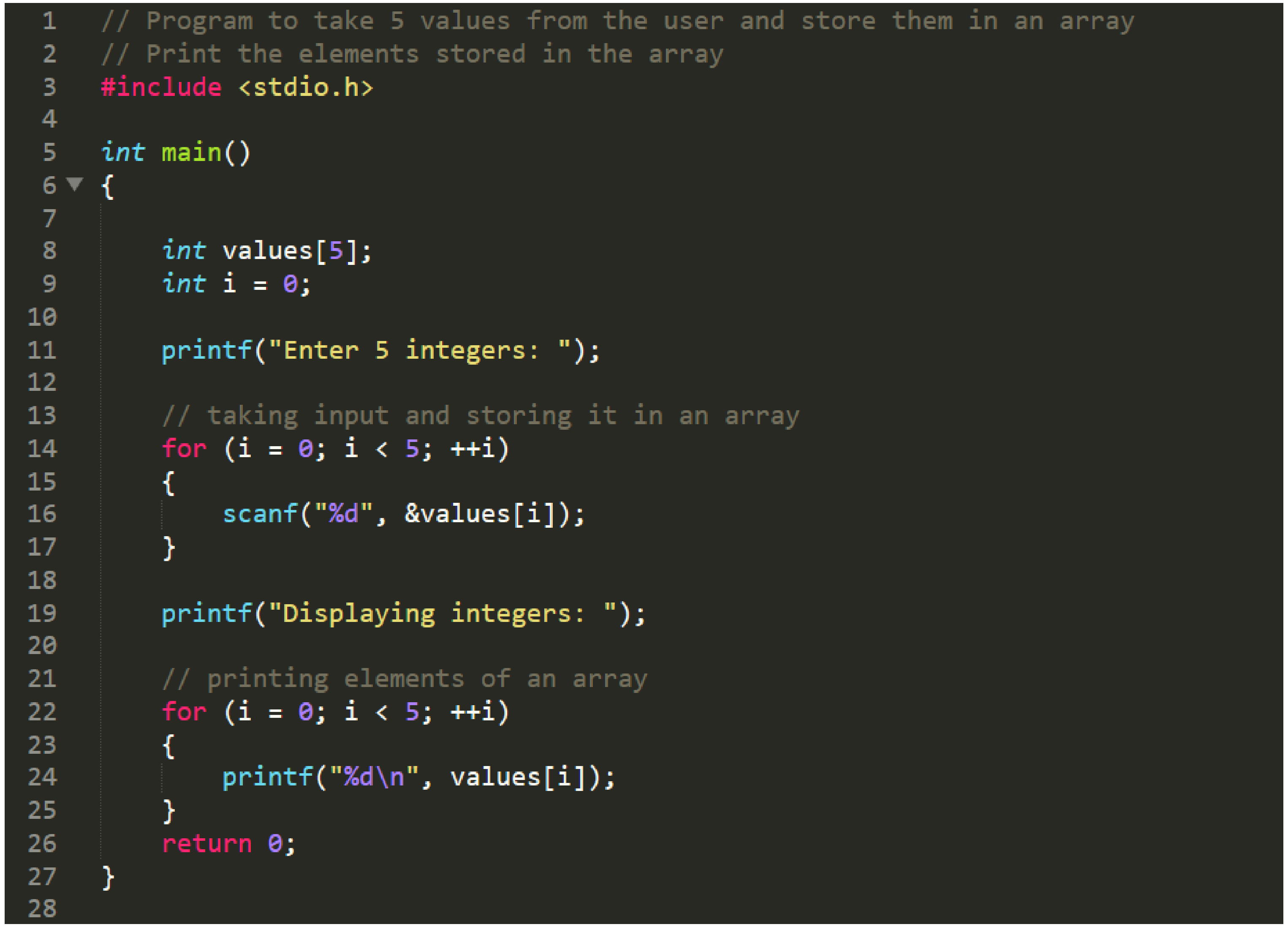
- Elements of array can be accessed by indices.
- Array index starts from 0 and goes till size of array minus 1.

  Considering the array mark declared in previous slide. The first element is mark[0], the second element is mark[1] and so on.

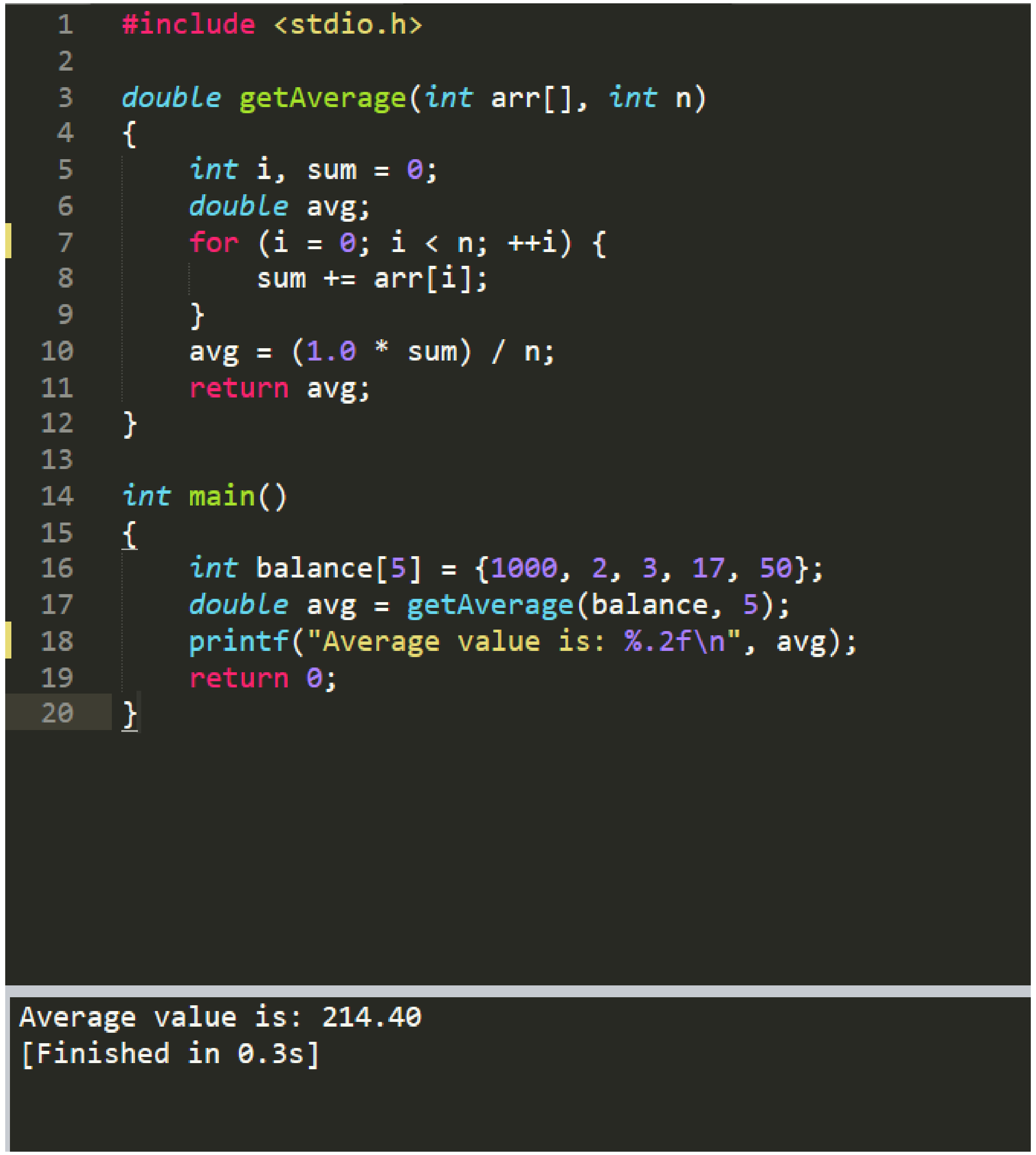


```
mark[0] is equal to 19
mark[1] is equal to 10
mark[2] is equal to 8
mark[3] is equal to 17
mark[4] is equal to 9
```

1D ARRAY SAMPLE PROGRAM



```c
// Program to take 5 values from the user and store them in an array
// Print the elements stored in the array
#include <stdio.h>

int main()
{

    int values[5];
    int i = 0;

    printf("Enter 5 integers: ");

    // taking input and storing it in an array
    for (i = 0; i < 5; ++i)
    {
        scanf("%d", &values[i]);
    }

    printf("Displaying integers: ");

    // printing elements of an array
    for (i = 0; i < 5; ++i)
    {
        printf("%d\n", values[i]);
    }
    return 0;
}

```



```c
#include <stdio.h>

double getAverage(int arr[], int n)
{
    int i, sum = 0;
    double avg;
    for (i = 0; i < n; ++i) {
        sum += arr[i];
    }
    avg = (1.0 * sum) / n;
    return avg;
}

int main()
{
    int balance[5] = {1000, 2, 3, 17, 50};
    double avg = getAverage(balance, 5);
    printf("Average value is: %.2f\n", avg);
    return 0;
}
```

```
Average value is: 214.40
[Finished in 0.3s]
```

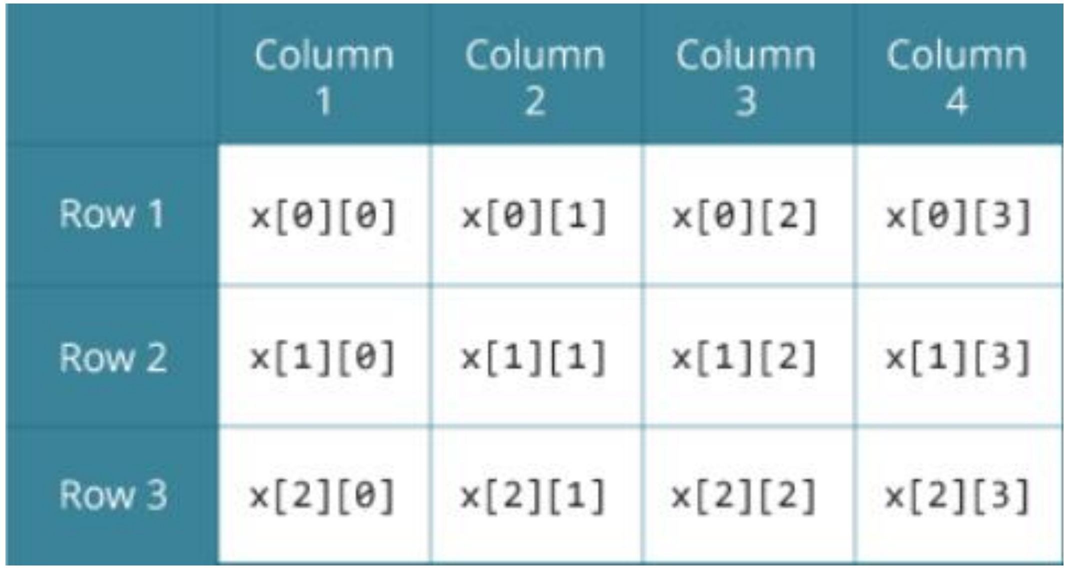
PASSING 1D ARRAY TO FUNCTION

- We can pass whole array as an actual argument to a function. The corresponding formal argument should be declared as an array variable of the same data type.
- On the left side is a program to compute the average of elements of an array.

2D ARRAY

- An array of arrays is known as 2D array.
- Just like 1D array, 2D array also store homogenous values (i.e. same data type values)

Here, **x** is a two-dimensional (2d) array. The array can hold 12 elements. This 2D array can be visualized as a table with 3 rows and each row has 4 columns.



	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]

DECLARING 2D ARRAY

The syntax to declare the 2D array is given below.

```
dataType arrayName[rows][columns];
```

Consider the following example.

```
int arr[4][3];
```

Here, 4 is the number of rows, and 3 is the number of columns.

INITIALIZING 2D ARRAY

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

2D array can also be initialized like

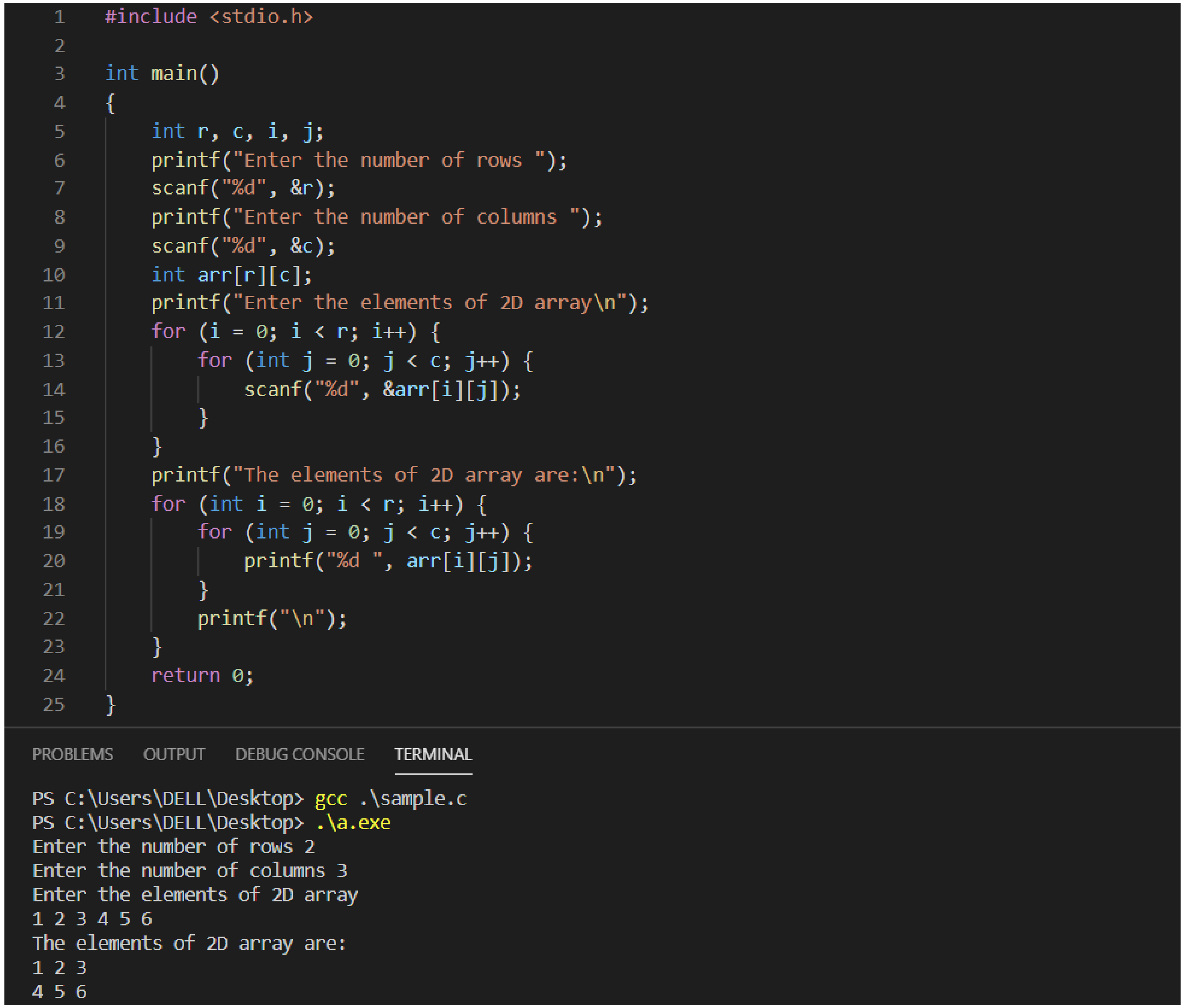
```
int arr[2][3] = {1, 3, 0, -1, 5, 9};
```

Note: Although both the above declarations are valid, we recommend you to use the first method as it is more readable.

We already know, when we initialize a 1D array during declaration, we need not to specify the size of it. However that's not the case with 2D array, you must always specify the second dimension even if you are specifying elements during the declaration.

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

2D ARRAY SAMPLE PROGRAM

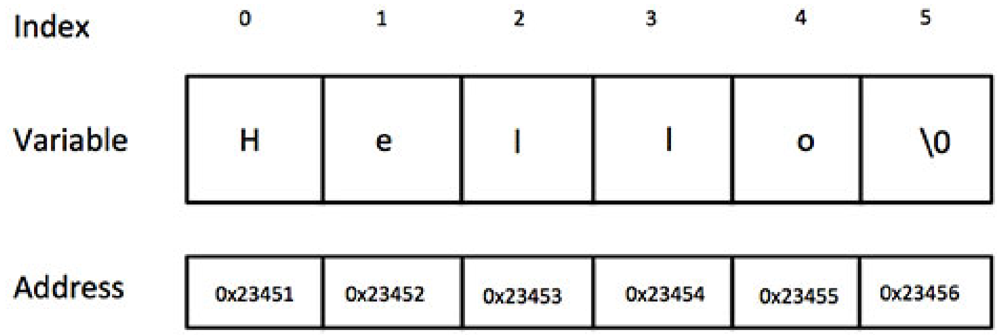


```c
#include <stdio.h>

int main()
{
    int r, c, i, j;
    printf("Enter the number of rows ");
    scanf("%d", &r);
    printf("Enter the number of columns ");
    scanf("%d", &c);
    int arr[r][c];
    printf("Enter the elements of 2D array\n");
    for (i = 0; i < r; i++) {
        for (int j = 0; j < c; j++) {
            scanf("%d", &arr[i][j]);
        }
    }
    printf("The elements of 2D array are:\n");
    for (int i = 0; i < r; i++) {
        for (int j = 0; j < c; j++) {
            printf("%d ", arr[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

```
PS C:\Users\DELL\Desktop> gcc .\sample.c
PS C:\Users\DELL\Desktop> .\a.exe
Enter the number of rows 2
Enter the number of columns 3
Enter the elements of 2D array
1 2 3 4 5 6
The elements of 2D array are:
1 2 3
4 5 6
```

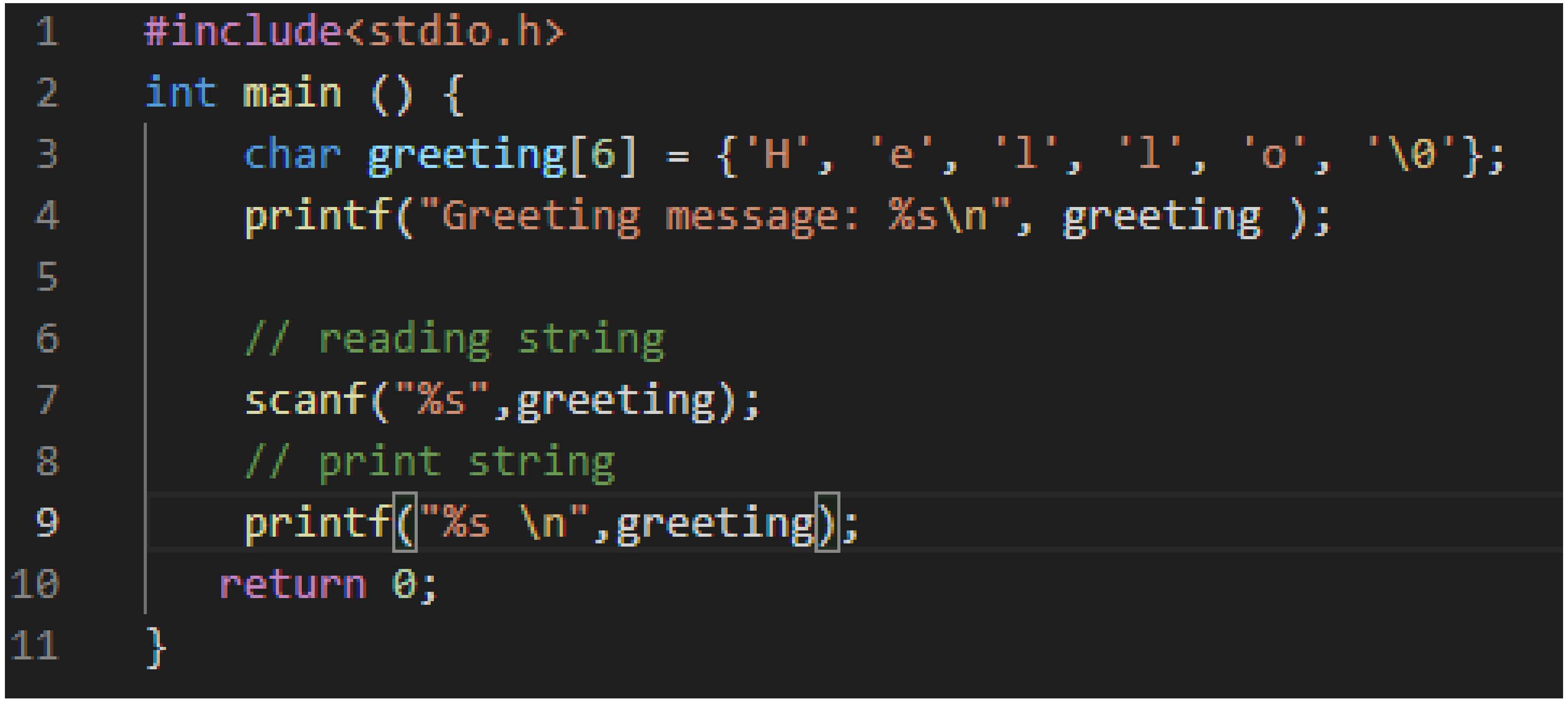
STRINGS

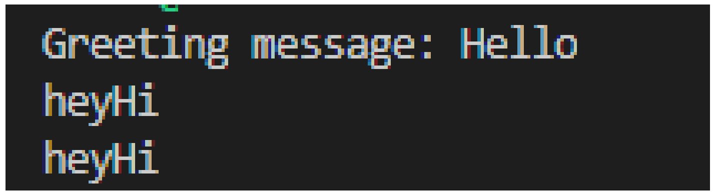
- Strings are actually one-dimensional array of characters terminated by a **null** character '\0'.
- `char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};`
- `char greeting[] = "Hello";`
- Functions of Strings are found in string.h header file;



STRING FUNCTIONS

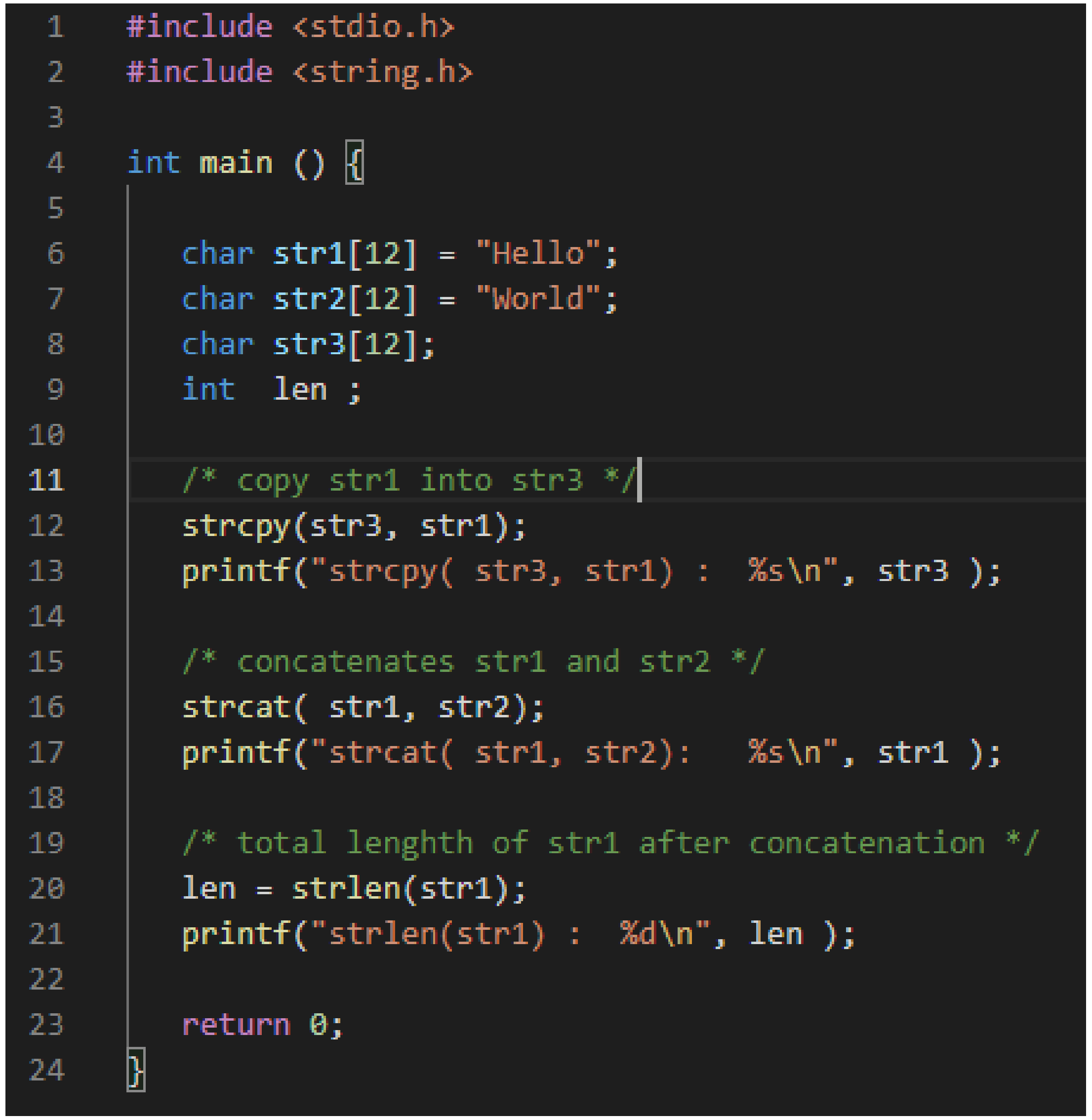
Sr.No.	Function & Purpose
1	**strcpy(s1, s2);** Copies string s2 into string s1.
2	**strcat(s1, s2);** Concatenates string s2 onto the end of string s1.
3	**strlen(s1);** Returns the length of string s1.
4	**strcmp(s1, s2);** Returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 if 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.


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

    // reading string
    scanf("%s",greeting);
    // print string
    printf("%s \n",greeting);
  return 0;
}
```

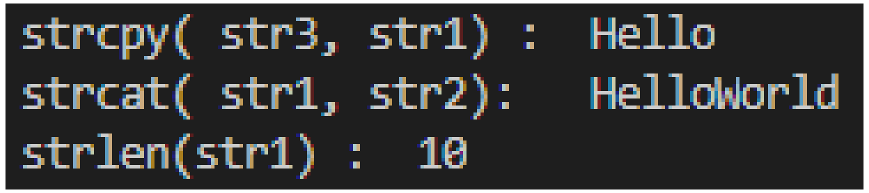

```
Greeting message: Hello
heyHi
heyHi
```

STRING EXAMPLE 1

STRING EXAMPLE 2



```c
#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;
}
```



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