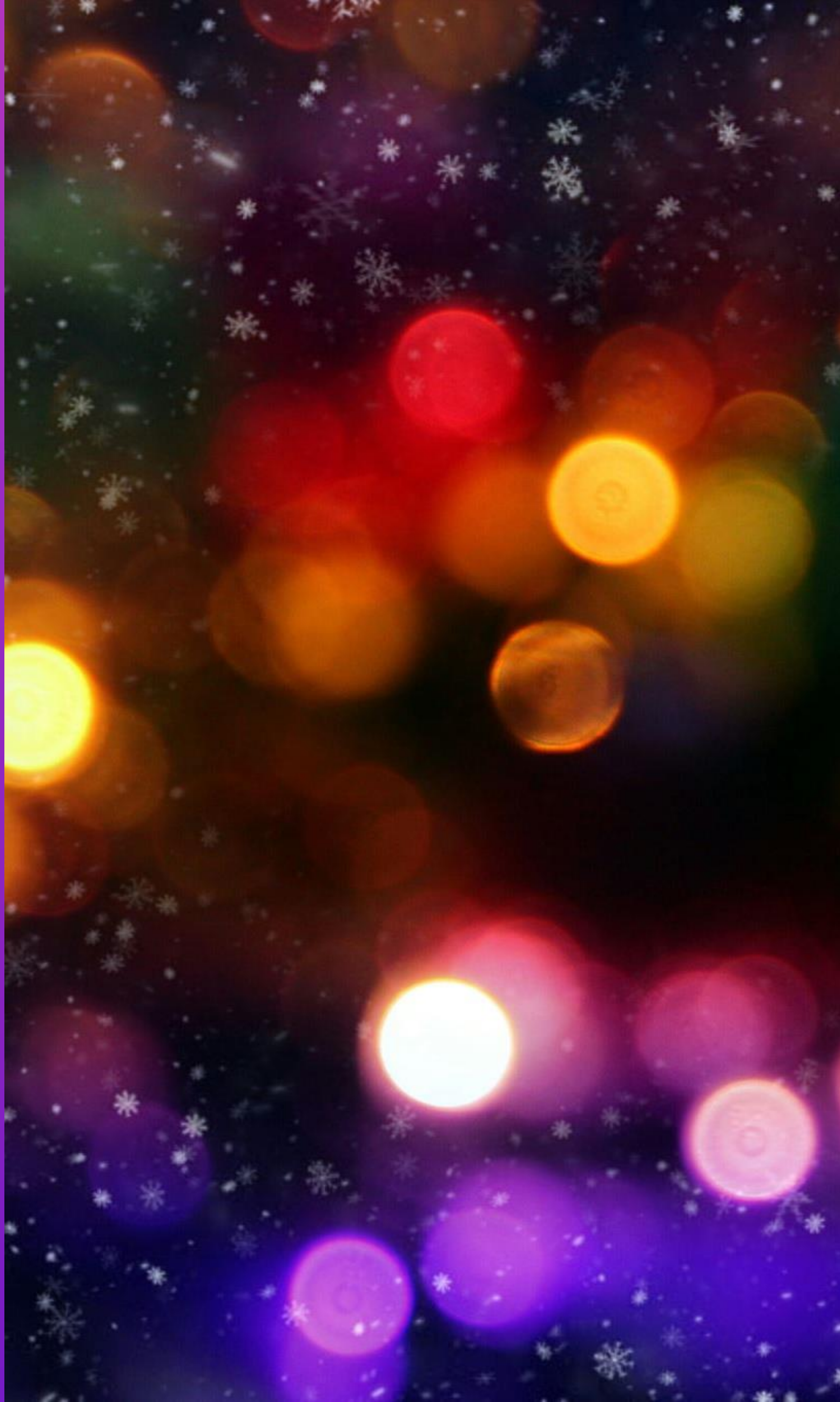
MNNIT COMPUTER CODING CLUB

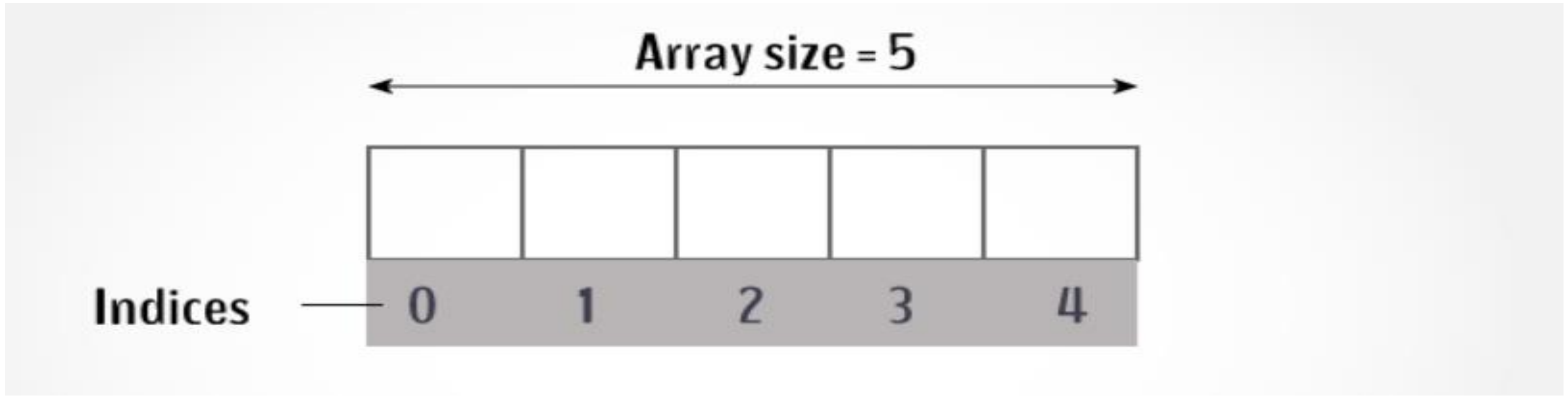
CLASS-9

BASICS OF C



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

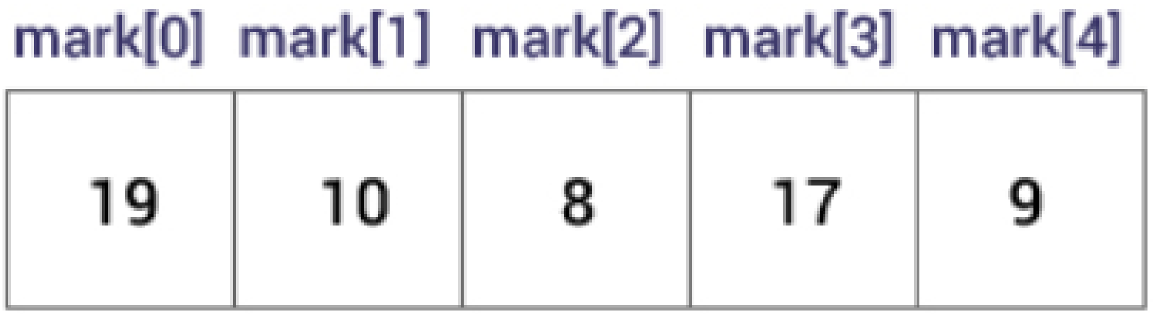
It is possible to initialize an array during declaration. For example,

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

An array can also be initialized like

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

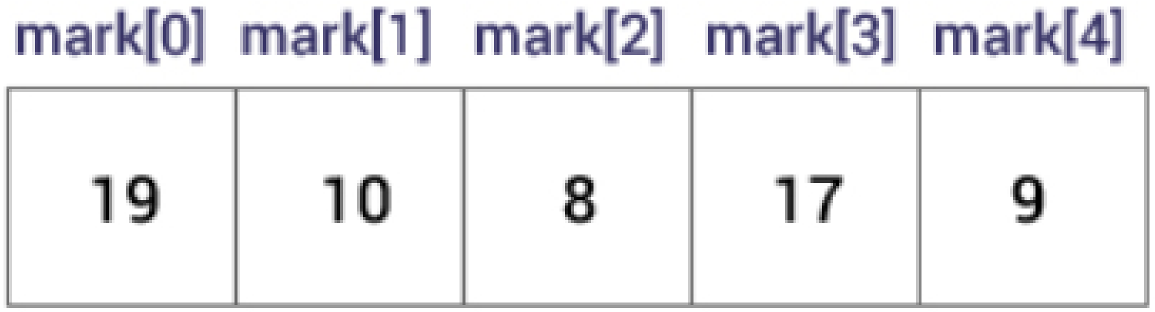
Here, we haven't specified the size. However, the compiler knows its size is 5 as we are initializing it with 5 elements.



ACCESSING 1D ARRAY ELEMENTS

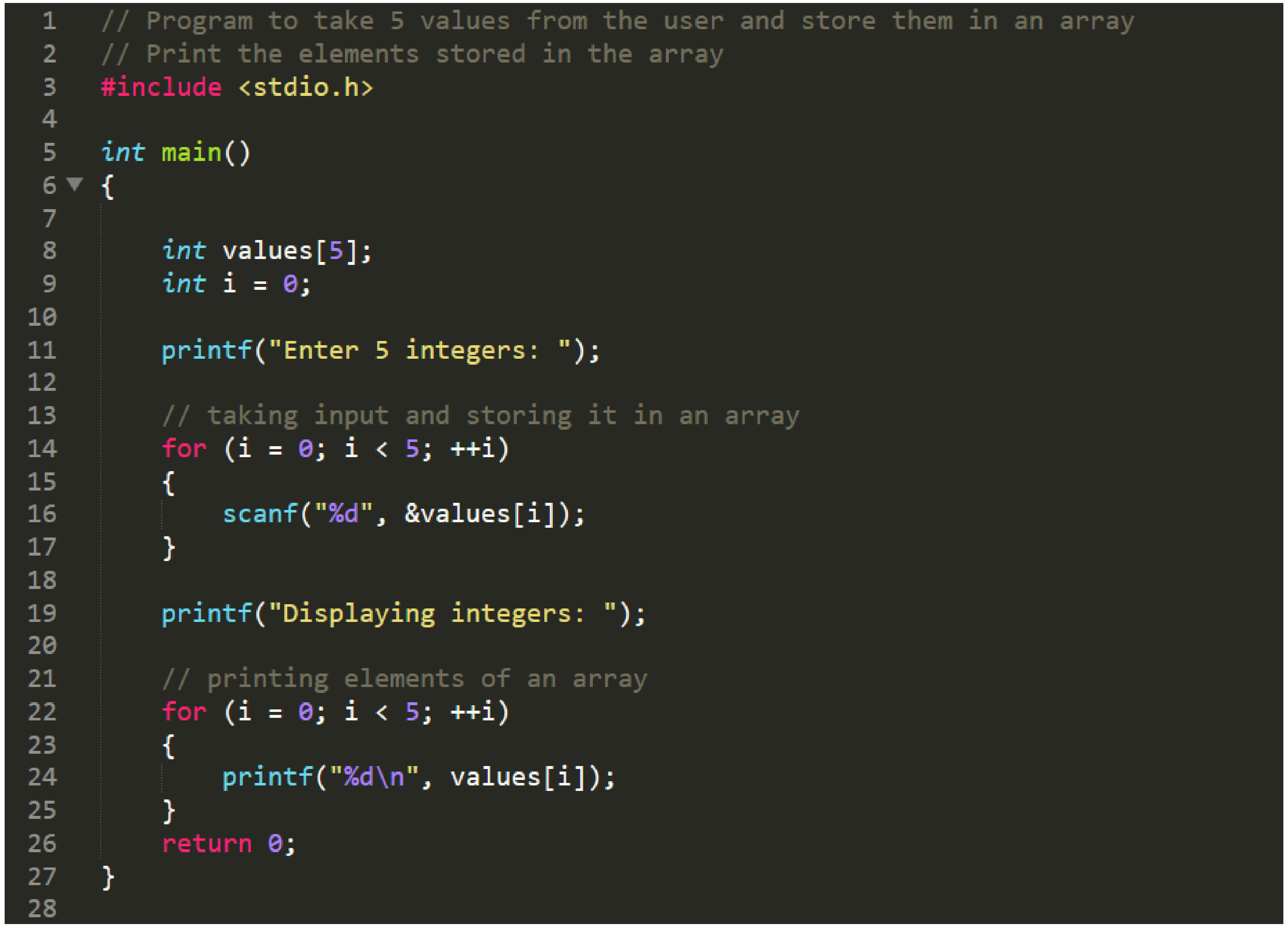
- Elements of array can be accessed by indices.
- Array index starts from 0 and goes till size of array minus 1.

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

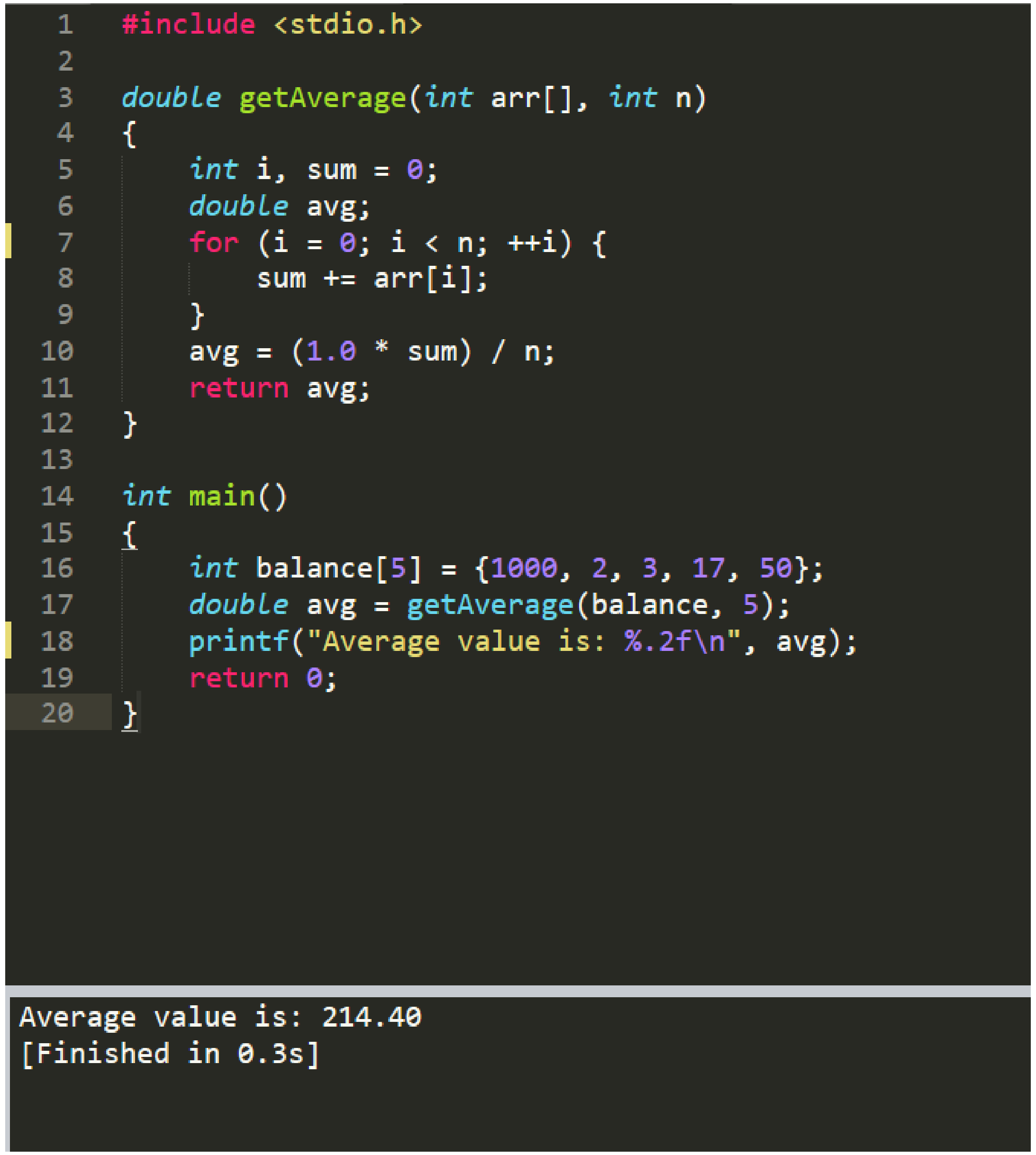


```
mark[0] is equal to 19
mark[1] is equal to 10
mark[2] is equal to 8
mark[3] is equal to 17
mark[4] is equal to 9
```

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]
```

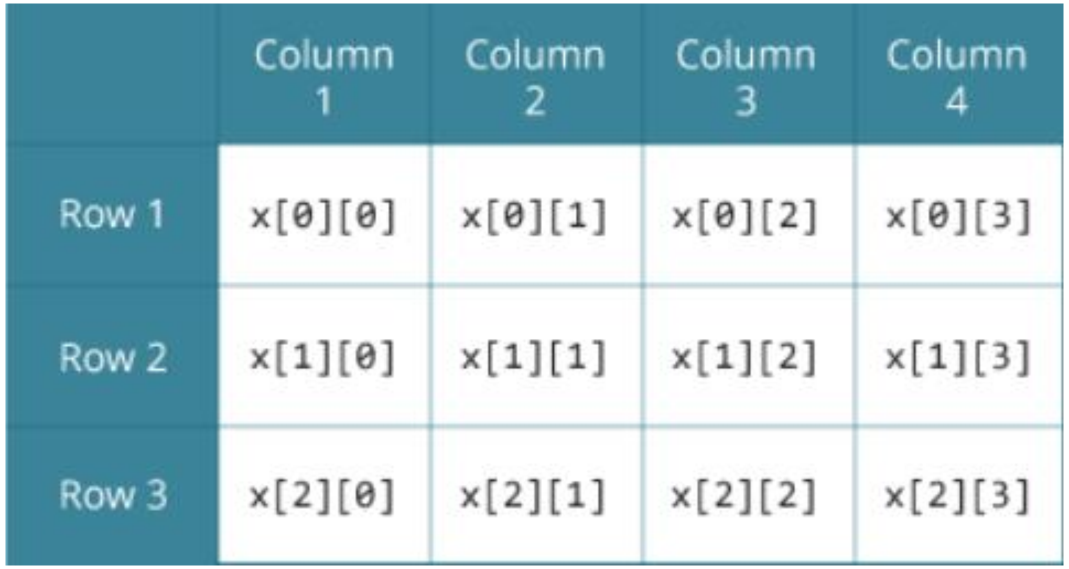
PASSING 1D ARRAY TO FUNCTION

- 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

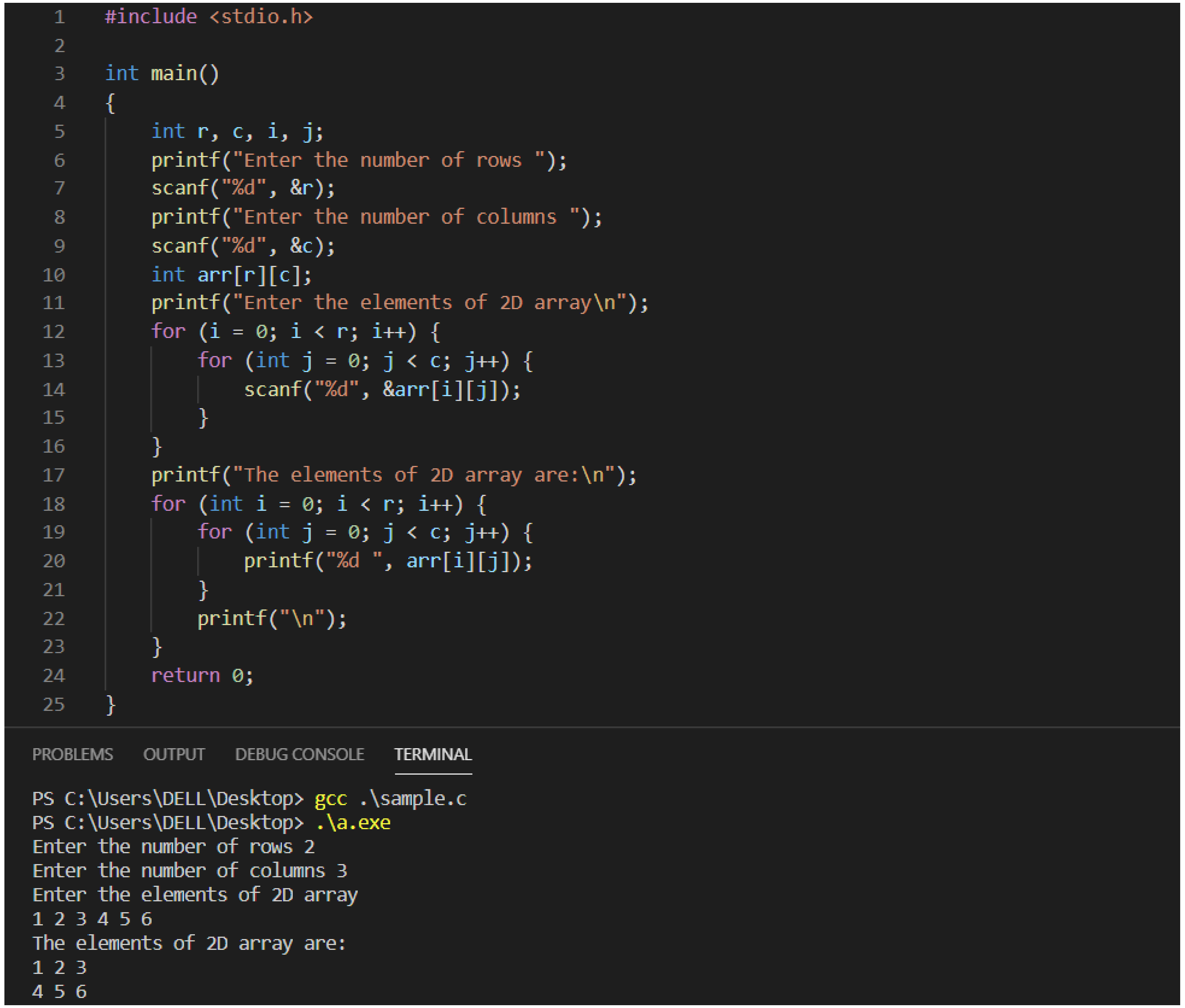
```
int arr[2][3] = {1, 3, 0, -1, 5, 9};
```

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

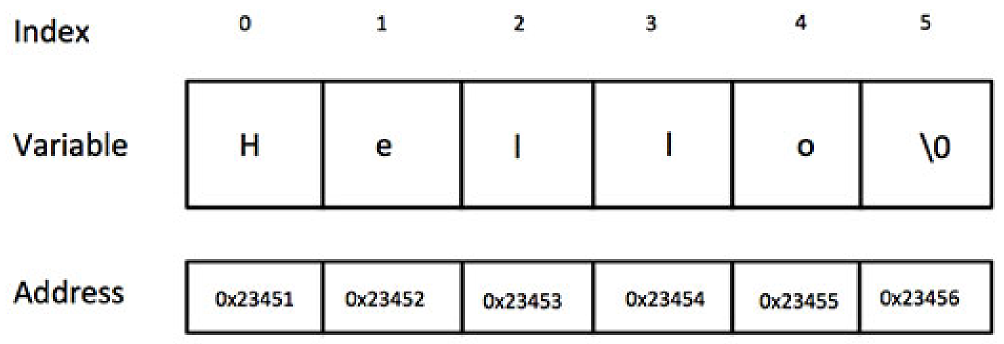


```c
#include <stdio.h>

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
```

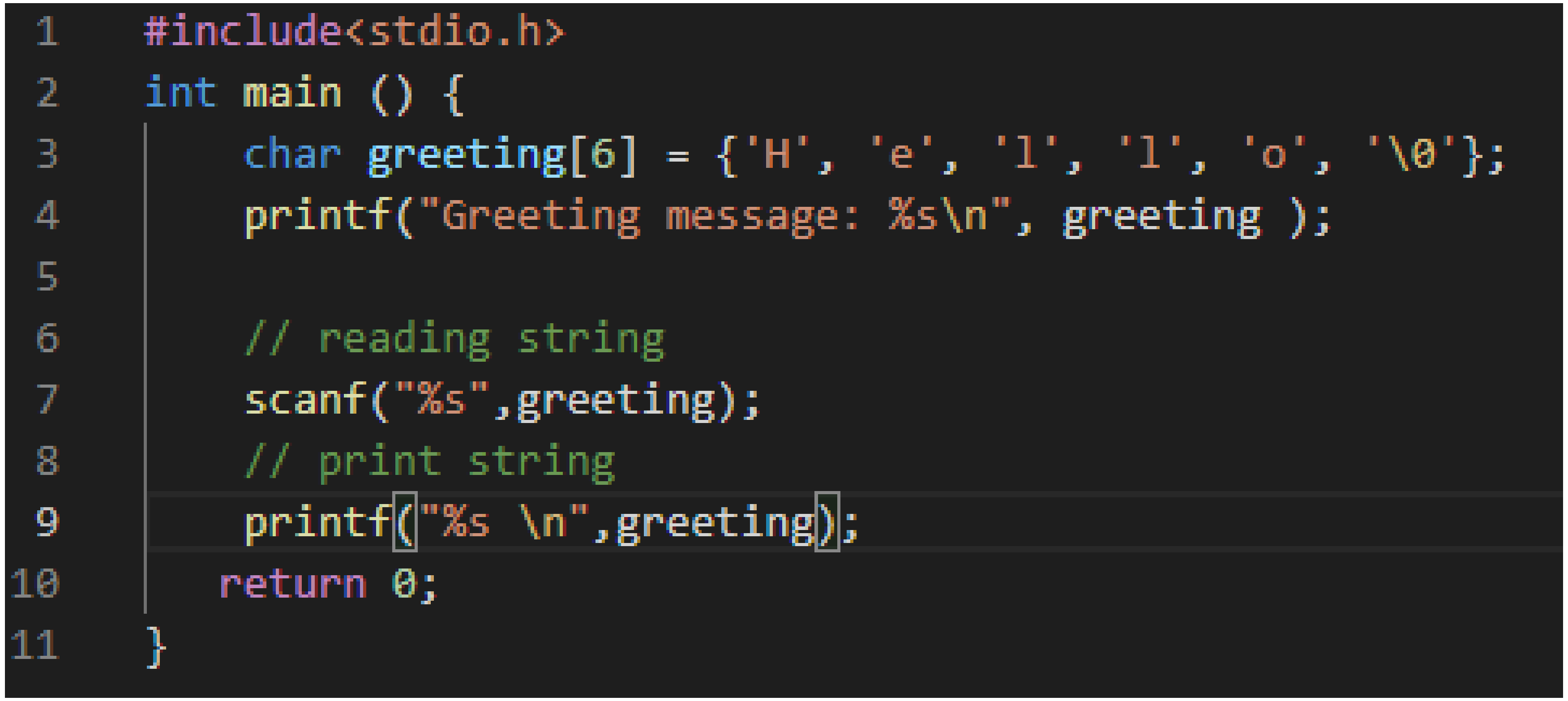
STRINGS

- 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;

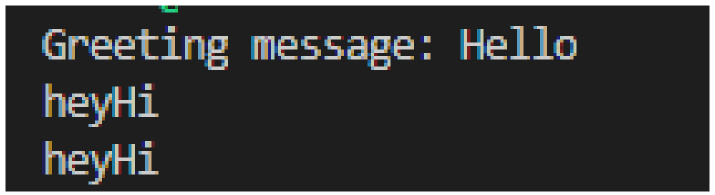


STRING FUNCTIONS

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.



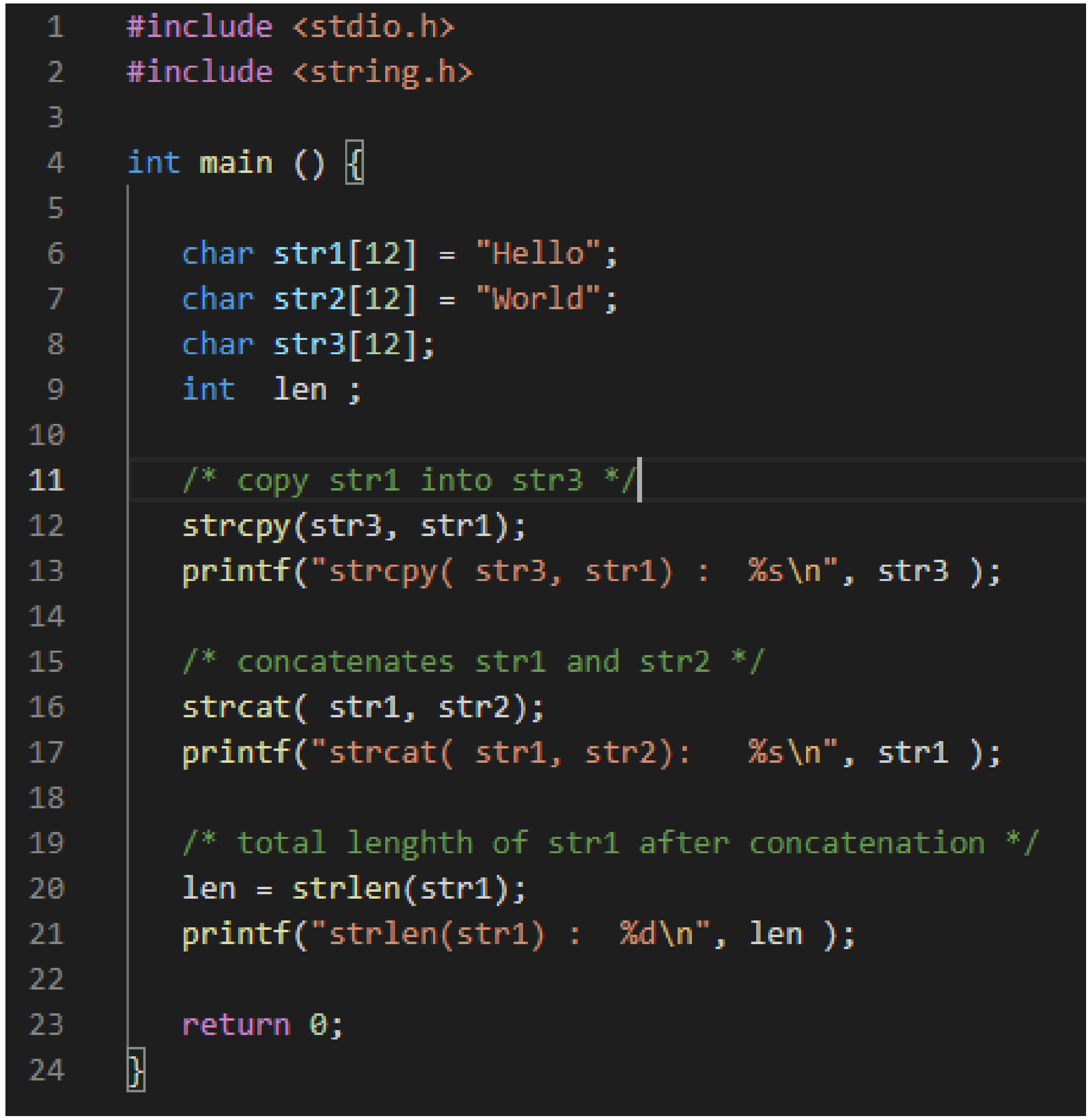
```c
#include<stdio.h>
int main () {
    char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
    printf("Greeting message: %s\n", greeting );

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



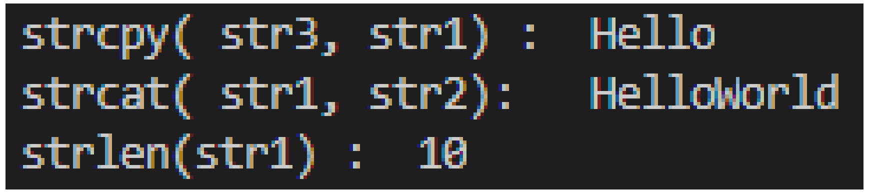
```
Greeting message: Hello
heyHi
heyHi
```

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
```

POINTERS

- The address of the first byte allocated to variable is known as address of variable.
- **'&' operator is the "address of" operator** in C which returns the address.
- The address operator cannot be used with a constant or an expression

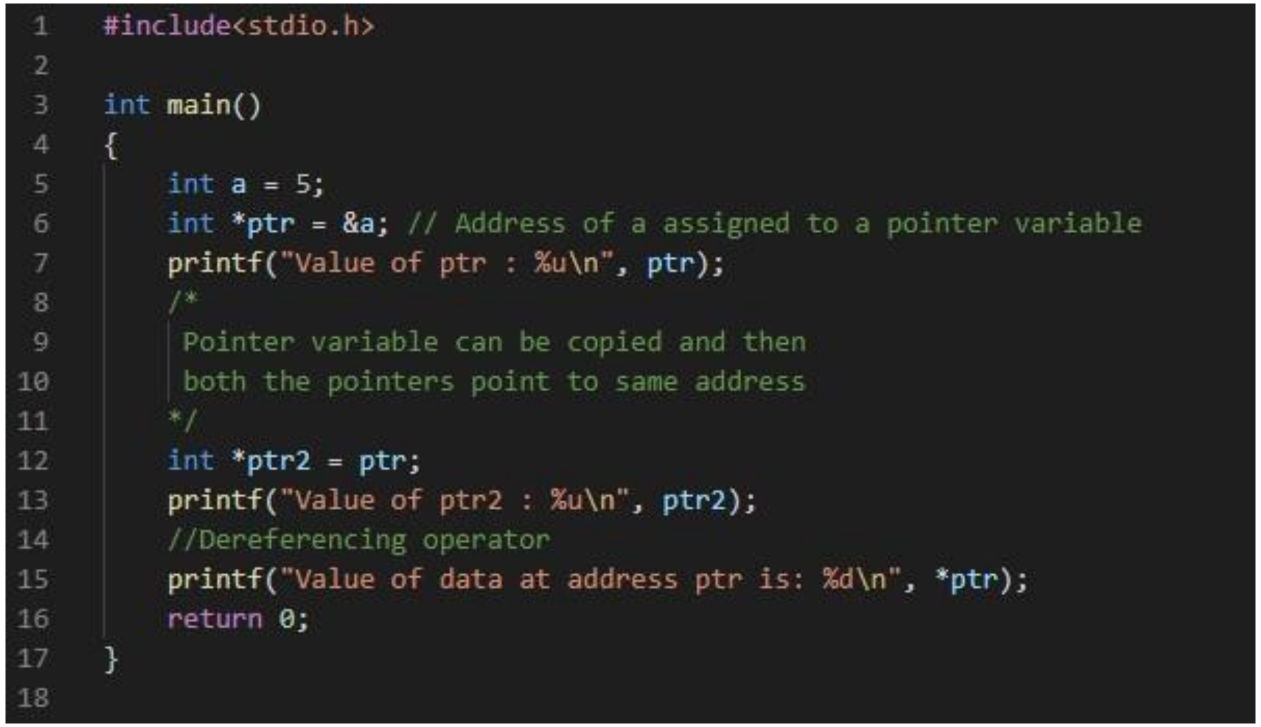
Eg -

int a = 5;	-	&a	Valid
float f = 8.6;	-	&f	Valid
Constant	-	&26	Invalid
Expression	-	&(a+f)	Invalid

- Pointer variables are used to store the memory address. Used like normal variables.

POINTERS

- The general syntax of declaration of pointer variable is: datatype *p_name;
- The size allocated to all pointer variables is same as they all store integers.
- **'*' is used to dereference the pointer**
- In the program attached,
  - ptr can be used to access the address of variable a and *ptr can be used to access its value
  - Writing *(&a) and a is same



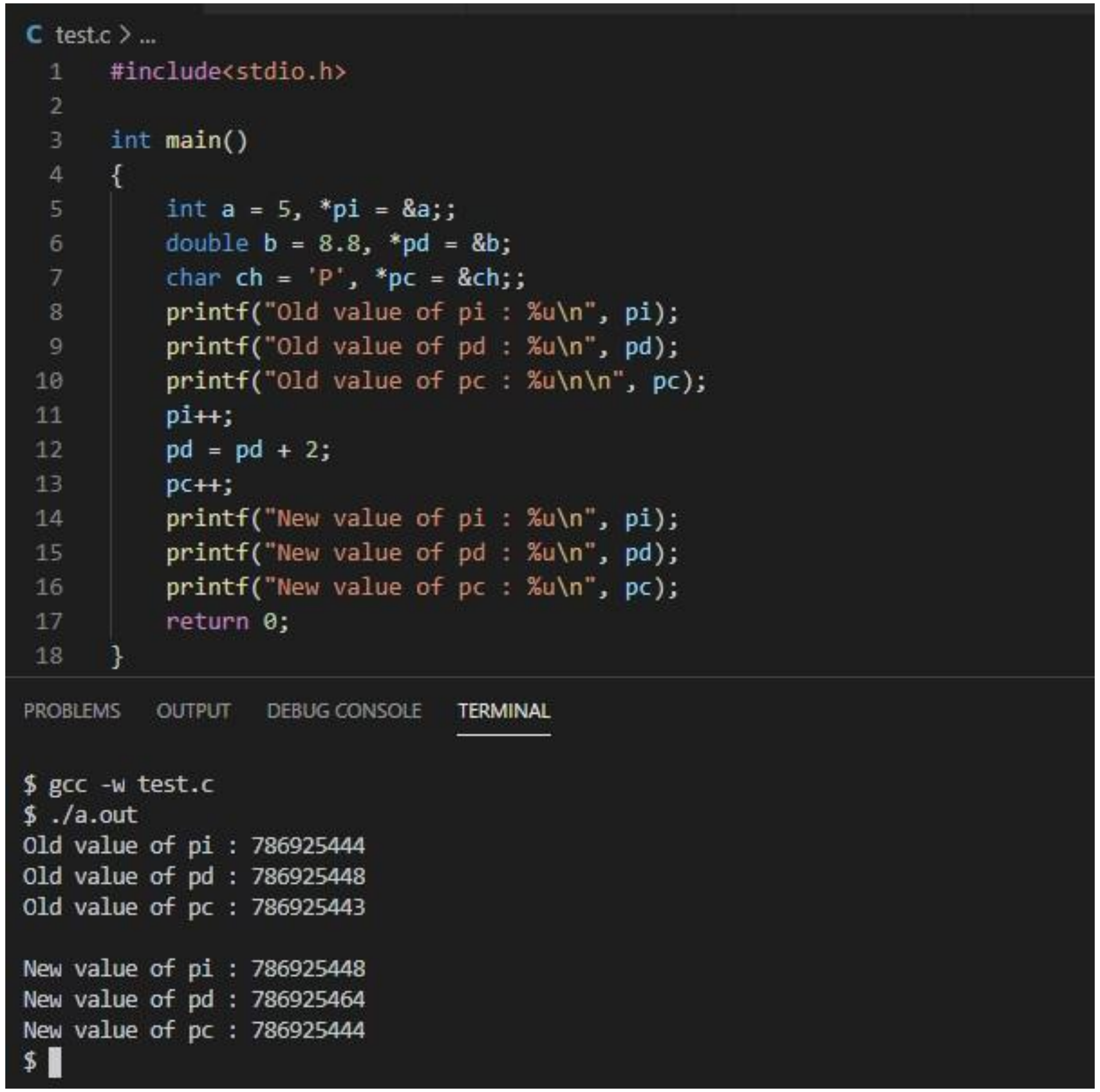
```c
#include<stdio.h>

int main()
{
    int a = 5;
    int *ptr = &a; // Address of a assigned to a pointer variable
    printf("Value of ptr : %u\n", ptr);
    /*
     Pointer variable can be copied and then
     both the pointers point to same address
    */
    int *ptr2 = ptr;
    printf("Value of ptr2 : %u\n", ptr2);
    //Dereferencing operator
    printf("Value of data at address ptr is: %d\n", *ptr);
    return 0;
}

```

POINTER ARITHMETIC

- Addition and subtraction of an integer and a pointer variable is supported
- Pointer variables can also be used for increment and decrement operation
- **The increment/decrement operation changes the value of pointer variable by the size of data type that it points to.**
- Subtraction of two pointer variables of same base type returns the number of values present between them

Eg – int *ptr1 = 2000, *ptr2 = 2020;

printf("%u\n", ptr2-ptr1);   Output : 5



```c
#include<stdio.h>

int main()
{
    int a = 5, *pi = &a;;
    double b = 8.8, *pd = &b;
    char ch = 'P', *pc = &ch;;
    printf("Old value of pi : %u\n", pi);
    printf("Old value of pd : %u\n", pd);
    printf("Old value of pc : %u\n\n", pc);
    pi++;
    pd = pd + 2;
    pc++;
    printf("New value of pi : %u\n", pi);
    printf("New value of pd : %u\n", pd);
    printf("New value of pc : %u\n", pc);
    return 0;
}
```

```
$ gcc -w test.c
$ ./a.out
Old value of pi : 786925444
Old value of pd : 786925448
Old value of pc : 786925443

New value of pi : 786925448
New value of pd : 786925464
New value of pc : 786925444
$
```

COMBINATION OF DEREFERENCE AND INCREMENT/DECREMENT

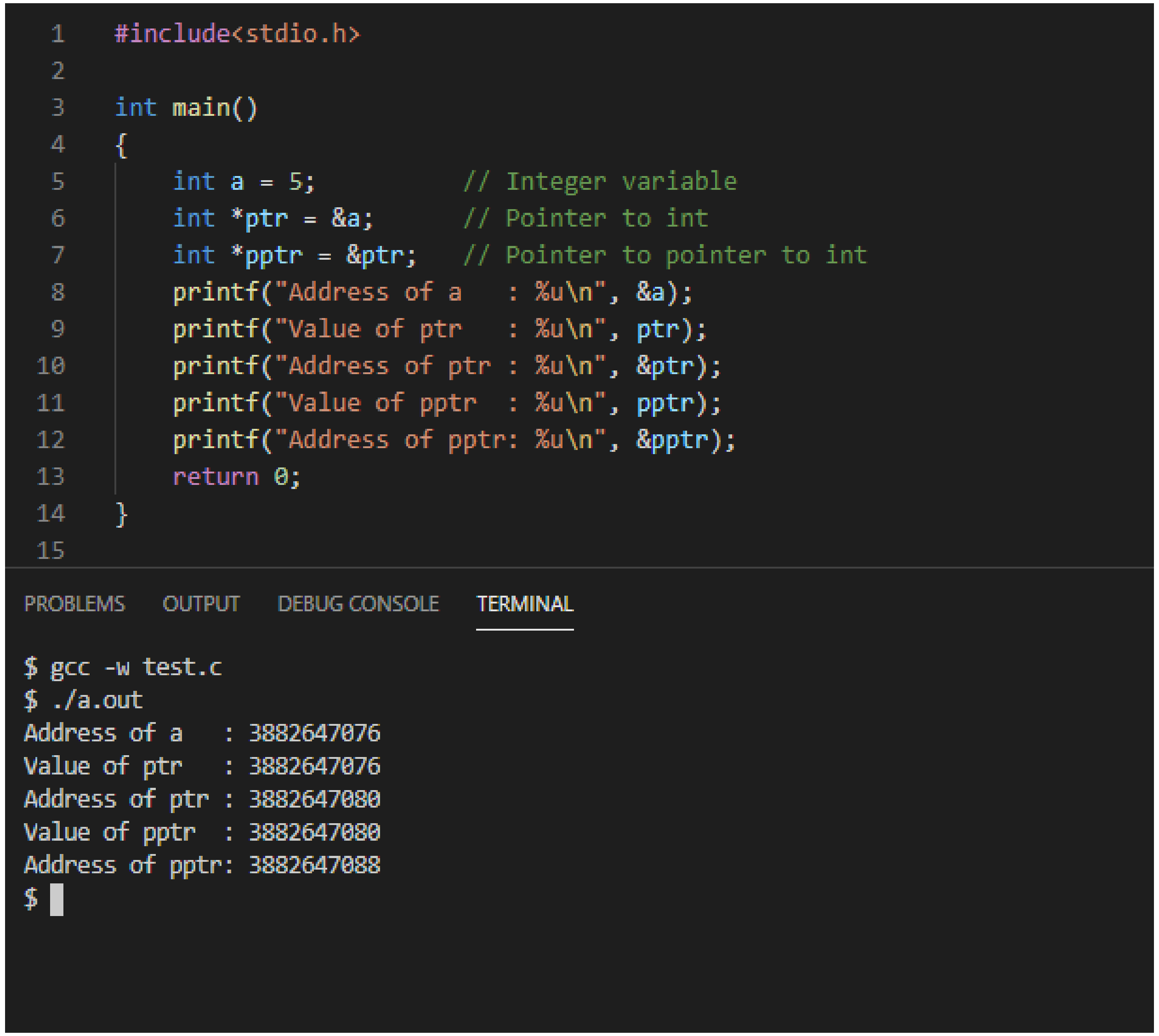
- The dereference operator (*), address of operator (&) and increment/decrement have same precedence and are **Right to Left Associative**.

Expression	Evaluation	
x = *ptr++	x = *ptr	ptr = ptr + 1
x = *++ptr	ptr = ptr + 1	x = *ptr
x = (*ptr)++	x = *ptr	*ptr = *ptr + 1
x = ++*ptr	*ptr = *ptr + 1	x = *ptr

POINTER TO POINTER

- Pointer variable contains an address, and this variable takes space in memory so it itself has an address
- A pointer-to-pointer variable is used to store the address of a pointer variable
- The general syntax of declaration of pointer variable is:

datatype **pp_name;



```c
#include<stdio.h>

int main()
{
    int a = 5;            // Integer variable
    int *ptr = &a;        // Pointer to int
    int *pptr = &ptr;     // Pointer to pointer to int
    printf("Address of a   : %u\n", &a);
    printf("Value of ptr   : %u\n", ptr);
    printf("Address of ptr : %u\n", &ptr);
    printf("Value of pptr  : %u\n", pptr);
    printf("Address of pptr: %u\n", &pptr);
    return 0;
}

```

```
$ gcc -w test.c
$ ./a.out
Address of a   : 3882647076
Value of ptr   : 3882647076
Address of ptr : 3882647080
Value of pptr  : 3882647080
Address of pptr: 3882647088
$ 
```