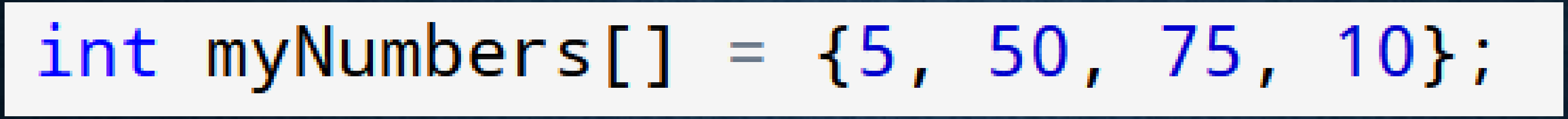
MODULE 2

Arrays

- Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.
- To create an array, define the data type (like int) and specify the name of the array followed by square brackets [ ].

    dataType arrayName[arraySize];

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

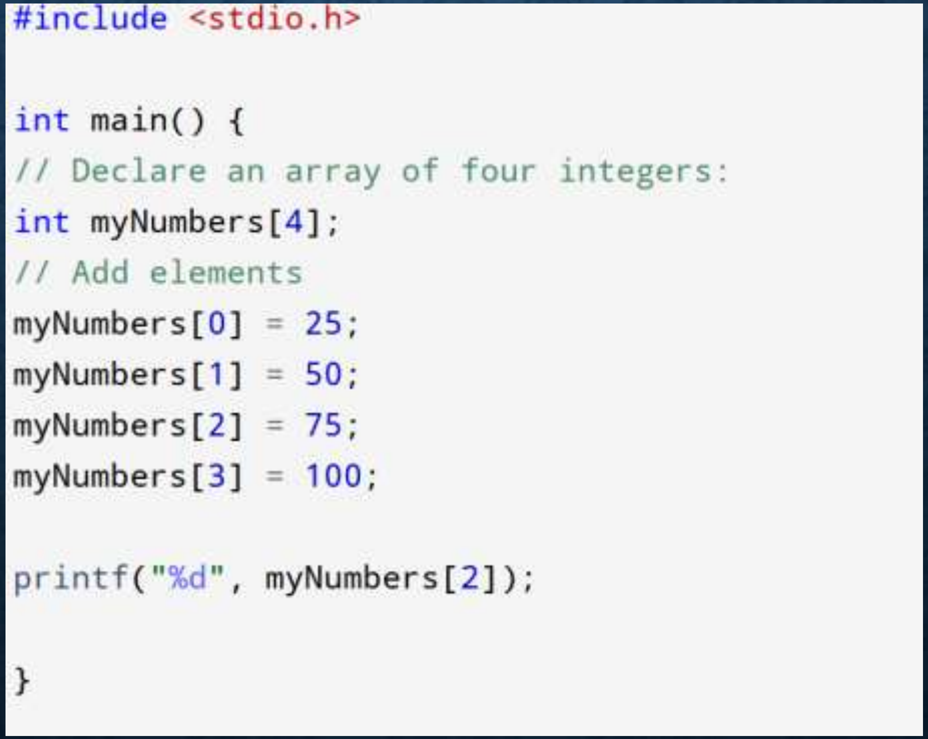


```
int myNumbers[] = {5, 50, 75, 10};
```

- To insert values to it, use a comma-separated list inside curly braces, and make sure all values are of the same data type:
- Here, we haven't specified the size. However, the compiler knows its size is 5 as we are initializing it with 5 elements.

- Another common way to create arrays, is to specify the size of the array, and add elements later:

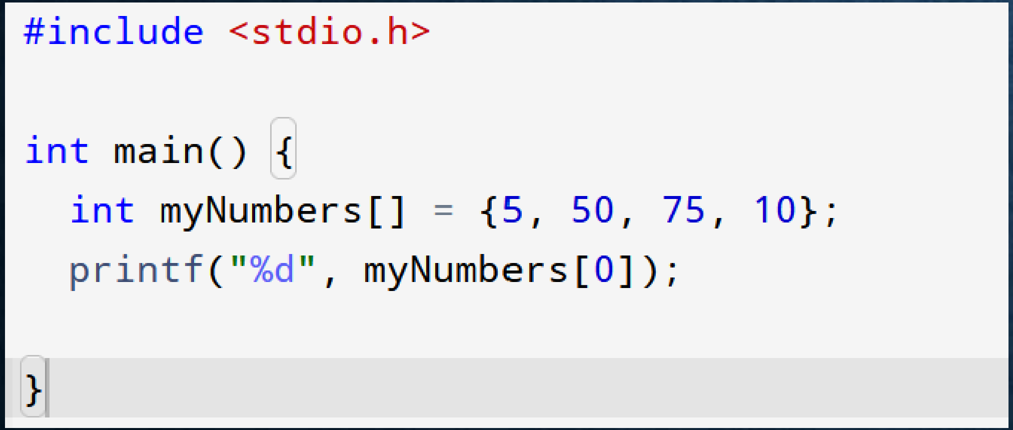
Example



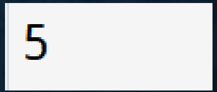
```c
#include <stdio.h>

int main() {
// Declare an array of four integers:
int myNumbers[4];
// Add elements
myNumbers[0] = 25;
myNumbers[1] = 50;
myNumbers[2] = 75;
myNumbers[3] = 100;

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

}
```

Access the elements of an array

- To access an array element, refer to its **index number**.
- Array indexes start with **0**: [0] is the first element. [1] is the second element, etc.



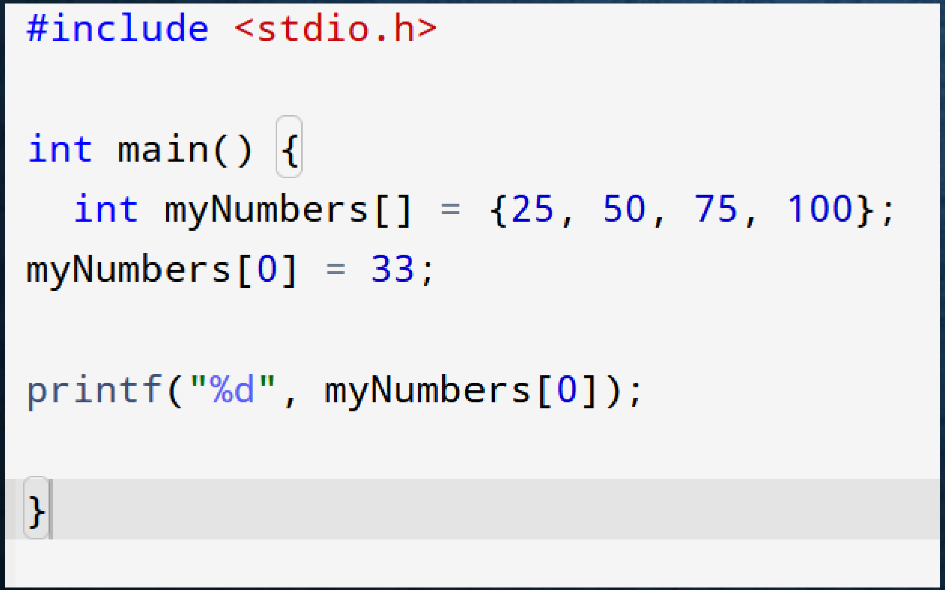
```c
#include <stdio.h>

int main() {
  int myNumbers[] = {5, 50, 75, 10};
  printf("%d", myNumbers[0]);

}
```



```
5
```

CHANGE AN ARRAY ELEMENT

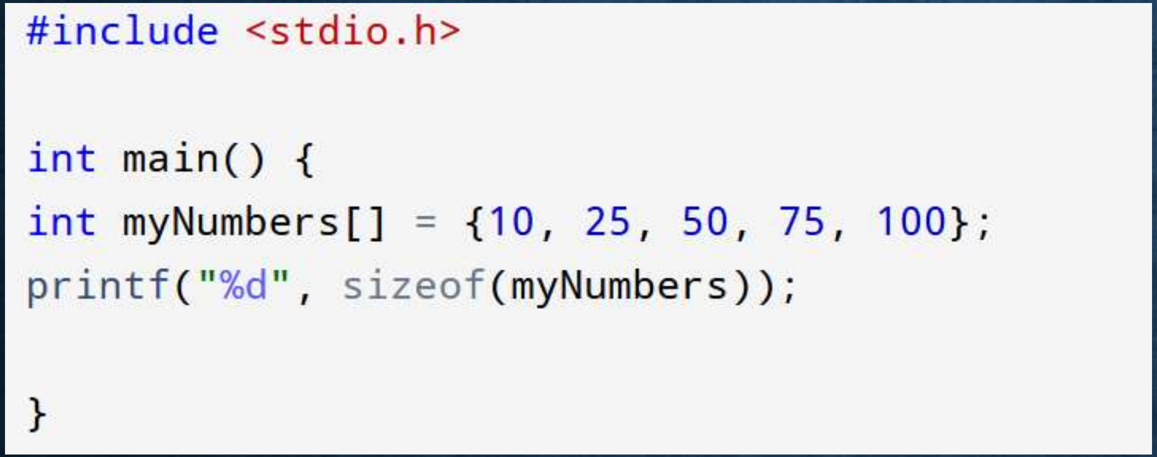
- To change the value of a specific element, refer to the index number:
- Eg: myNumbers[0] = 33;



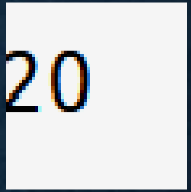
```
#include <stdio.h>

int main() {
  int myNumbers[] = {25, 50, 75, 100};
myNumbers[0] = 33;

printf("%d", myNumbers[0]);

}
```

GET ARRAY SIZE OR LENGTH

- To get the size of an array, you can use the sizeof operator:



```
#include <stdio.h>

int main() {
int myNumbers[] = {10, 25, 50, 75, 100};
printf("%d", sizeof(myNumbers));

}
```



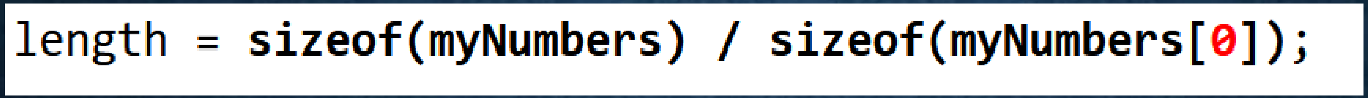
```
20
```

Why did the result show 20 instead of 5, when the array contains 5 elements?

- It is because the sizeof operator returns the size of a type in bytes.

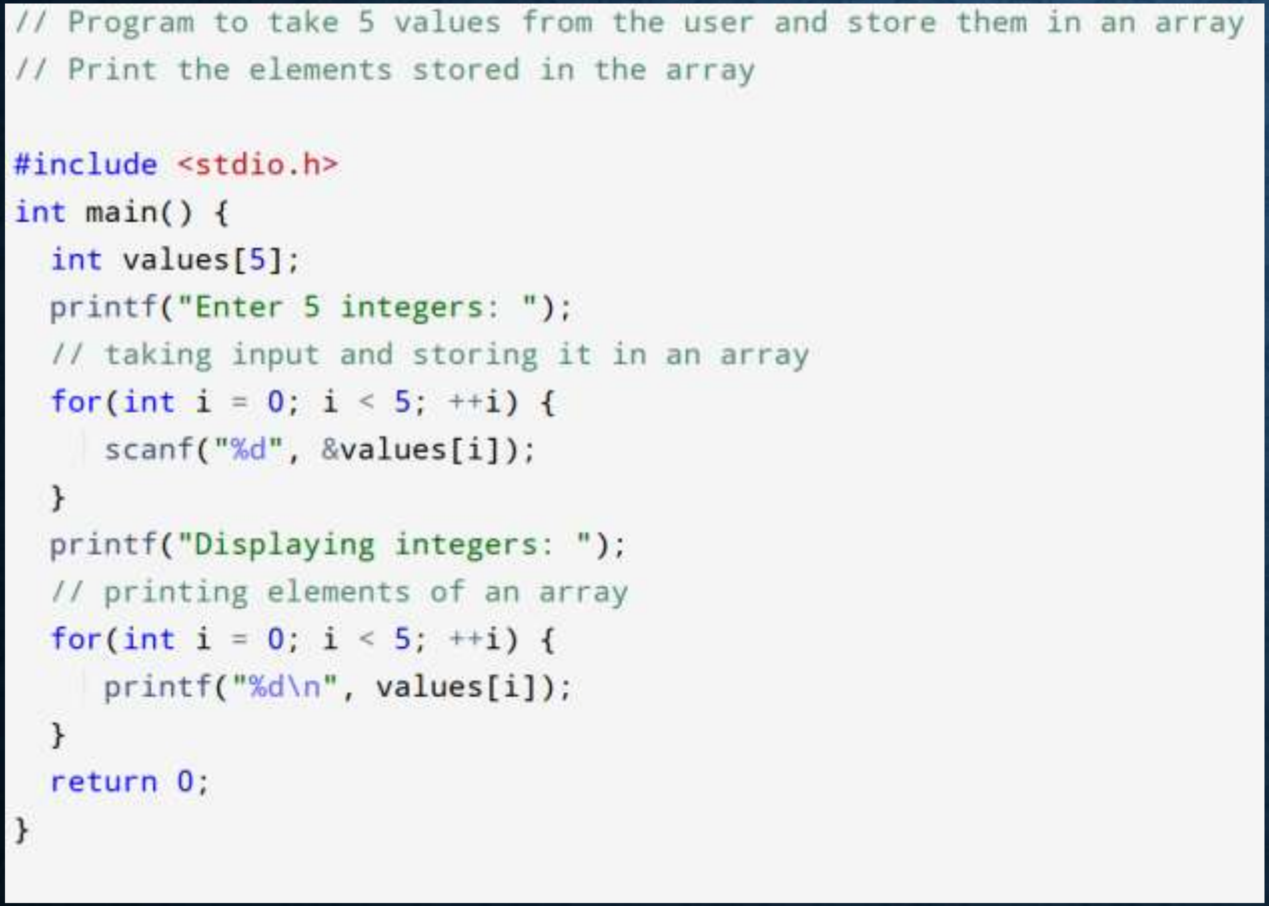
int type is usually 4 bytes, so from the example above, 4 x 5 (4 bytes x 5 elements) = 20 bytes.

- when you just want to find out how many elements an array has, you can use the following formula (which divides the size of the array by the size of the first element in the array):

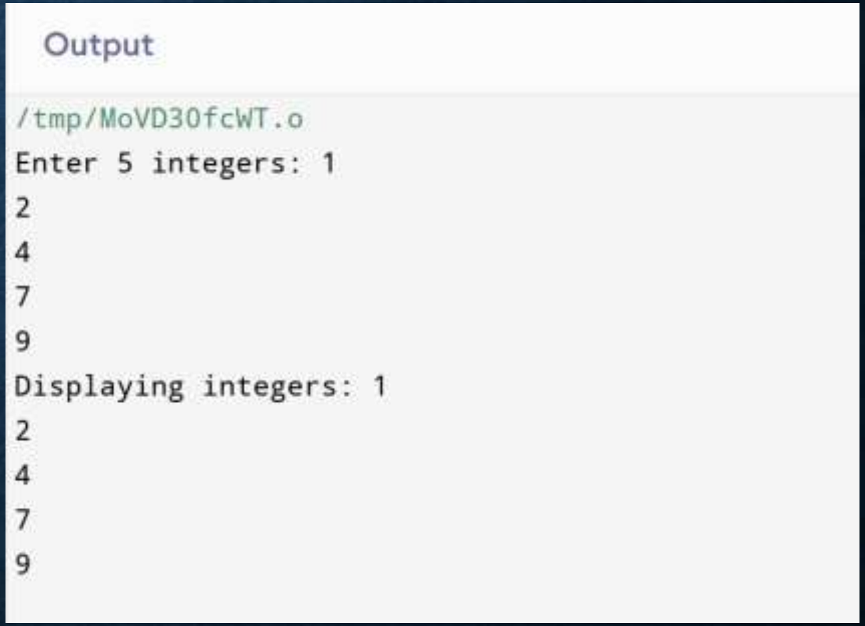


```
length = sizeof(myNumbers) / sizeof(myNumbers[0]);
```

ARRAY INPUT/OUTPUT



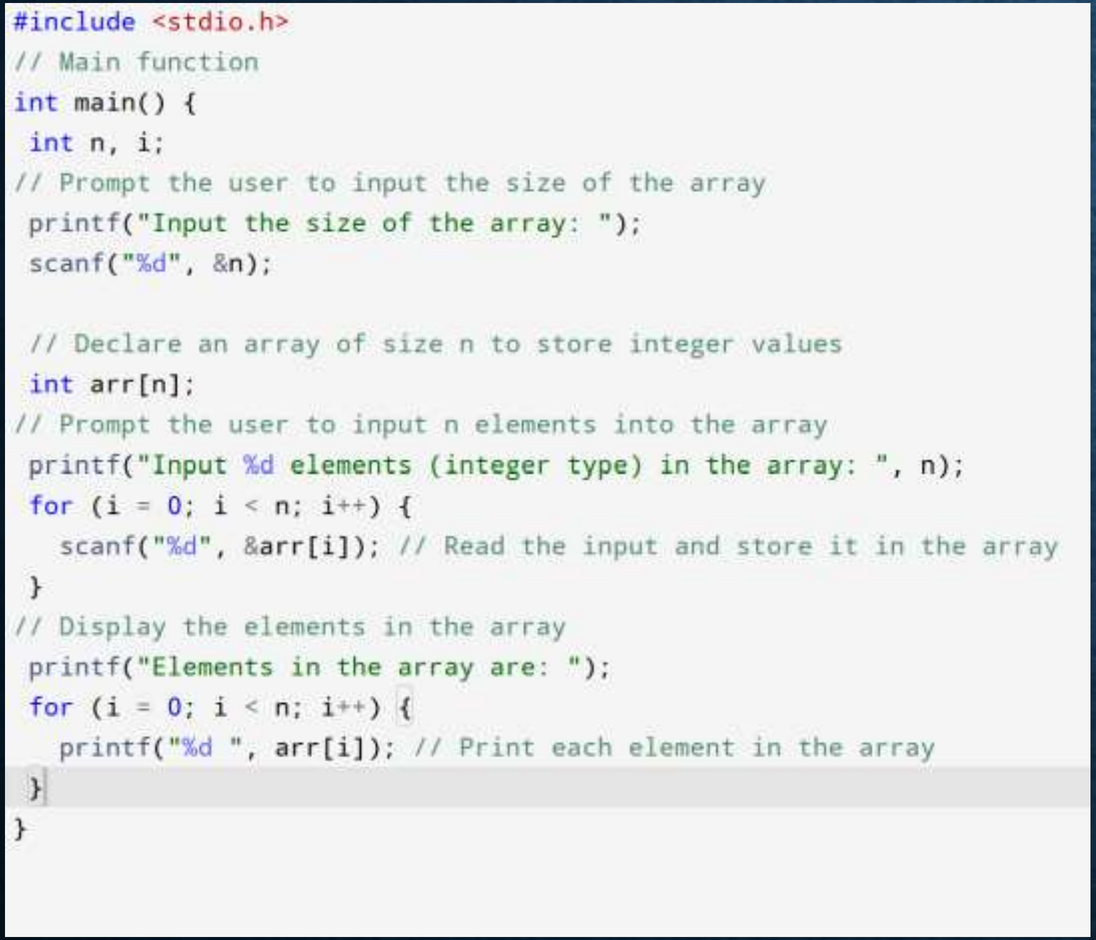
```c
// Program to take 5 values from the user and store them in an array
// Print the elements stored in the array

#include <stdio.h>
int main() {
  int values[5];
  printf("Enter 5 integers: ");
  // taking input and storing it in an array
  for(int i = 0; i < 5; ++i) {
     scanf("%d", &values[i]);
  }
  printf("Displaying integers: ");
  // printing elements of an array
  for(int i = 0; i < 5; ++i) {
     printf("%d\n", values[i]);
  }
  return 0;
}
```



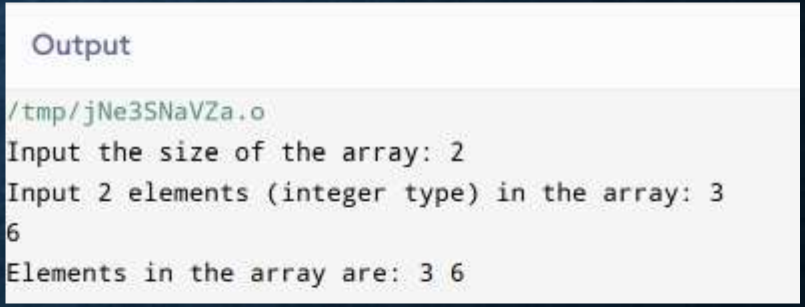
Output

```
/tmp/MoVD30fcWT.o
Enter 5 integers: 1
2
4
7
9
Displaying integers: 1
2
4
7
9
```

Read and print elements of a C array of any size?



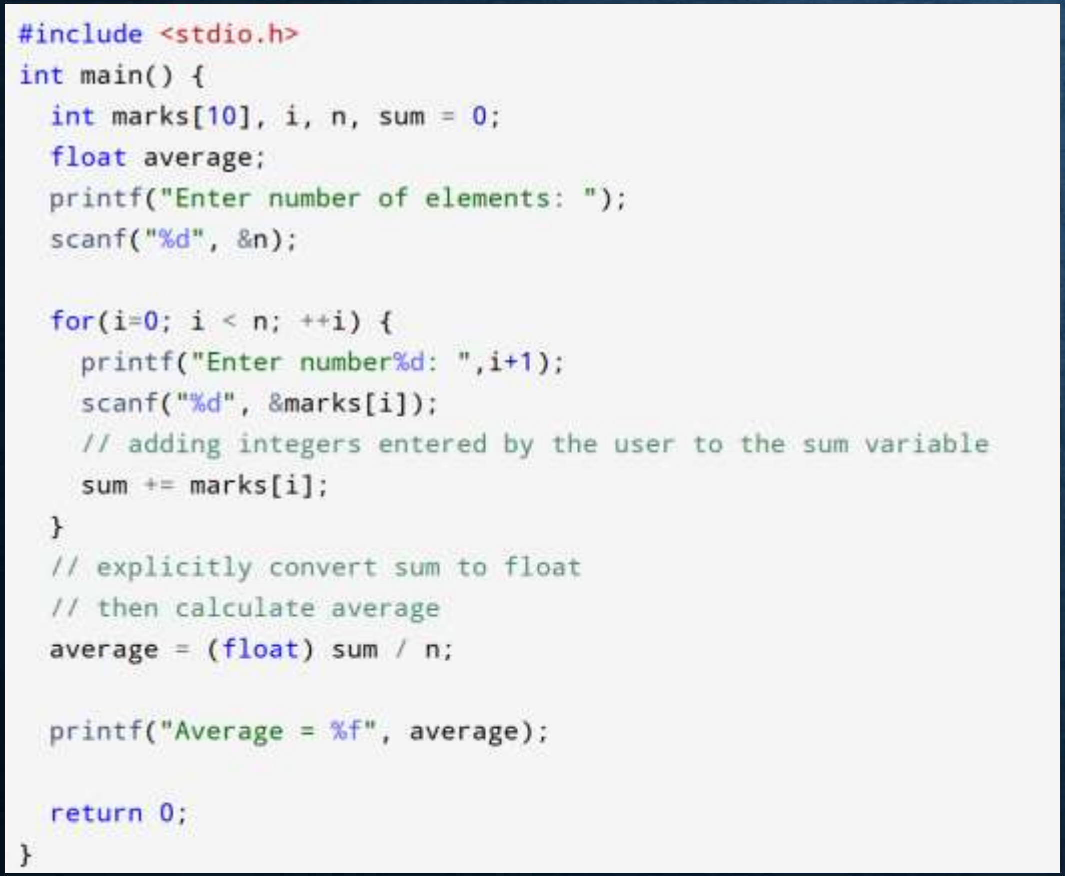
```c
#include <stdio.h>
// Main function
int main() {
 int n, i;
// Prompt the user to input the size of the array
 printf("Input the size of the array: ");
 scanf("%d", &n);

 // Declare an array of size n to store integer values
 int arr[n];
// Prompt the user to input n elements into the array
 printf("Input %d elements (integer type) in the array: ", n);
 for (i = 0; i < n; i++) {
   scanf("%d", &arr[i]); // Read the input and store it in the array
 }
// Display the elements in the array
 printf("Elements in the array are: ");
 for (i = 0; i < n; i++) {
   printf("%d ", arr[i]); // Print each element in the array
 }
}
```



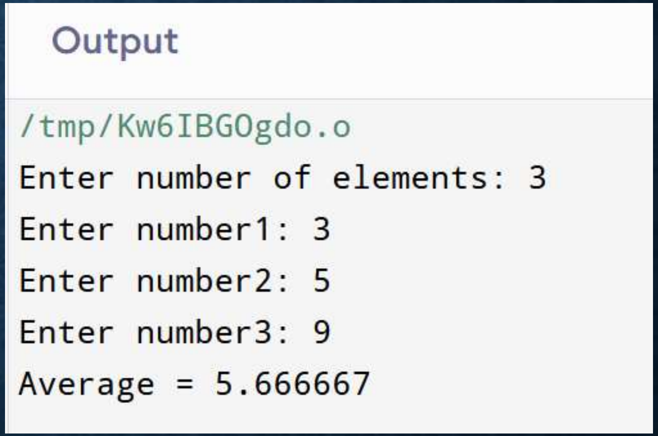
Output

```
/tmp/jNe3SNaVZa.o
Input the size of the array: 2
Input 2 elements (integer type) in the array: 3
6
Elements in the array are: 3 6
```

Program to find the average of n numbers using arrays

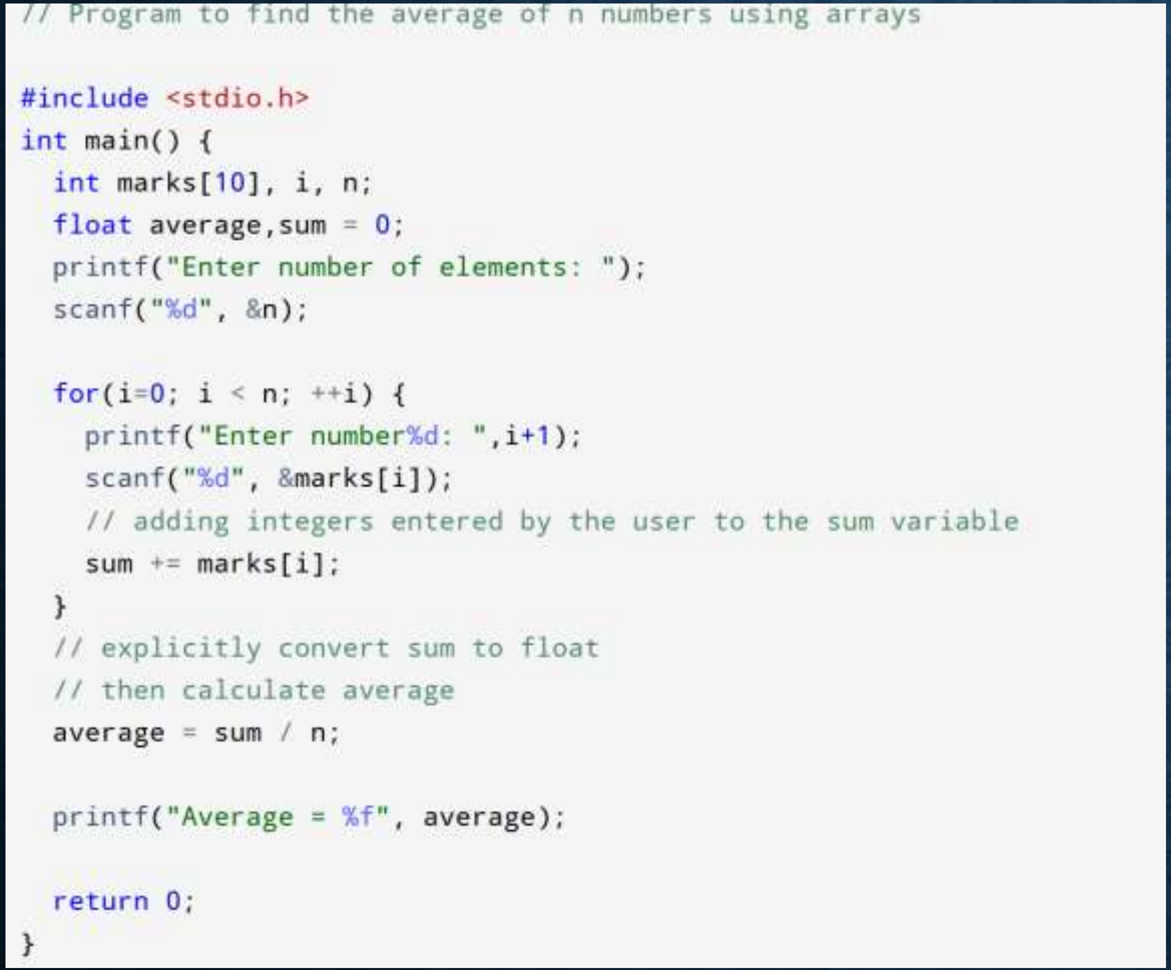


```c
#include <stdio.h>
int main() {
  int marks[10], i, n, sum = 0;
  float average;
  printf("Enter number of elements: ");
  scanf("%d", &n);

  for(i=0; i < n; ++i) {
    printf("Enter number%d: ",i+1);
    scanf("%d", &marks[i]);
    // adding integers entered by the user to the sum variable
    sum += marks[i];
  }
  // explicitly convert sum to float
  // then calculate average
  average = (float) sum / n;

  printf("Average = %f", average);

  return 0;
}
```

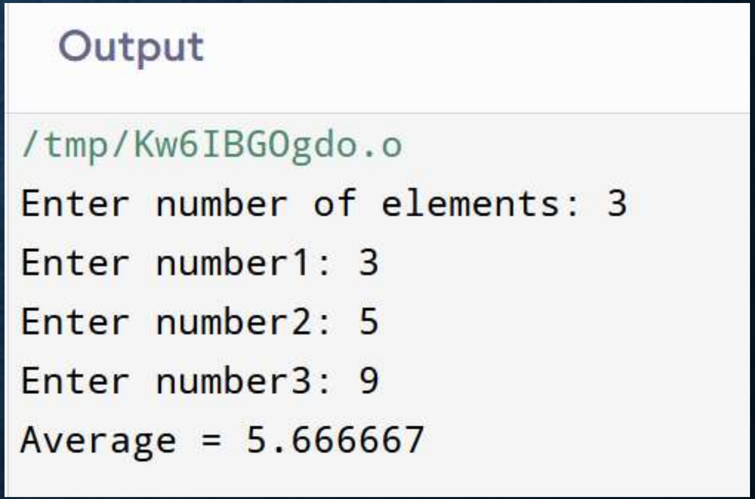


Output

```
/tmp/Kw6IBGOgdo.o
Enter number of elements: 3
Enter number1: 3
Enter number2: 5
Enter number3: 9
Average = 5.666667
```



```c
// Program to find the average of n numbers using arrays

#include <stdio.h>
int main() {
  int marks[10], i, n;
  float average,sum = 0;
  printf("Enter number of elements: ");
  scanf("%d", &n);

  for(i=0; i < n; ++i) {
    printf("Enter number%d: ",i+1);
    scanf("%d", &marks[i]);
    // adding integers entered by the user to the sum variable
    sum += marks[i];
  }
  // explicitly convert sum to float
  // then calculate average
  average = sum / n;

  printf("Average = %f", average);

  return 0;
}
```



Output

```
/tmp/Kw6IBGOgdo.o
Enter number of elements: 3
Enter number1: 3
Enter number2: 5
Enter number3: 9
Average = 5.666667
```

MULTIDIMENSIONAL ARRAYS IN C

- A multi-dimensional array can be defined as an array that has more than one dimension. Having more than one dimension means that it can grow in multiple directions. Some popular multidimensional arrays are 2D arrays and 3D arrays.
- Syntax

- The general form of declaring N-dimensional arrays is shown below:

```
Datatype  arr_name[size1][size2]….[sizeN];
```

- dataype: Type of data to be stored in the array.
- arr_name: Name assigned to the array.
- size1, size2,…, sizeN: Size of each dimension.

INITIALIZATION OF 2D ARRAYS

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

Or

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

***Note:** The number of elements in initializer list should always be either less than or equal to the total number of elements in the array.*

- The elements will be stored in the array from left to right and top to bottom.
- So, the first 4 elements from the left will be filled in the first row, the next 4 elements in the second row, and so on.
- This is clearly shown in the second syntax where each set of inner braces represents one row.

ACCESS THE ELEMENTS OF A 2D ARRAY

- To access an element of a two-dimensional array, you must specify the index number of both the row and column.
- This statement accesses the value of the element in the **first row (0)** and **third column (2)** of the **matrix** array.

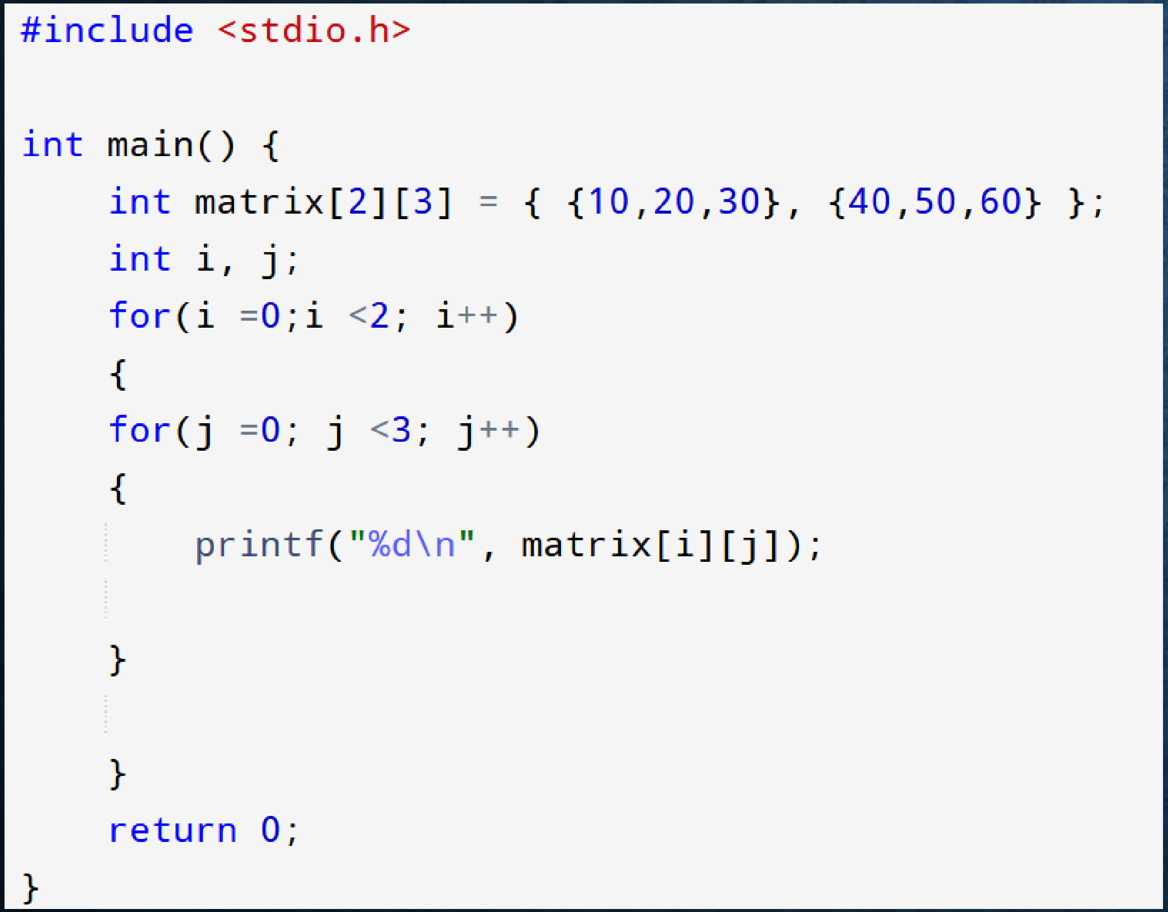
Example

```
int matrix[2][3] = { {10, 20, 30}, {40, 50, 60} };


printf("%d", matrix[0][1]);  // Outputs 20
```

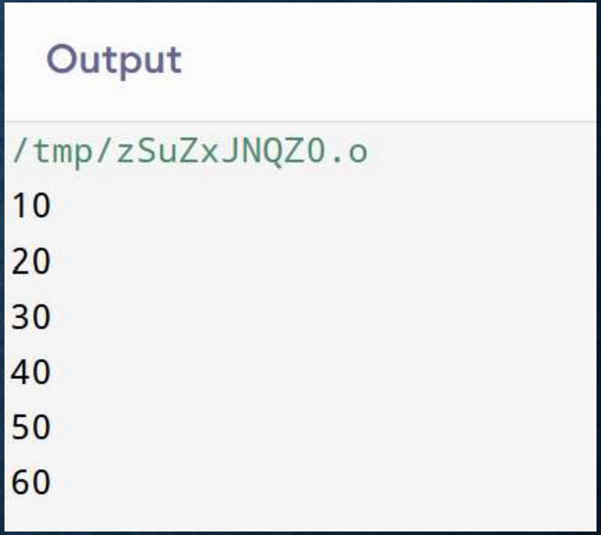
Updation

```
matrix[0][0] = 15;
```

LOOP THROUGH A 2D ARRAY



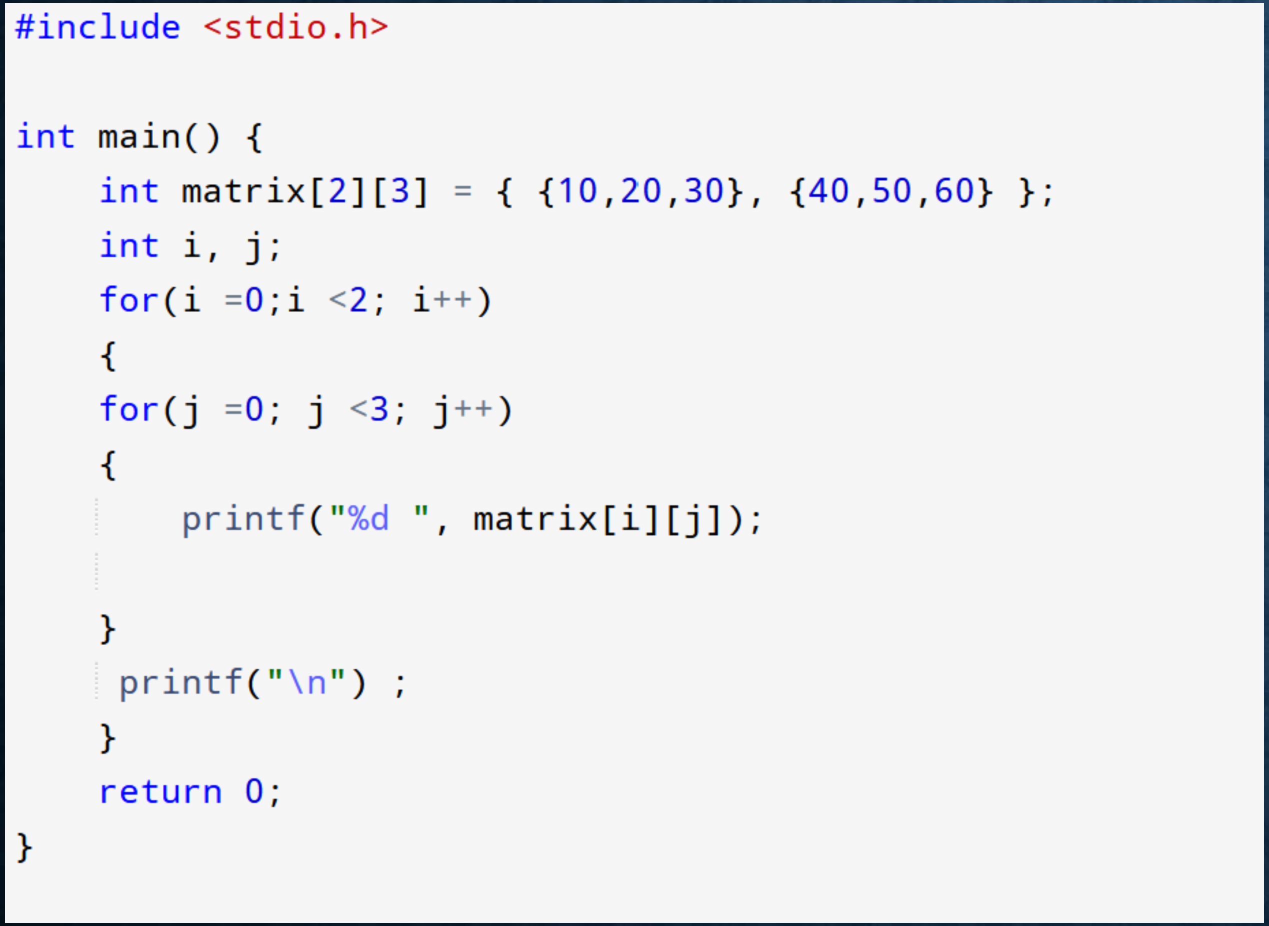
```c
#include <stdio.h>

int main() {
    int matrix[2][3] = { {10,20,30}, {40,50,60} };
    int i, j;
    for(i =0;i <2; i++)
    {
    for(j =0; j <3; j++)
    {
        printf("%d\n", matrix[i][j]);

    }

    }
    return 0;
}
```



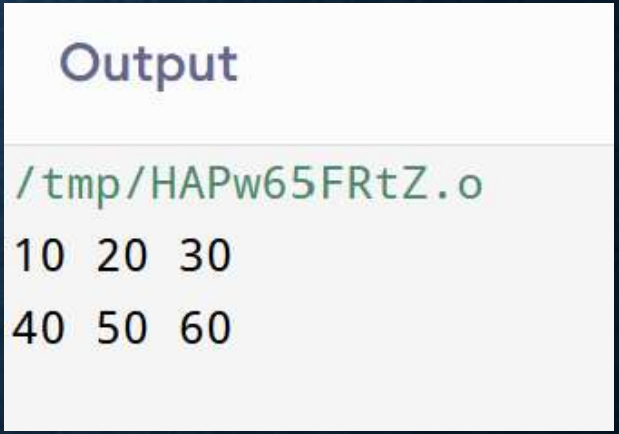
Output

```
/tmp/zSuZxJNQZ0.o
10
20
30
40
50
60
```

LOOP THROUGH A 2D ARRAY



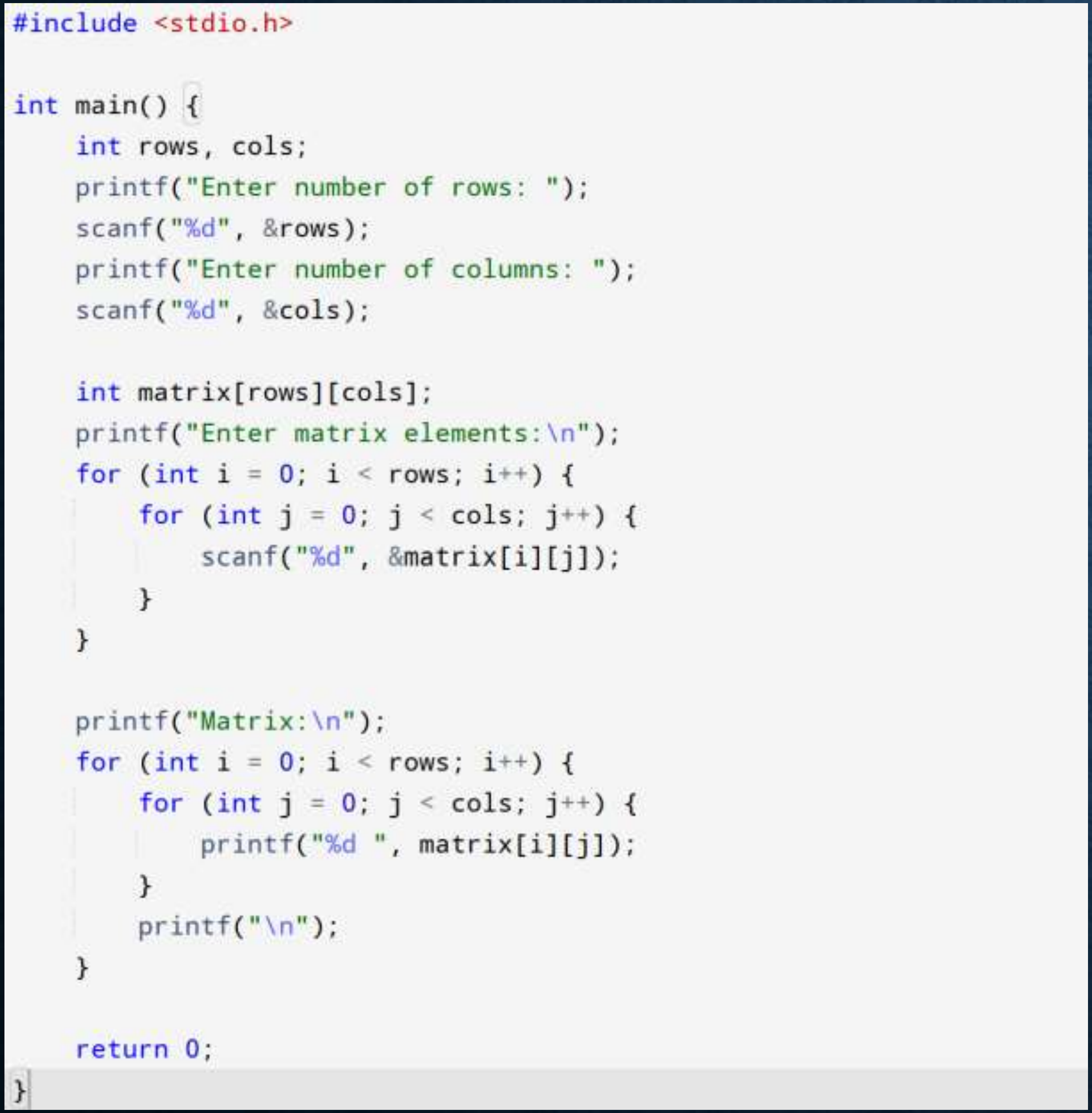
```
#include <stdio.h>

int main() {
    int matrix[2][3] = { {10,20,30}, {40,50,60} };
    int i, j;
    for(i =0;i <2; i++)
    {
    for(j =0; j <3; j++)
    {
        printf("%d ", matrix[i][j]);

    }
    printf("\n") ;
    }
    return 0;
}
```



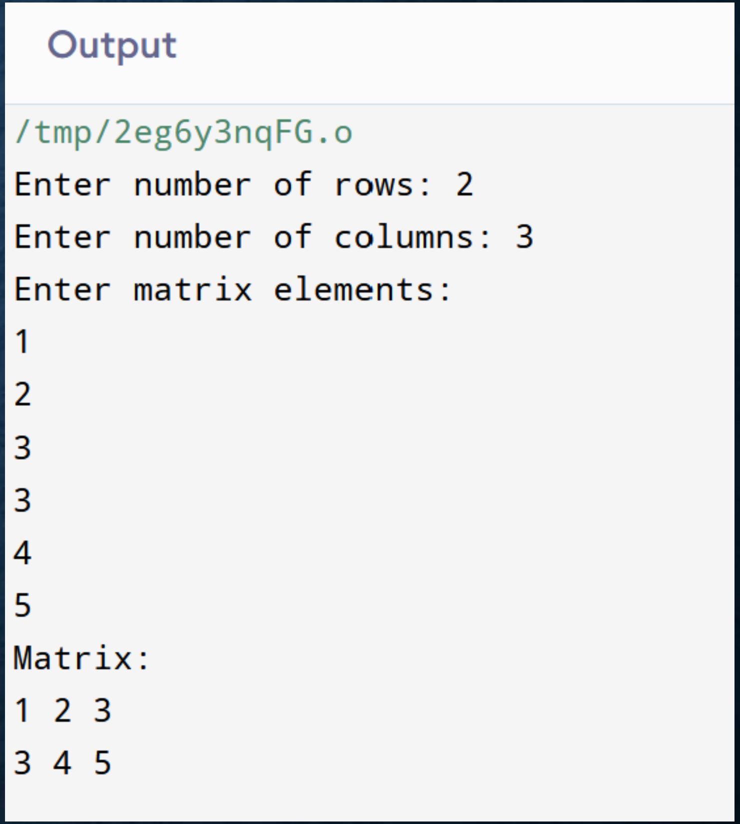
Output

```
/tmp/HAPw65FRtZ.o
10 20 30
40 50 60
```

CREATE AND DISPLAY MATRIX



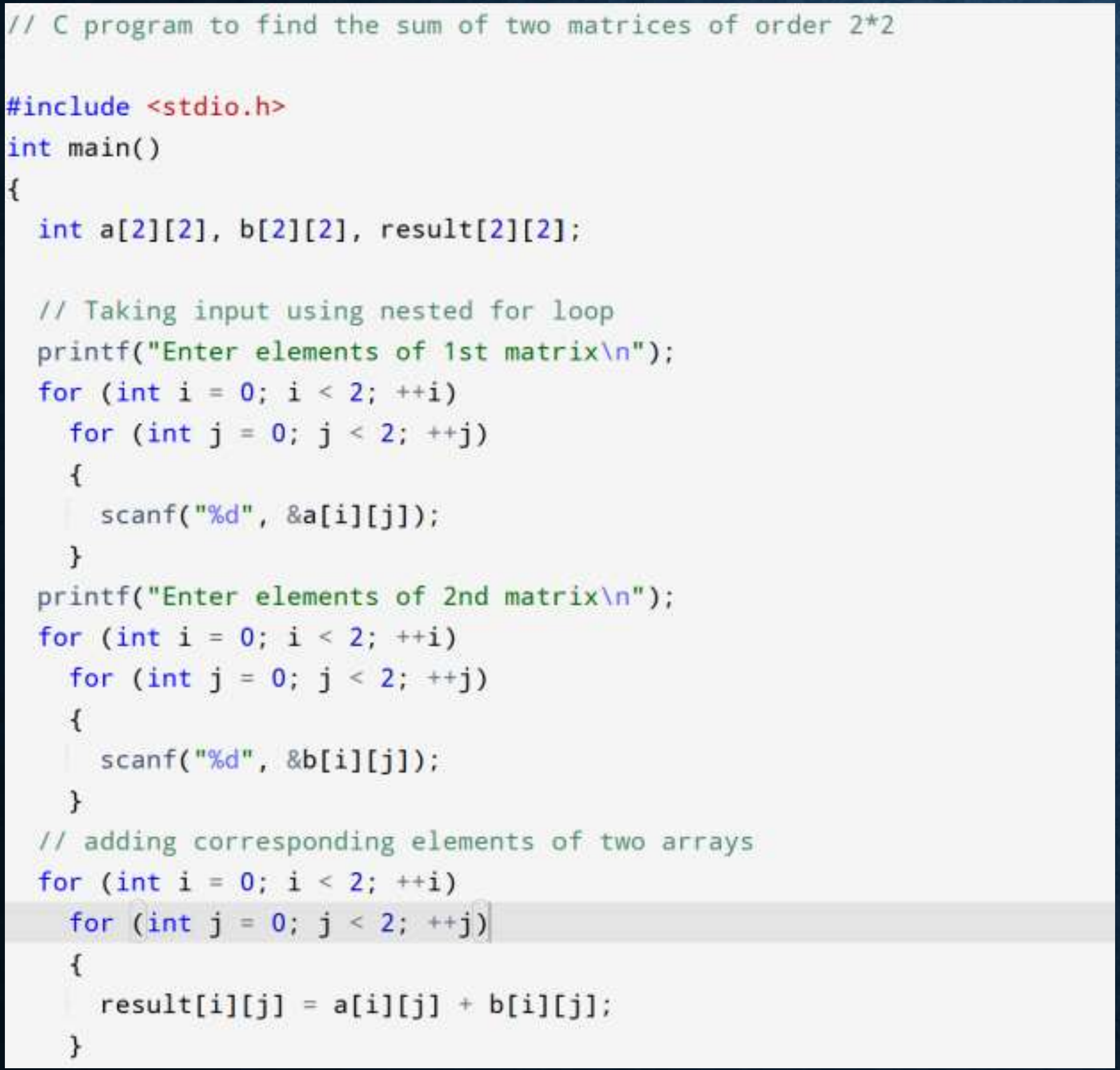
```c
#include <stdio.h>

int main() {
    int rows, cols;
    printf("Enter number of rows: ");
    scanf("%d", &rows);
    printf("Enter number of columns: ");
    scanf("%d", &cols);

    int matrix[rows][cols];
    printf("Enter matrix elements:\n");
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            scanf("%d", &matrix[i][j]);
        }
    }

    printf("Matrix:\n");
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            printf("%d ", matrix[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```



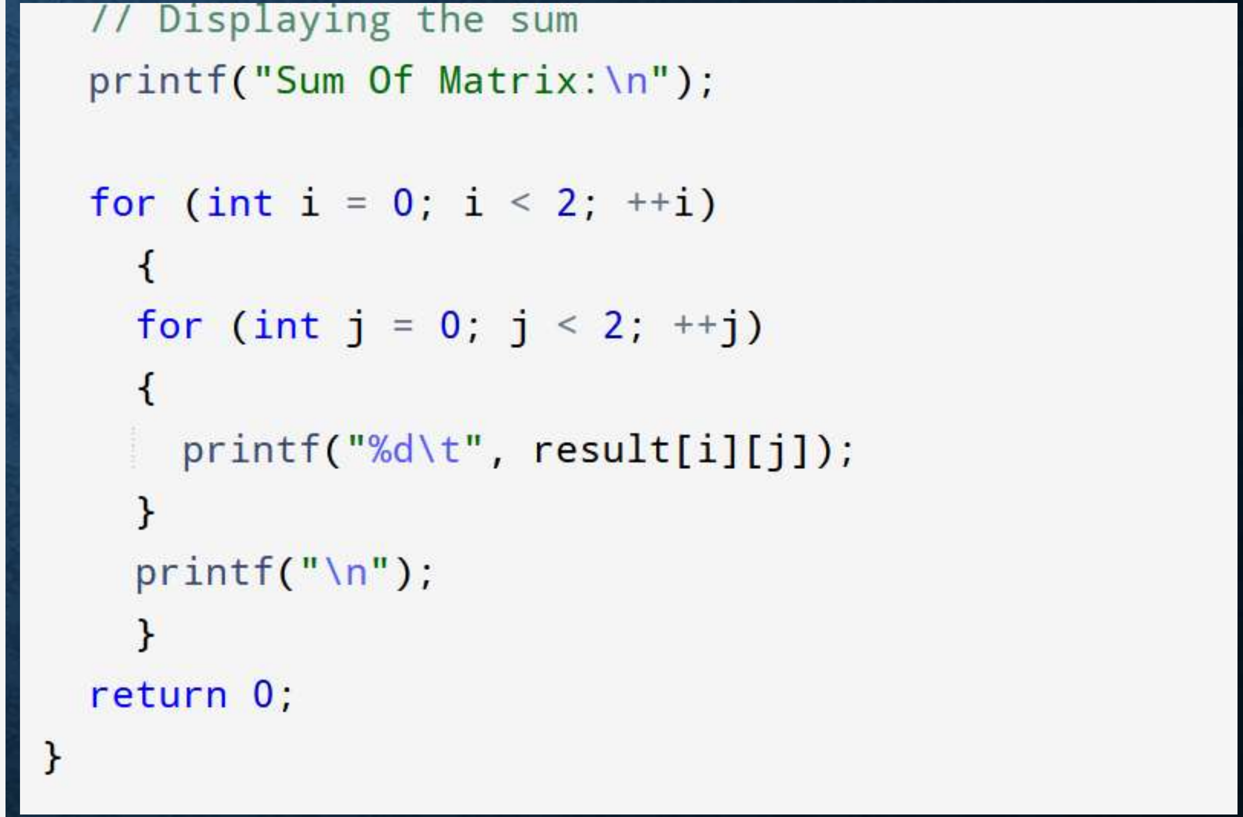
Output

```
/tmp/2eg6y3nqFG.o
Enter number of rows: 2
Enter number of columns: 3
Enter matrix elements:
1
2
3
3
4
5
Matrix:
1 2 3
3 4 5
```

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



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

#include <stdio.h>
int main()
{
  int a[2][2], b[2][2], result[2][2];

  // Taking input using nested for loop
  printf("Enter elements of 1st matrix\n");
  for (int i = 0; i < 2; ++i)
    for (int j = 0; j < 2; ++j)
    {
      scanf("%d", &a[i][j]);
    }
  printf("Enter elements of 2nd matrix\n");
  for (int i = 0; i < 2; ++i)
    for (int j = 0; j < 2; ++j)
    {
      scanf("%d", &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];
    }
```



```c
  // Displaying the sum
  printf("Sum Of Matrix:\n");

  for (int i = 0; i < 2; ++i)
    {
    for (int j = 0; j < 2; ++j)
    {
      printf("%d\t", result[i][j]);
    }
    printf("\n");
    }
  return 0;
}
```

POINTERS

- We can get the memory address of a variable with the reference operator &:

Eg:

```
int Age = 27; // an int variable

printf("%d", Age);  // Outputs the value of Age (27)

printf("%p", &Age); // Outputs the memory address of Age
```

- A pointer is a variable that stores the memory address of another variable as its value.
- A pointer variable points to a data type (like int) of the same type, and is created with the * operator.
- The address of the variable you are working with is assigned to the pointer:

- **Pointer Syntax**
- Here is how we can declare pointers.

```
int* p;
```

Here, we have declared a pointer p of int type.

Assigning addresses to Pointers:

```
int* pc, c;
c = 5;
pc = &c;
```

Here, 5 is assigned to the c variable. And, the address of c is assigned to the pc pointer.

GET VALUE OF THING POINTED BY POINTERS

- To get the value of the thing pointed by the pointers, we use the * operator. For example:

```
int* pc, c;
c = 5;
pc = &c;
printf("%d", *pc);   // Output: 5
```

- Here, the address of c is assigned to the pc pointer. To get the value stored in that address, we used *pc.

- Note: In the above example, pc is a pointer, not *pc. You cannot and should not do something like *pc = &c;

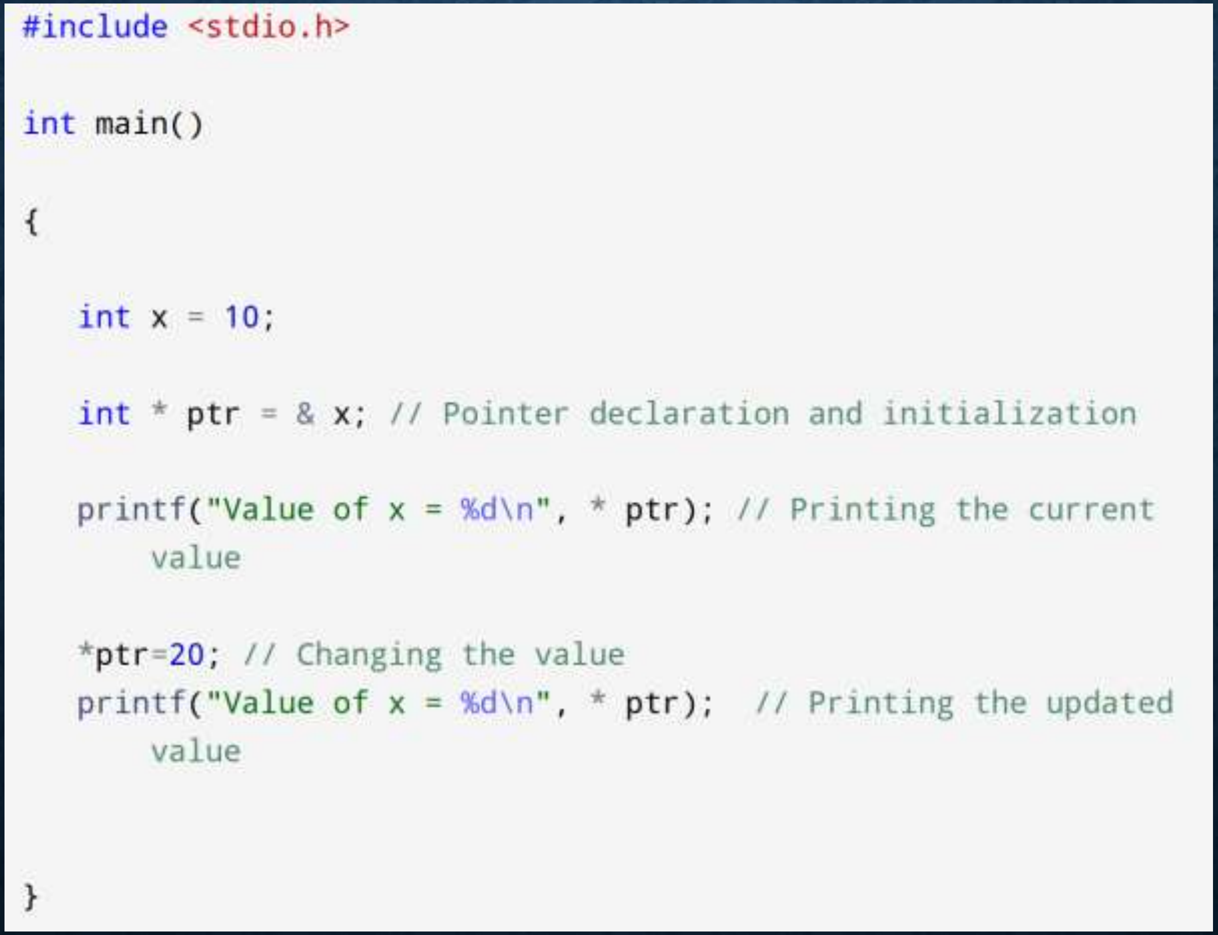
- * is called the dereference operator (when working with pointers). It operates on a pointer and gives the value stored in that pointer.

C – IMPORTANCE OF POINTERS

- Pointers are one of the things that make C stand out from other programming languages, like Python and Java.
- With pointers, you can access and modify the data located in the memory, pass the data efficiently between the functions, and create dynamic data structures like linked lists, trees, and graphs.
- They are important in C, because they allow us to manipulate the data in the computer's memory. This can reduce the code and improve the performance. And sometimes you even have to use pointers, for example when working with files and memory management.
- **But be careful**; pointers must be handled with care, since it is possible to damage data stored in other memory addresses.

C – ACCESS AND MANIPULATE VALUES USING POINTER

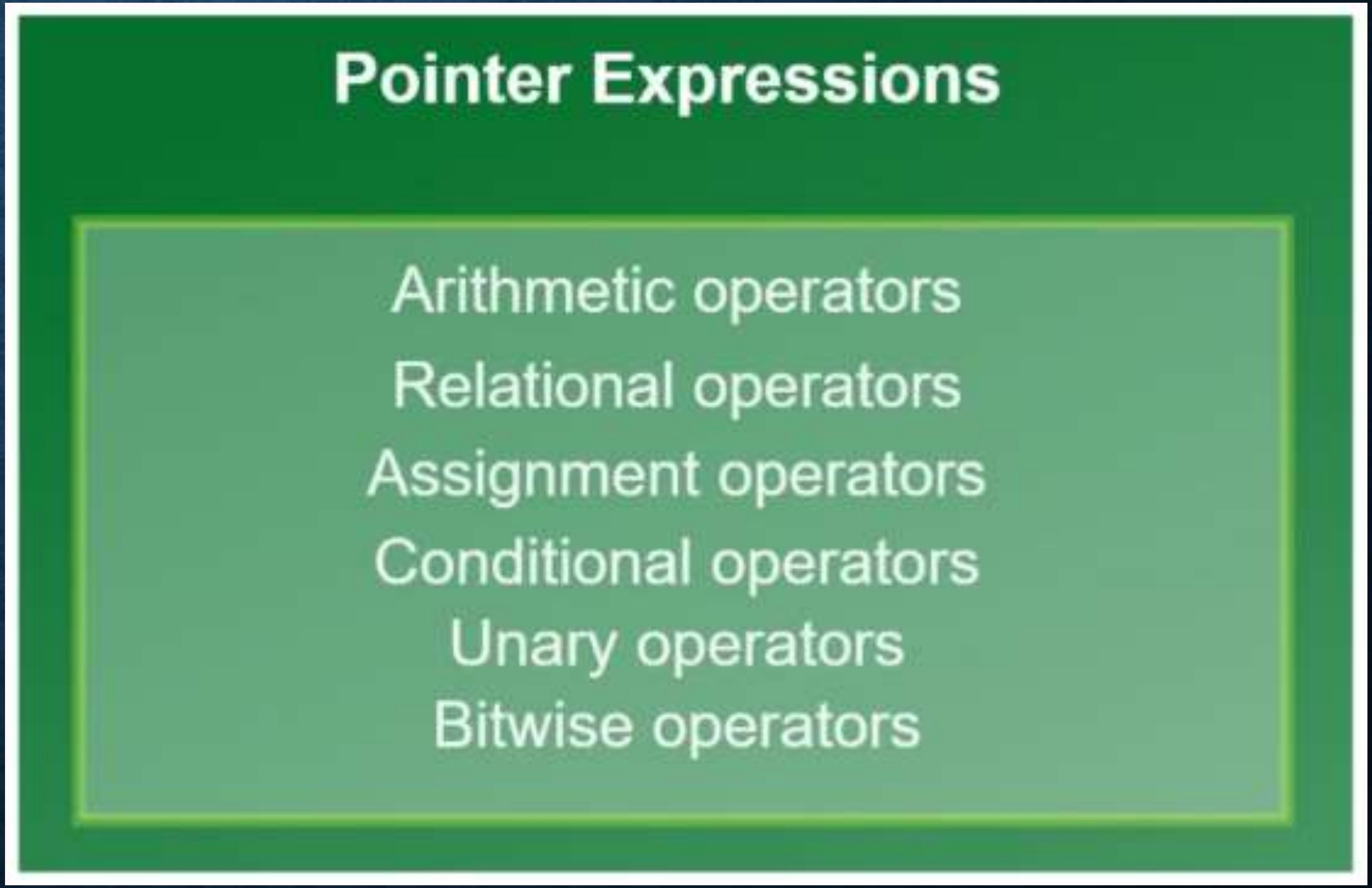
- The value of the variable which is pointed by a pointer can be accessed and manipulated by using the pointer variable.
- You need to use the asterisk (*) sign with the pointer variable to access and manipulate the variable's value.



```
#include <stdio.h>

int main()

{

   int x = 10;

   int * ptr = & x; // Pointer declaration and initialization

   printf("Value of x = %d\n", * ptr); // Printing the current
       value

   *ptr=20; // Changing the value
   printf("Value of x = %d\n", * ptr);  // Printing the updated
       value


}
```

POINTER EXPRESSIONS

Just like any other variable, these operations can be also **performed on pointer variables**.



Pointer Expressions

Arithmetic operators
Relational operators
Assignment operators
Conditional operators
Unary operators
Bitwise operators

ARITHMETIC OPERATIONS USING POINTERS

- We can add an integer or subtract an integer using a pointer pointing to that integer variable.
- Operations possible (Addition, subtraction, multiplication, division & modulo)

Examples:

While performing division, make sure you put a blank space between ‘/’ and ‘*’ of the pointer as together it would make a multi-line comment(‘/*’).

```
*ptr1 + *ptr2
*ptr1 * *ptr2
*ptr1 + *ptr2 - *ptr3
```

- Can also directly perform arithmetic expressions on integers by dereferencing pointers.

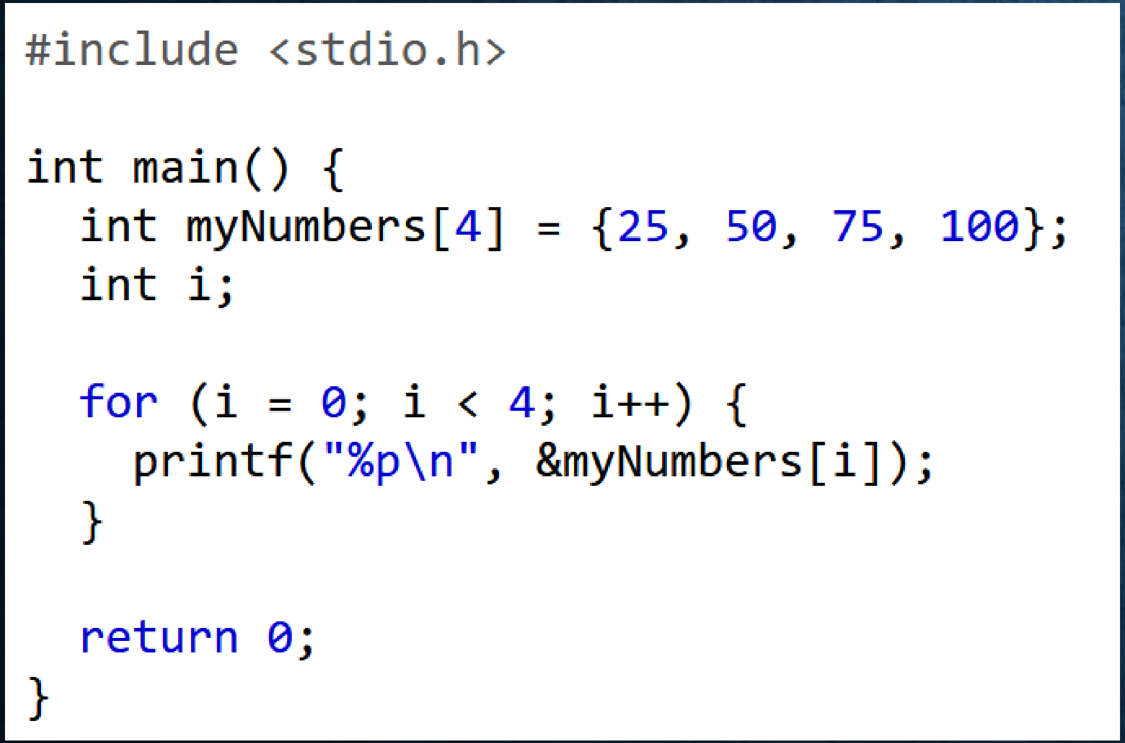
***p1 + 10, *p2 - 5, *p1 - *p2 + 10, *p1/2**

Similarly we can write expressions using relational, conditonal, unary, assignment, bitwise operators also.

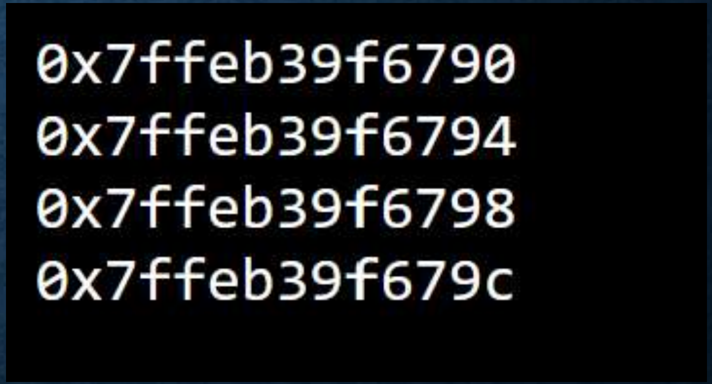
POINTERS AND ARRAYS

- You can also use pointers to access arrays.

Example



```c
#include <stdio.h>

int main() {
  int myNumbers[4] = {25, 50, 75, 100};
  int i;

  for (i = 0; i < 4; i++) {
    printf("%p\n", &myNumbers[i]);
  }

  return 0;
}
```



```
0x7ffeb39f6790
0x7ffeb39f6794
0x7ffeb39f6798
0x7ffeb39f679c
```

Note that the last number of each of the elements' memory address is different, with an addition of 4.

POINTERS AND ARRAYS

How Are Pointers Related to Arrays

The **name of an array**, is actually a **pointer** to the **first element** of the array.

```
int myNumbers[4] = {25, 50, 75, 100};
printf("%p\n", myNumbers);          // 0x7ffe70f9d8f0
printf("%p\n", &myNumbers[0] ;   // 0x7ffe70f9d8f0
printf("%d", *myNumbers);       //25
```

Since myNumbers is a pointer to the first element in myNumbers, you can use the * operator to access it:

To access the rest of the elements in myNumbers, you can increment the pointer/array (+1, +2, etc):

```
printf("%d\n", *(myNumbers + 1));   // 50
printf("%d", *(myNumbers + 2));      // 75
```

POINTERS AND ARRAYS

How Are Pointers Related to Arrays

We can also use **loops** to access the array elements

```
int myNumbers[4] = {25, 50, 75, 100};
int *ptr = myNumbers;
int i;

for (i = 0; i < 4; i++)
 {
  printf("%d\t", *(ptr + i));    //25   50   75
100
}
```

We can update the value of array elements with pointers:

```
*myNumbers = 13; //1st
element
*(myNumbers +1) = 17; //2nd
```

PROGRAMS ON ARRAYS AND POINTERS IN C

With Code and Output Examples

1. ACCESSING ARRAY ELEMENTS USING POINTERS

Program:

```
#include <stdio.h>
int main(){
  int arr[3] = {10,20,30};
  int *ptr = arr;
  for(int i=0;i<3;i++){
    printf("%d ", *(ptr+i));
  }
  return 0;
}
```

Output:

```
10 20 30
```

POINTER TO STRUCTURE IN C

- You can use pointers with structs to make your code more efficient, especially when passing structs to functions or changing their values.
- To use a pointer to a struct, just add the * symbol, like you would with other data types.
- To access its members, you must use the -> operator instead of the dot . syntax:

EXAMPLE

```
// Define a struct
struct Car {
char brand[50];
int year;
};
int main() {
struct Car car = {"Toyota", 2020};
// Declare a pointer to the struct
struct Car *ptr = &car;
// Access members using the -> operator
printf("Brand: %s\n", ptr->brand);
printf("Year: %d\n", ptr->year);
return 0;
}
```

2. POINTER TO STRUCTURE

Program:

```
struct Student {
   int id;
   char name[20];
};
int main(){
   struct Student s1 = {1, "John"};
   struct Student *ptr = &s1;
   printf("%d %s", ptr->id, ptr->name);
   return 0;
}
```

Output:

```
1 John
```

3. ARRAY ELEMENTS USING POINTER

Program:

```
#include <stdio.h>
int main(){
  int arr[5] = {1,2,3,4,5};
  int *ptr = arr;
  int sum=0;
  for(int i=0;i<5;i++){
    sum += *(ptr+i);
  }
  printf("Sum = %d", sum);
  return 0;
}
```

4. POINTER ARITHMETIC

Program:

```
#include <stdio.h>
int main(){
  int arr[3] = {5,10,15};
  int *ptr = arr;
  printf("%d\n", *ptr);
  printf("%d\n", *(ptr+1));
  printf("%d\n", *(ptr+2));
  return 0;
}
```

Output:

```
5
10
15
```