CS 332/532 Systems Programming

Lecture 4

- Arrays, Pointers -

Professor : Mahmut Unan – UAB CS

Agenda

- Arrays
- 2D Arrays
- Pointers

Arrays

- **One-Dimensional Arrays**
  - An array is a data structure that contains a number of values, or else elements, of the same type.
  - Each element can be accessed by its position within the array
  - Always declare the array before your try to use them

    data_type array_name[number_of_elements];

    ```
    int sampleArray[100];
    float anotherArray[250];
    ```

predefined size

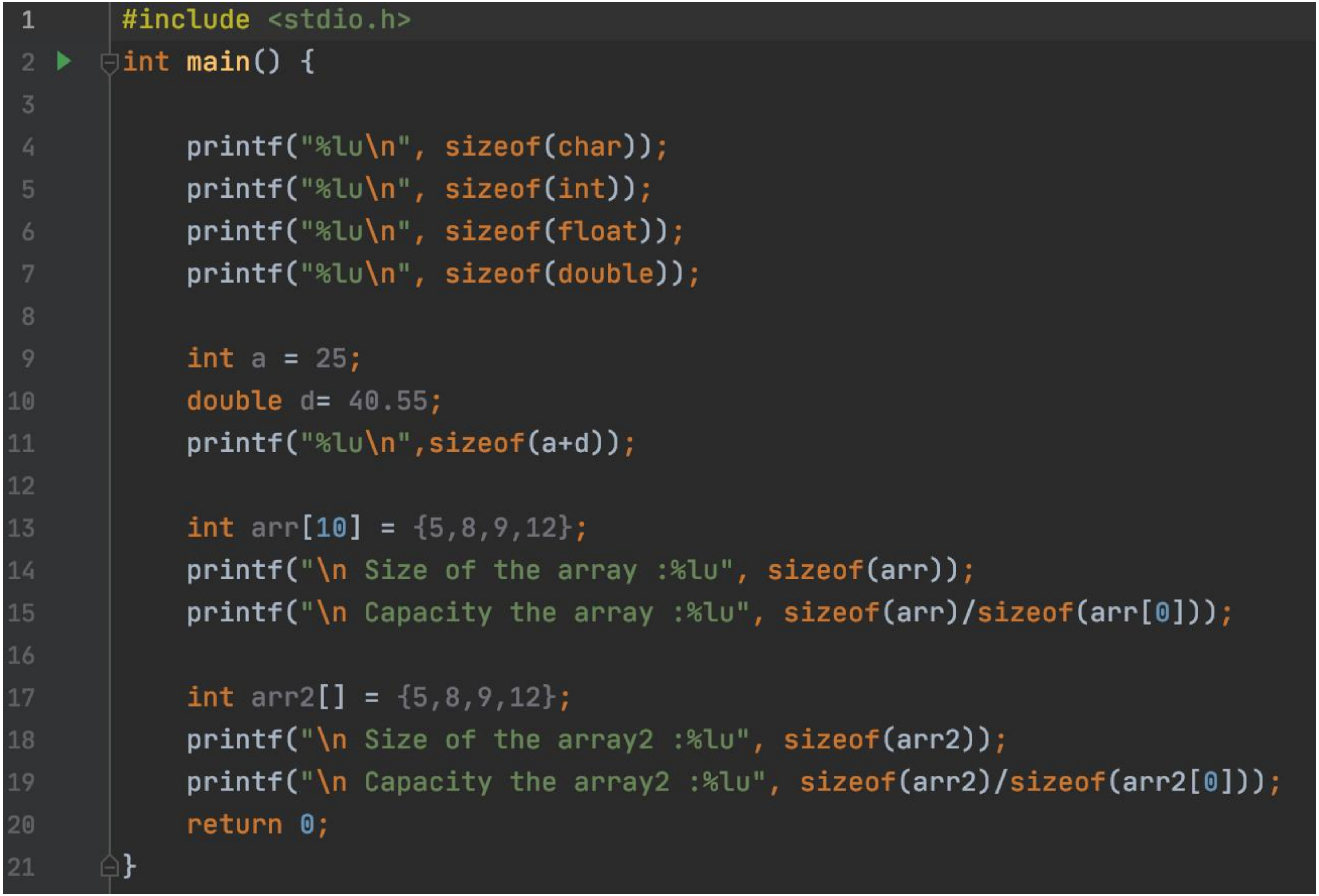
```
/* use macros */
#define ARRAY_SIZE 250
float sampleArray[ARRAY_SIZE];
```

✓

```
/* never use const */
const int array_size = 250;
float sampleArray(array_size)
/* this is not legal */
```

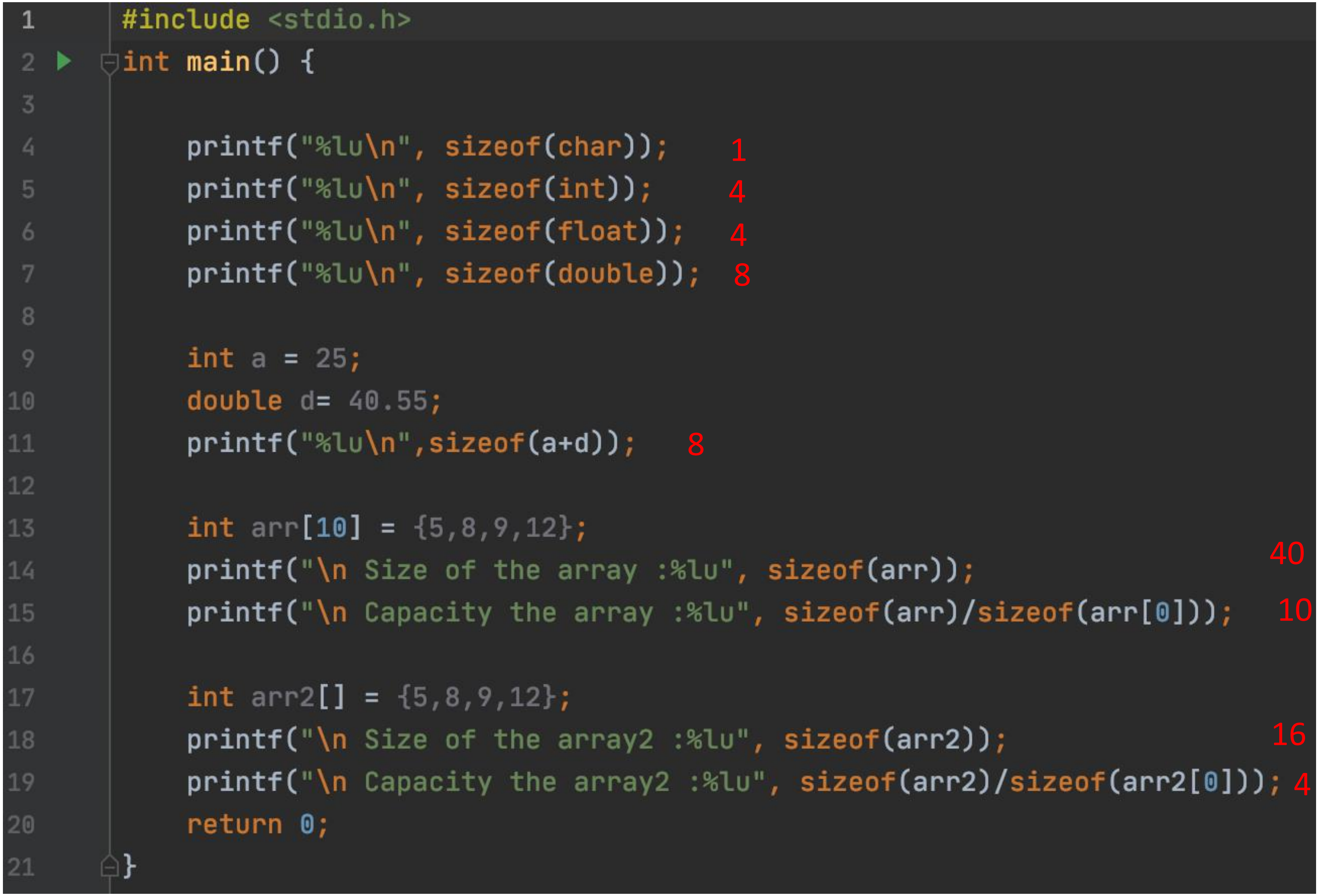
✗

sizeof()



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

    printf("%lu\n", sizeof(char));
    printf("%lu\n", sizeof(int));
    printf("%lu\n", sizeof(float));
    printf("%lu\n", sizeof(double));

    int a = 25;
    double d= 40.55;
    printf("%lu\n",sizeof(a+d));

    int arr[10] = {5,8,9,12};
    printf("\n Size of the array :%lu", sizeof(arr));
    printf("\n Capacity the array :%lu", sizeof(arr)/sizeof(arr[0]));

    int arr2[] = {5,8,9,12};
    printf("\n Size of the array2 :%lu", sizeof(arr2));
    printf("\n Capacity the array2 :%lu", sizeof(arr2)/sizeof(arr2[0]));
    return 0;
}
```

sizeof()

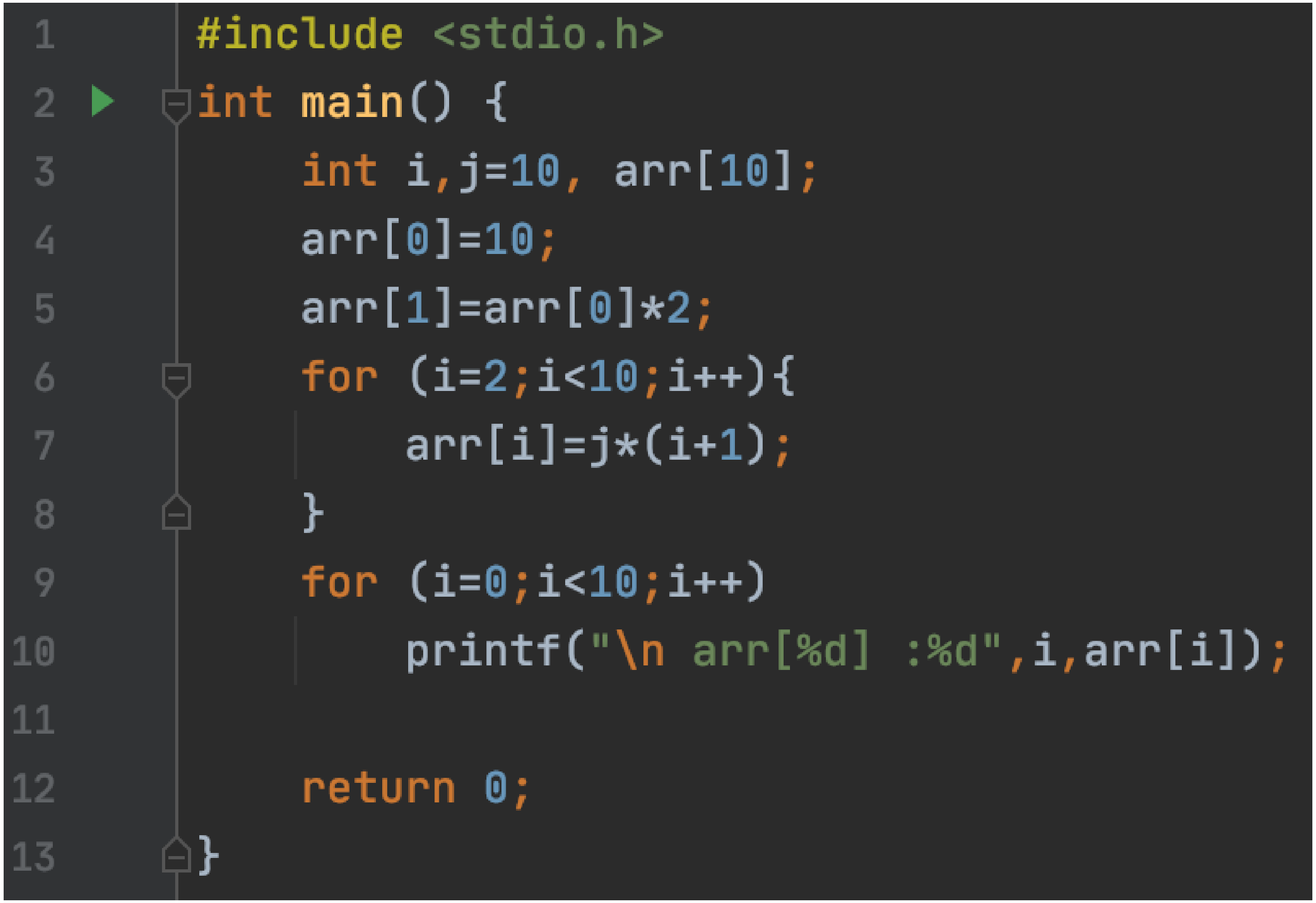


```
#include <stdio.h>
int main() {

    printf("%lu\n", sizeof(char));     1
    printf("%lu\n", sizeof(int));      4
    printf("%lu\n", sizeof(float));    4
    printf("%lu\n", sizeof(double));   8

    int a = 25;
    double d= 40.55;
    printf("%lu\n",sizeof(a+d));    8

    int arr[10] = {5,8,9,12};
    printf("\n Size of the array :%lu", sizeof(arr));                            40
    printf("\n Capacity the array :%lu", sizeof(arr)/sizeof(arr[0]));   10

    int arr2[] = {5,8,9,12};
    printf("\n Size of the array2 :%lu", sizeof(arr2));                          16
    printf("\n Capacity the array2 :%lu", sizeof(arr2)/sizeof(arr2[0])); 4
    return 0;
}
```

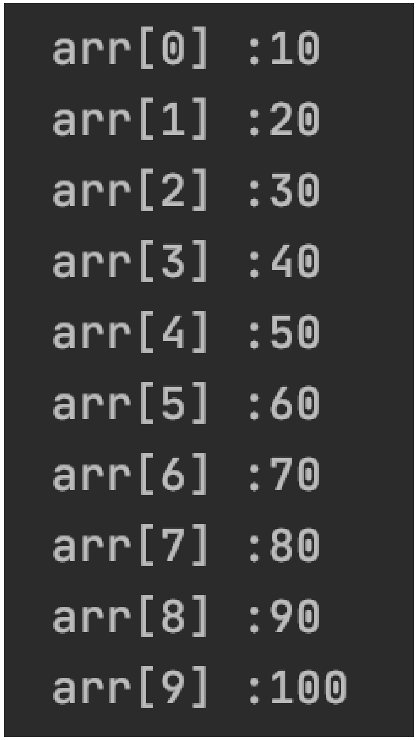
Initialize the Array

```
int arr[3]={10,20,30};
int arr2[10]={10,20};

int arr3[]={10,20,30,40};

/* be careful with the
following*/
const int arr4[] = {10,20,30,40}
```

Assign - Access elements



```c
#include <stdio.h>
int main() {
    int i,j=10, arr[10];
    arr[0]=10;
    arr[1]=arr[0]*2;
    for (i=2;i<10;i++){
        arr[i]=j*(i+1);
    }
    for (i=0;i<10;i++)
        printf("\n arr[%d] :%d",i,arr[i]);

    return 0;
}
```

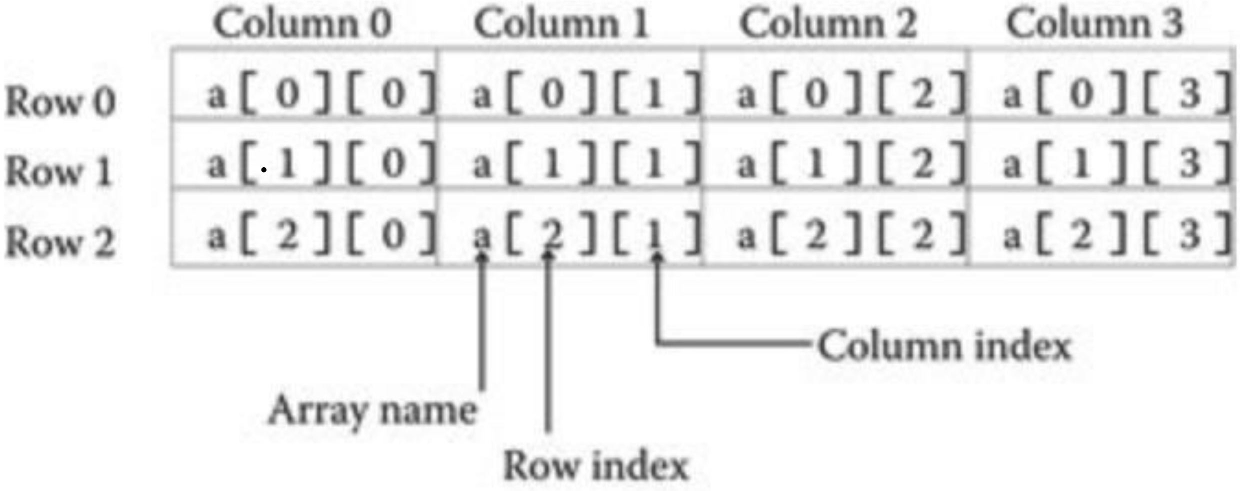


```
arr[0] :10
arr[1] :20
arr[2] :30
arr[3] :40
arr[4] :50
arr[5] :60
arr[6] :70
arr[7] :80
arr[8] :90
arr[9] :100
```

2D Arrays

- **data_type** array_name[number_of_rows][number_of_columns]

```
int a[3][4];
```



	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

initialize 2D array

```
int arr[3][3] = {{10, 20, 30},
{40, 50, 60}, {70, 80, 90}};
```

```
int arr[3][4] = {10, 20, 30, 40,
50, 60, 70, 80, 90};
```

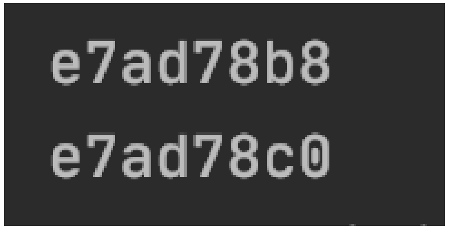
```
10 20 30 40
50 60 70 80
90 0   0  0
```

Pointers

```
int a = 5;
float arr[25];
```

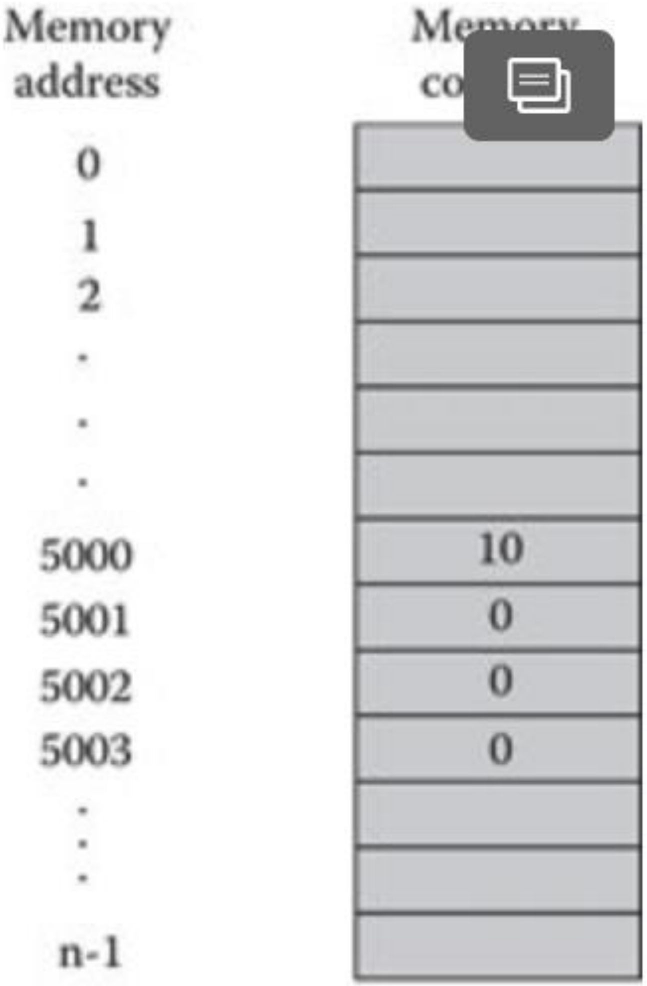
- /* To reach the memory location variables, arrays…etc use ampersand (&) operator */

```
printf(“\n %x”, &a);
printf(“\n %x”, &arr);
```



```
e7ad78b8
e7ad78c0
```

Memory Address



Memory address	Memory co
0	
1	
2	
.	
.	
.	
5000	10
5001	0
5002	0
5003	0
.	
.	
.	
n-1	

Pointers / 2

- How to store this address?
  - – we use the pointers
  - – Pointers is a variable whose value is the address of another variable
- Declare the pointer before you use it

```
data_type *pointer_name;
int *ptr, a, b, c;
int * ptr, a, b, c;
int* ptr, a, b, c;
```

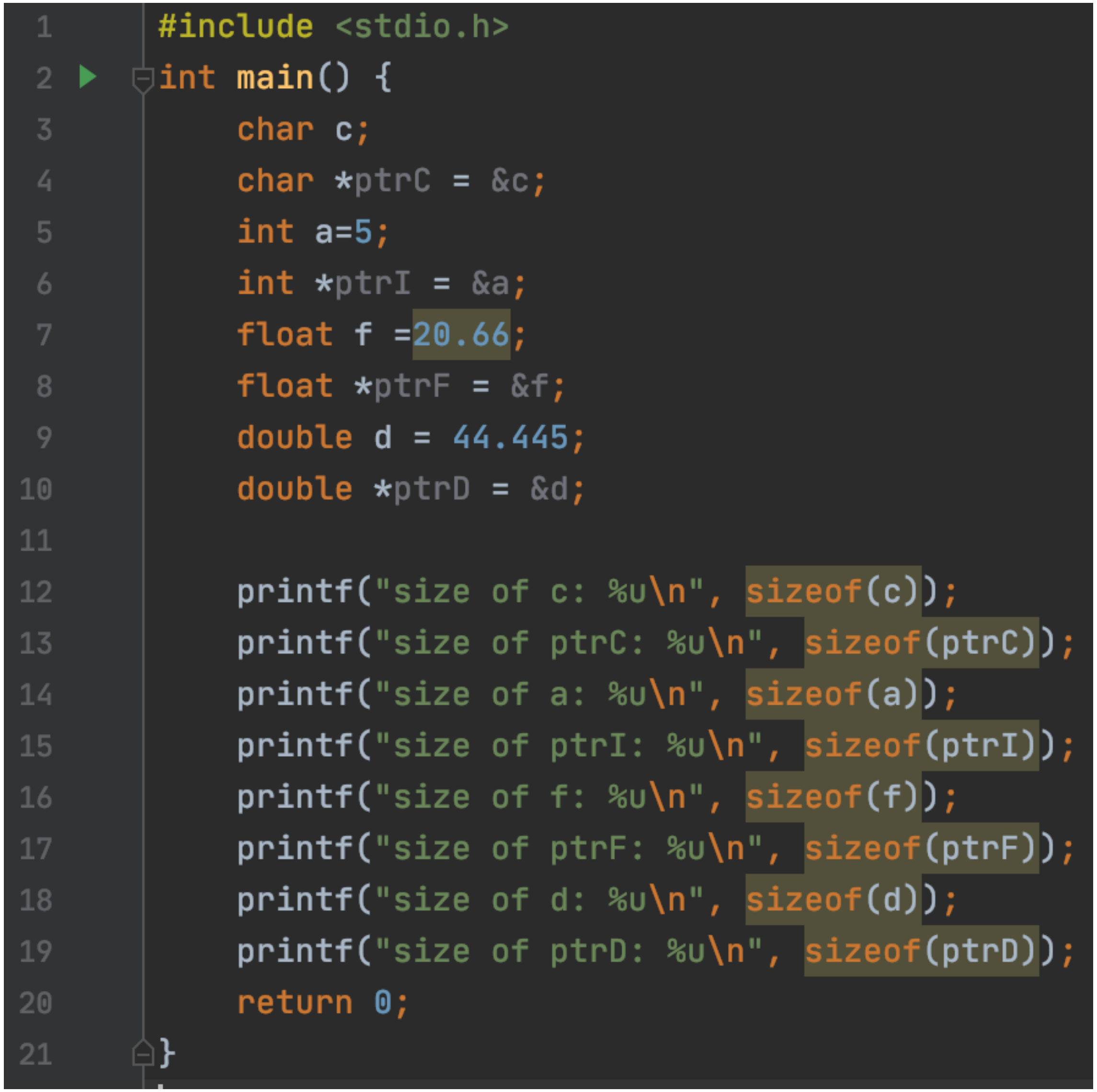
Pointers / 2

!!! Caution !!!!

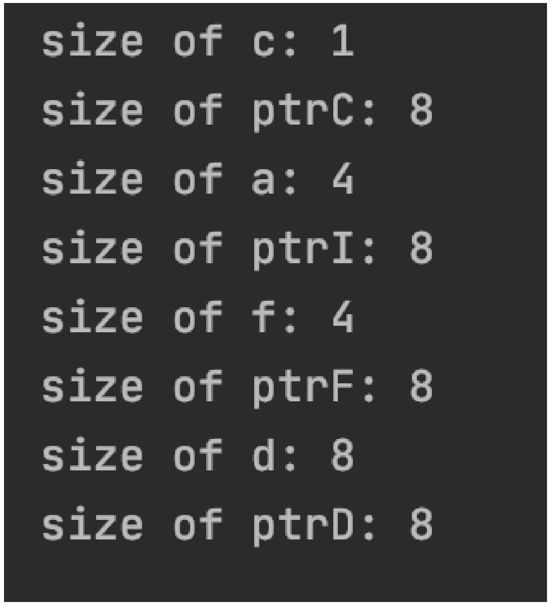
All three statements are correct and the result of each statement the “ptr” will be declared as the pointer but a,b, and c will be declared as int.
However; it is always better to use the first syntax

```
data_type *pointer_name;
int *ptr, a, b, c;
int * ptr, a, b, c;
int* ptr, a, b, c;
```

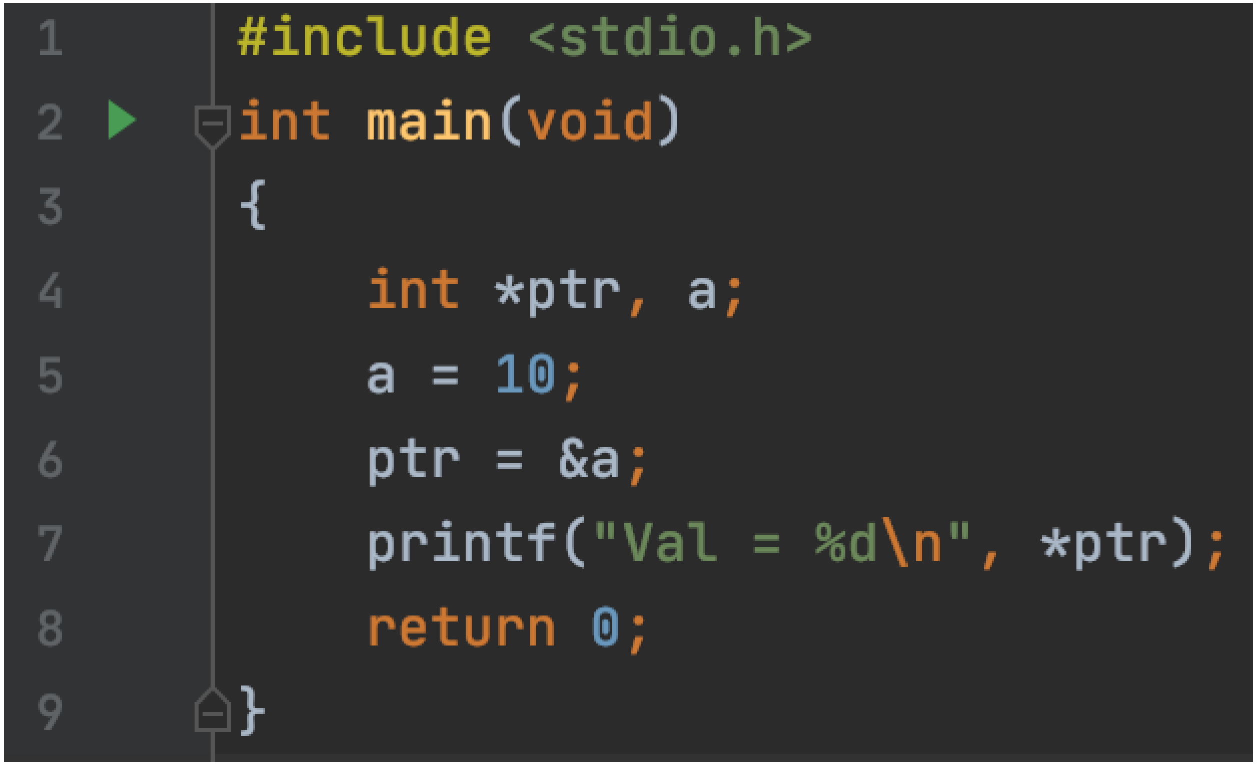
size of a pointer ???



```c
#include <stdio.h>
int main() {
    char c;
    char *ptrC = &c;
    int a=5;
    int *ptrI = &a;
    float f =20.66;
    float *ptrF = &f;
    double d = 44.445;
    double *ptrD = &d;

    printf("size of c: %u\n", sizeof(c));
    printf("size of ptrC: %u\n", sizeof(ptrC));
    printf("size of a: %u\n", sizeof(a));
    printf("size of ptrI: %u\n", sizeof(ptrI));
    printf("size of f: %u\n", sizeof(f));
    printf("size of ptrF: %u\n", sizeof(ptrF));
    printf("size of d: %u\n", sizeof(d));
    printf("size of ptrD: %u\n", sizeof(ptrD));
    return 0;
}
```



```
size of c: 1
size of ptrC: 8
size of a: 4
size of ptrI: 8
size of f: 4
size of ptrF: 8
size of d: 8
size of ptrD: 8
```

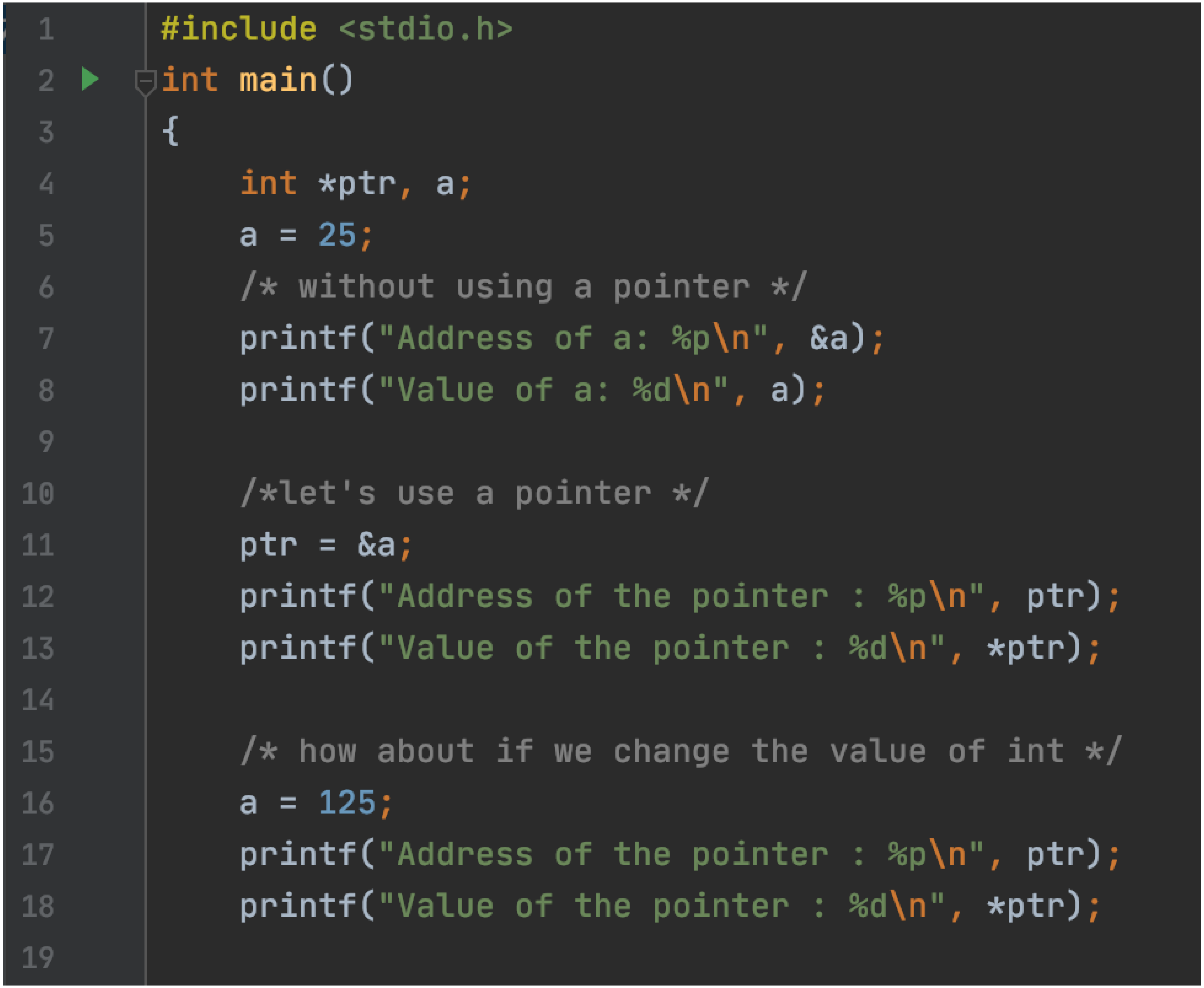


```c
#include <stdio.h>
int main(void)
{
    int *ptr, a;
    a = 10;
    ptr = &a;
    printf("Val = %d\n", *ptr);
    return 0;
}
```

10

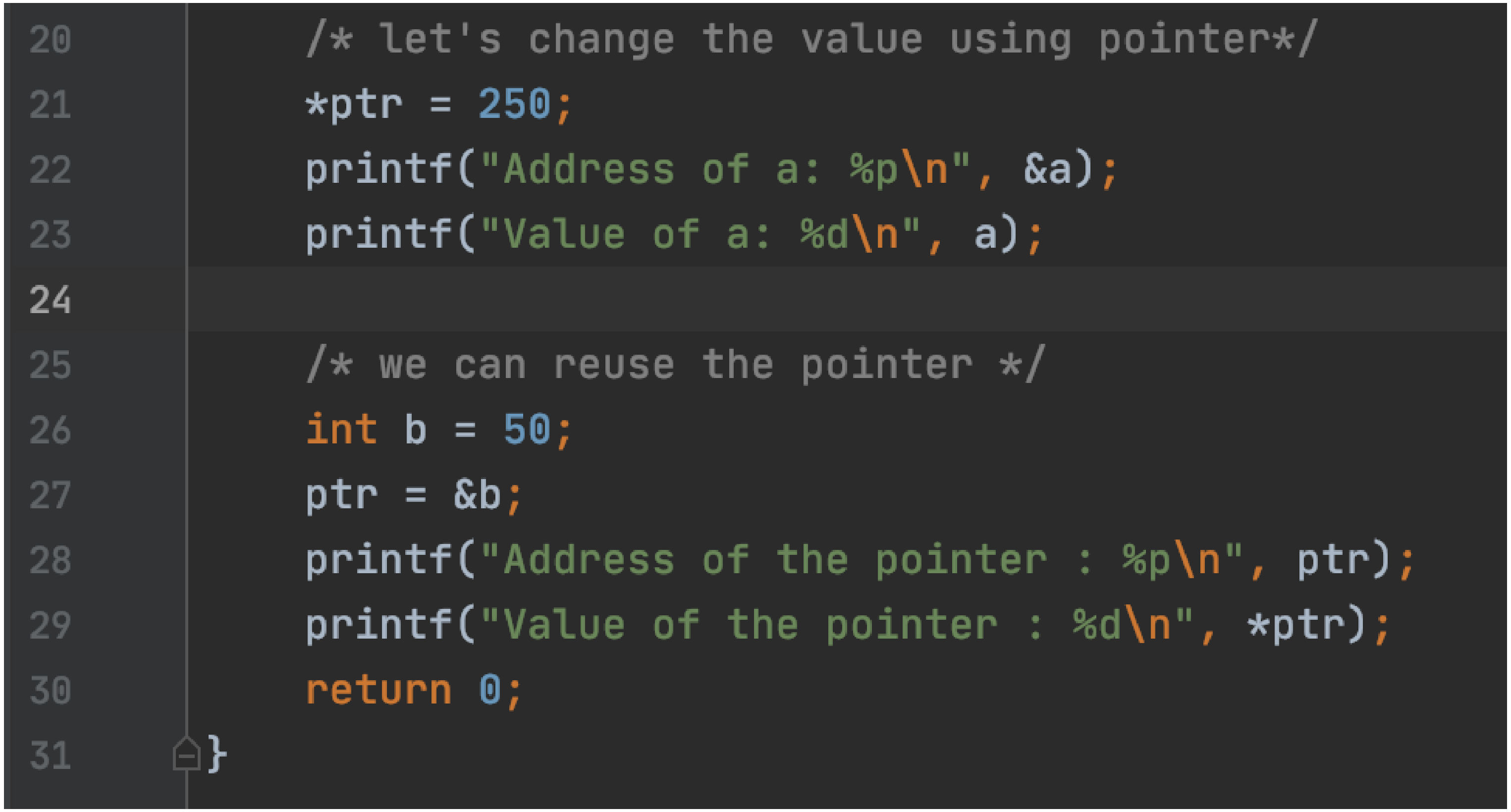
Always initialize the pointer before using it, otherwise you will get segmentation fault error

Example - page 1/2



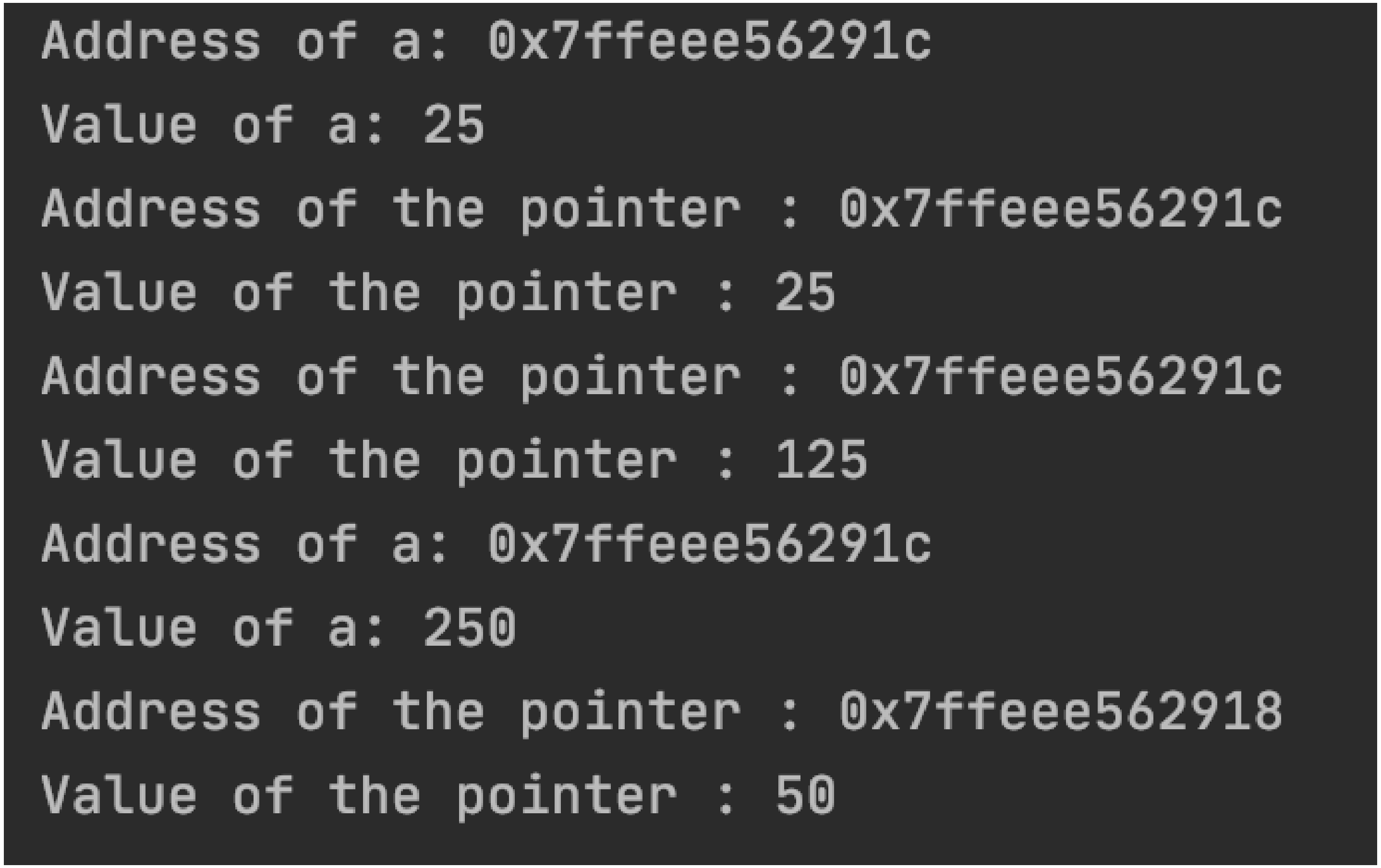
```c
#include <stdio.h>
int main()
{
    int *ptr, a;
    a = 25;
    /* without using a pointer */
    printf("Address of a: %p\n", &a);
    printf("Value of a: %d\n", a);

    /*let's use a pointer */
    ptr = &a;
    printf("Address of the pointer : %p\n", ptr);
    printf("Value of the pointer : %d\n", *ptr);

    /* how about if we change the value of int */
    a = 125;
    printf("Address of the pointer : %p\n", ptr);
    printf("Value of the pointer : %d\n", *ptr);

```

Example - page 2/2



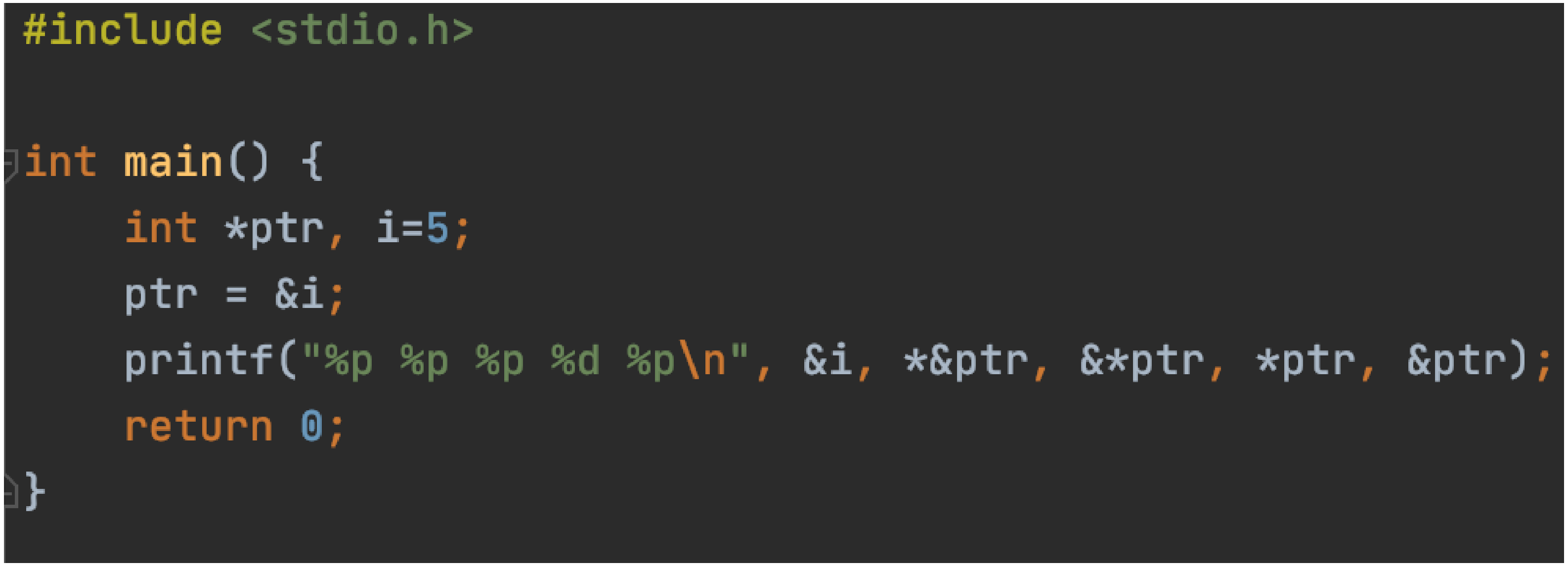
```c
    /* let's change the value using pointer*/
    *ptr = 250;
    printf("Address of a: %p\n", &a);
    printf("Value of a: %d\n", a);

    /* we can reuse the pointer */
    int b = 50;
    ptr = &b;
    printf("Address of the pointer : %p\n", ptr);
    printf("Value of the pointer : %d\n", *ptr);
    return 0;
}
```

Example - output

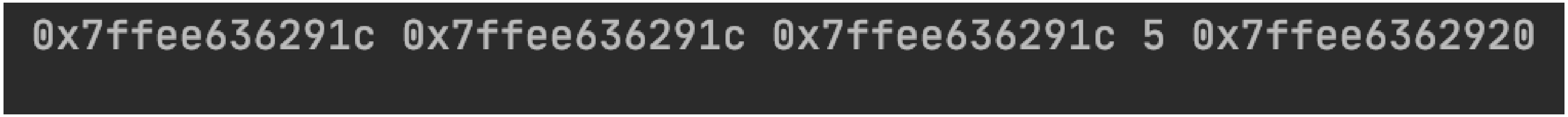


```
Address of a: 0x7ffeee56291c
Value of a: 25
Address of the pointer : 0x7ffeee56291c
Value of the pointer : 25
Address of the pointer : 0x7ffeee56291c
Value of the pointer : 125
Address of a: 0x7ffeee56291c
Value of a: 250
Address of the pointer : 0x7ffeee562918
Value of the pointer : 50
```

The * and & cancel each other when used together

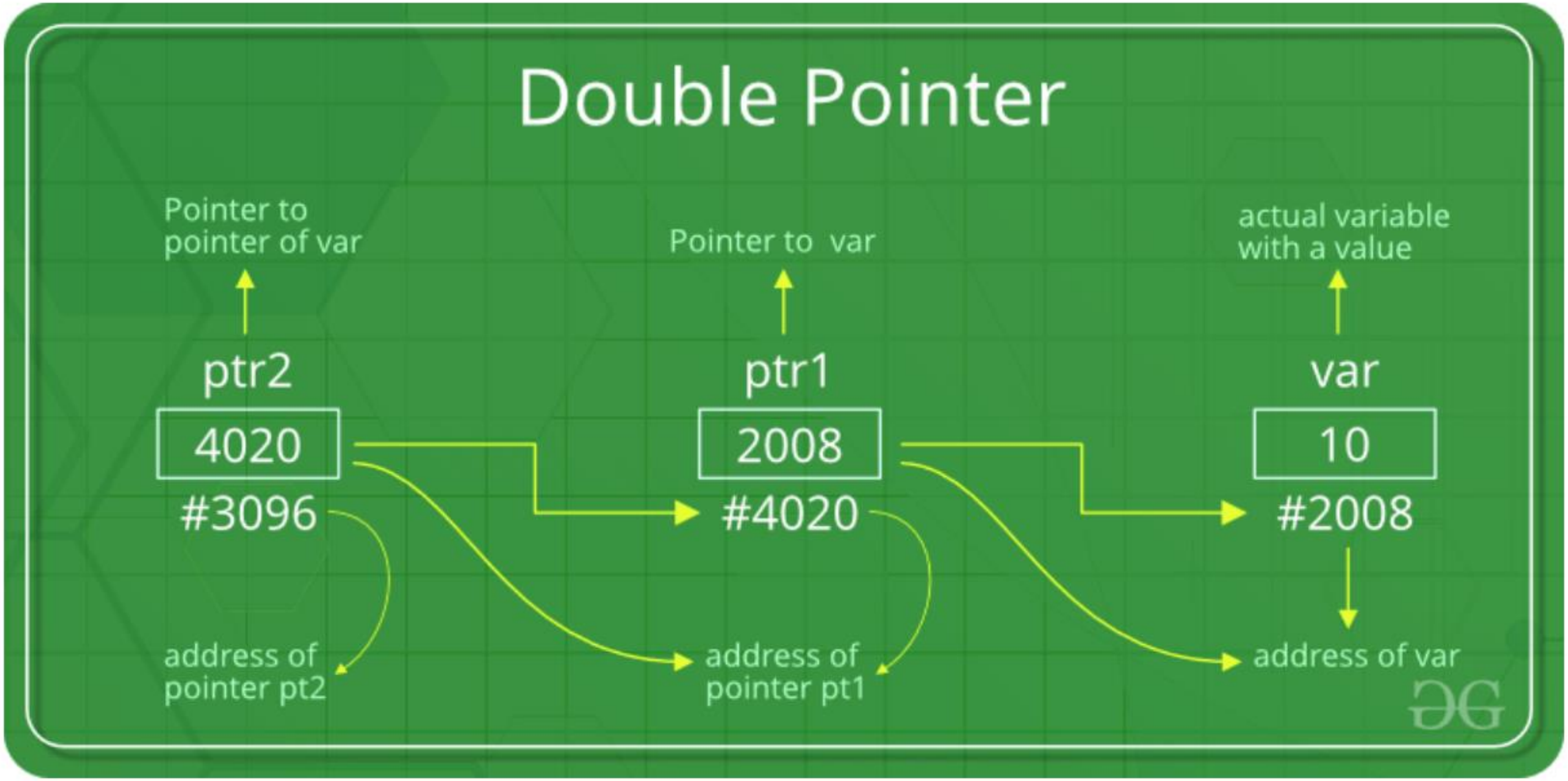


```c
#include <stdio.h>

int main() {
    int *ptr, i=5;
    ptr = &i;
    printf("%p %p %p %d %p\n", &i, *&ptr, &*ptr, *ptr, &ptr);
    return 0;
}
```

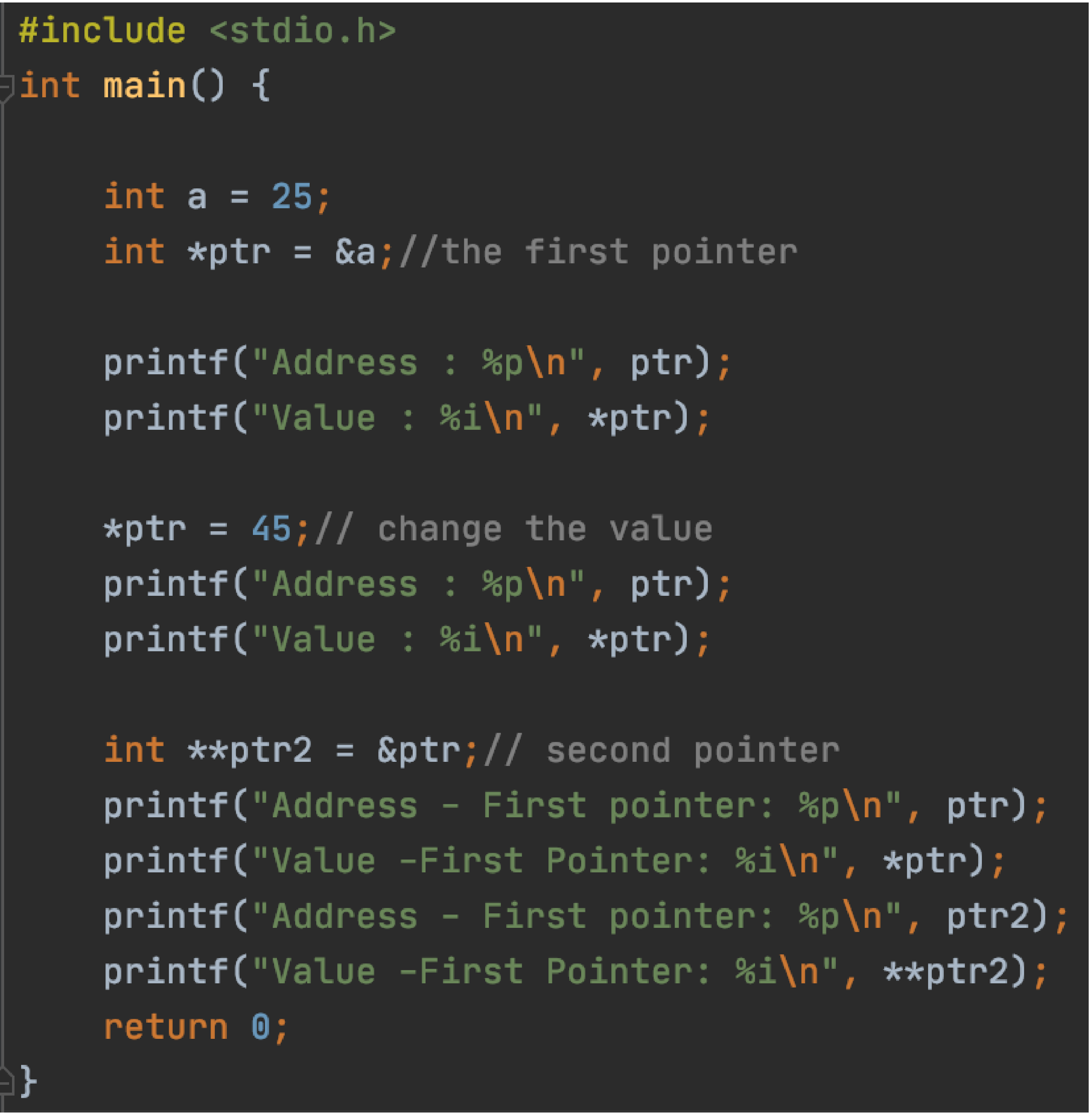


```
0x7ffee636291c 0x7ffee636291c 0x7ffee636291c 5 0x7ffee6362920
```

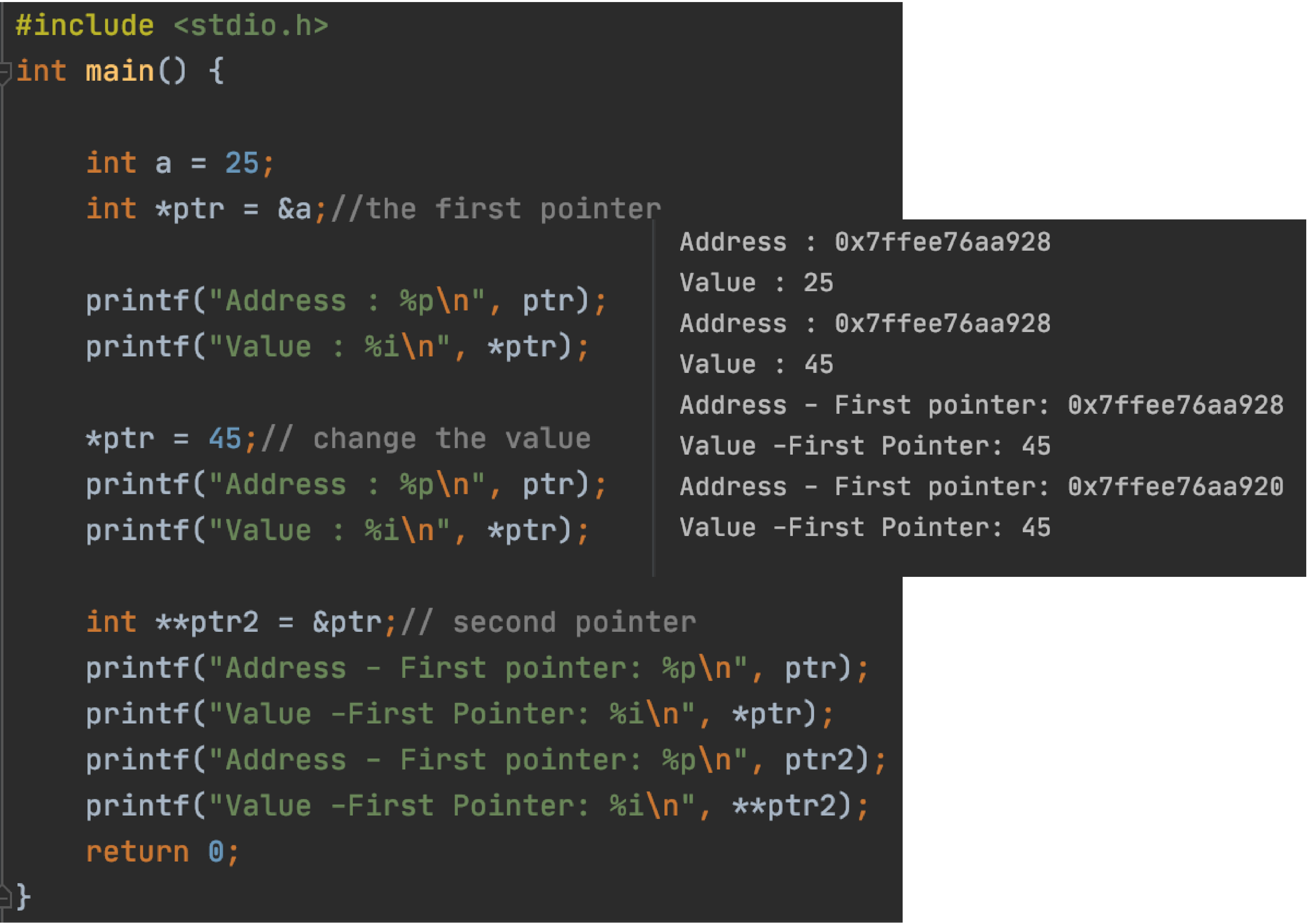
double pointer? even triple…



https://www.geeksforgeeks.org/double-pointer-pointer-pointer-c/



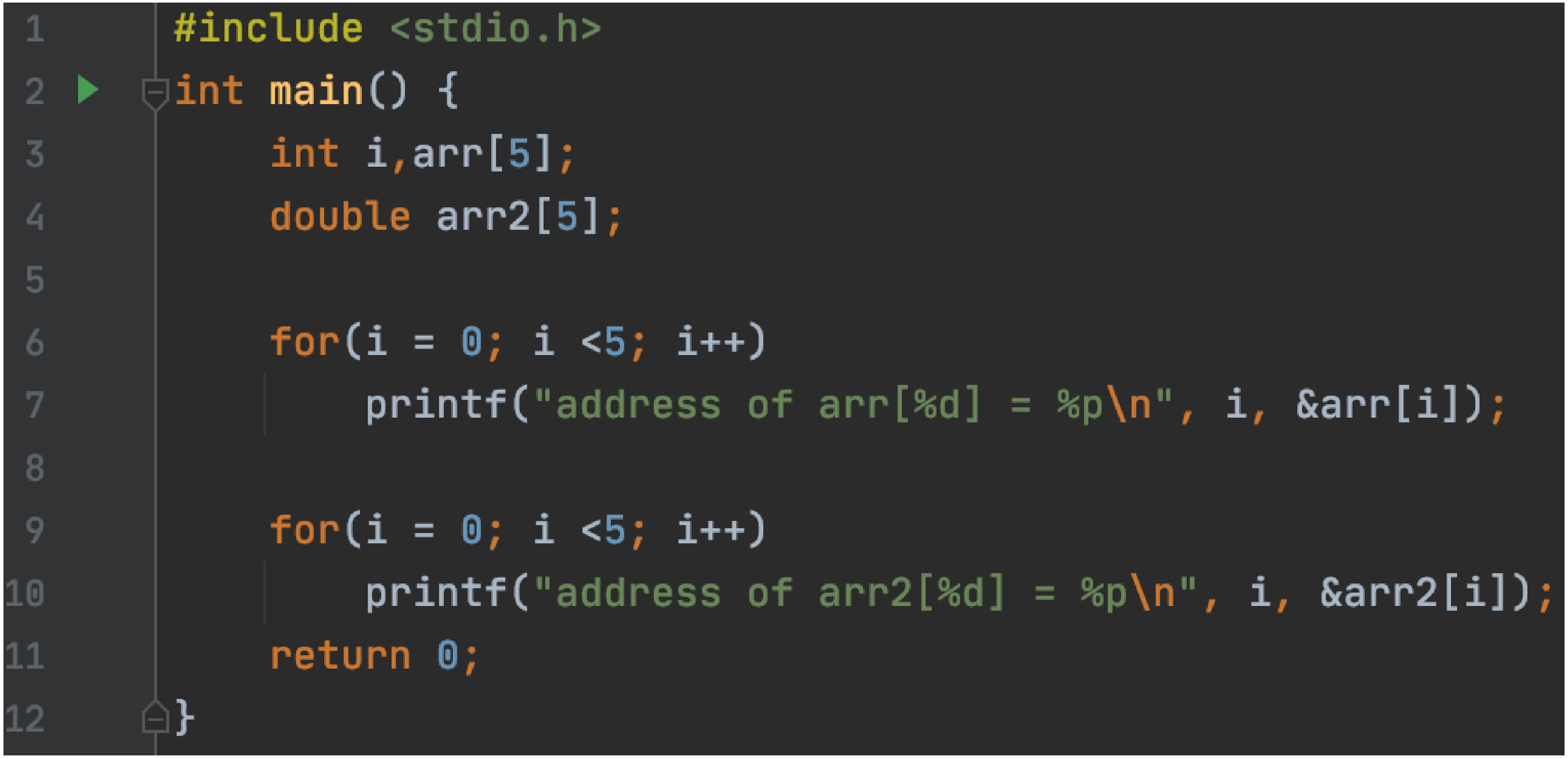
```c
#include <stdio.h>
int main() {

    int a = 25;
    int *ptr = &a;//the first pointer

    printf("Address : %p\n", ptr);
    printf("Value : %i\n", *ptr);

    *ptr = 45;// change the value
    printf("Address : %p\n", ptr);
    printf("Value : %i\n", *ptr);

    int **ptr2 = &ptr;// second pointer
    printf("Address - First pointer: %p\n", ptr);
    printf("Value -First Pointer: %i\n", *ptr);
    printf("Address - First pointer: %p\n", ptr2);
    printf("Value -First Pointer: %i\n", **ptr2);
    return 0;
}
```



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

    int a = 25;
    int *ptr = &a;//the first pointer

    printf("Address : %p\n", ptr);
    printf("Value : %i\n", *ptr);

    *ptr = 45;// change the value
    printf("Address : %p\n", ptr);
    printf("Value : %i\n", *ptr);

    int **ptr2 = &ptr;// second pointer
    printf("Address - First pointer: %p\n", ptr);
    printf("Value -First Pointer: %i\n", *ptr);
    printf("Address - First pointer: %p\n", ptr2);
    printf("Value -First Pointer: %i\n", **ptr2);
    return 0;
}
```

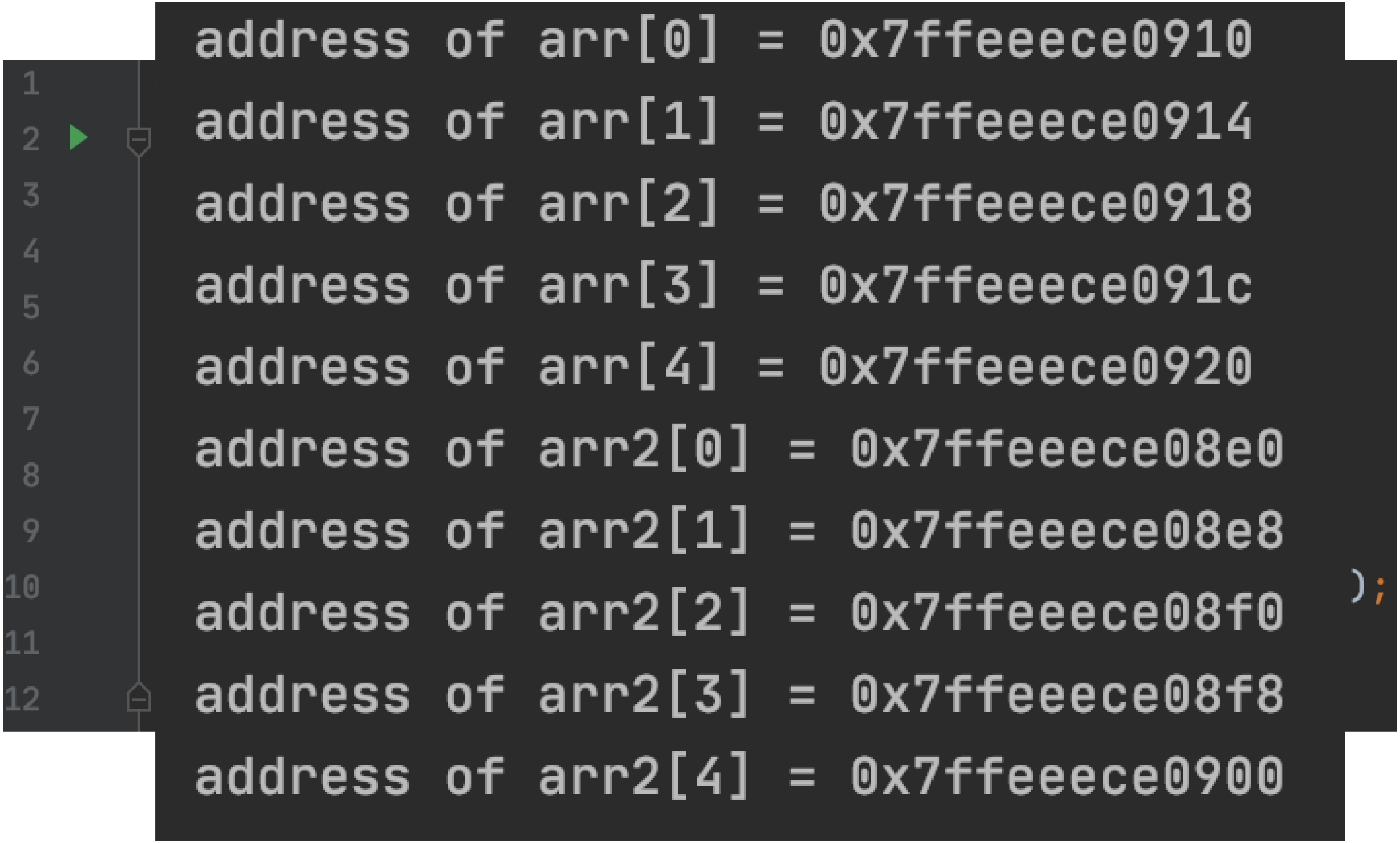
```
Address : 0x7ffee76aa928
Value : 25
Address : 0x7ffee76aa928
Value : 45
Address - First pointer: 0x7ffee76aa928
Value -First Pointer: 45
Address - First pointer: 0x7ffee76aa920
Value -First Pointer: 45
```

Arrays & Pointers



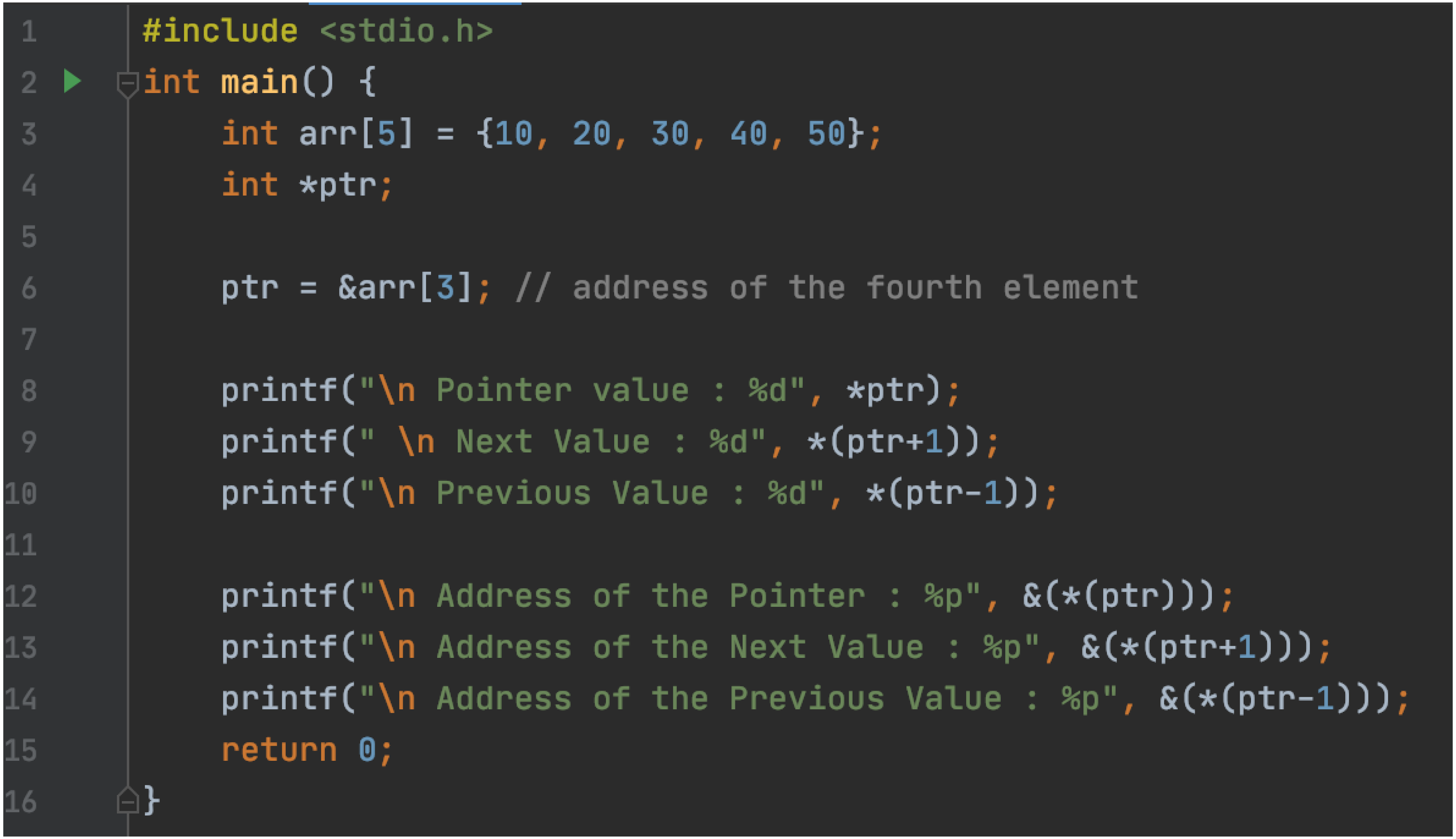
```c
#include <stdio.h>
int main() {
    int i,arr[5];
    double arr2[5];

    for(i = 0; i <5; i++)
        printf("address of arr[%d] = %p\n", i, &arr[i]);

    for(i = 0; i <5; i++)
        printf("address of arr2[%d] = %p\n", i, &arr2[i]);
    return 0;
}
```

Arrays & Pointers



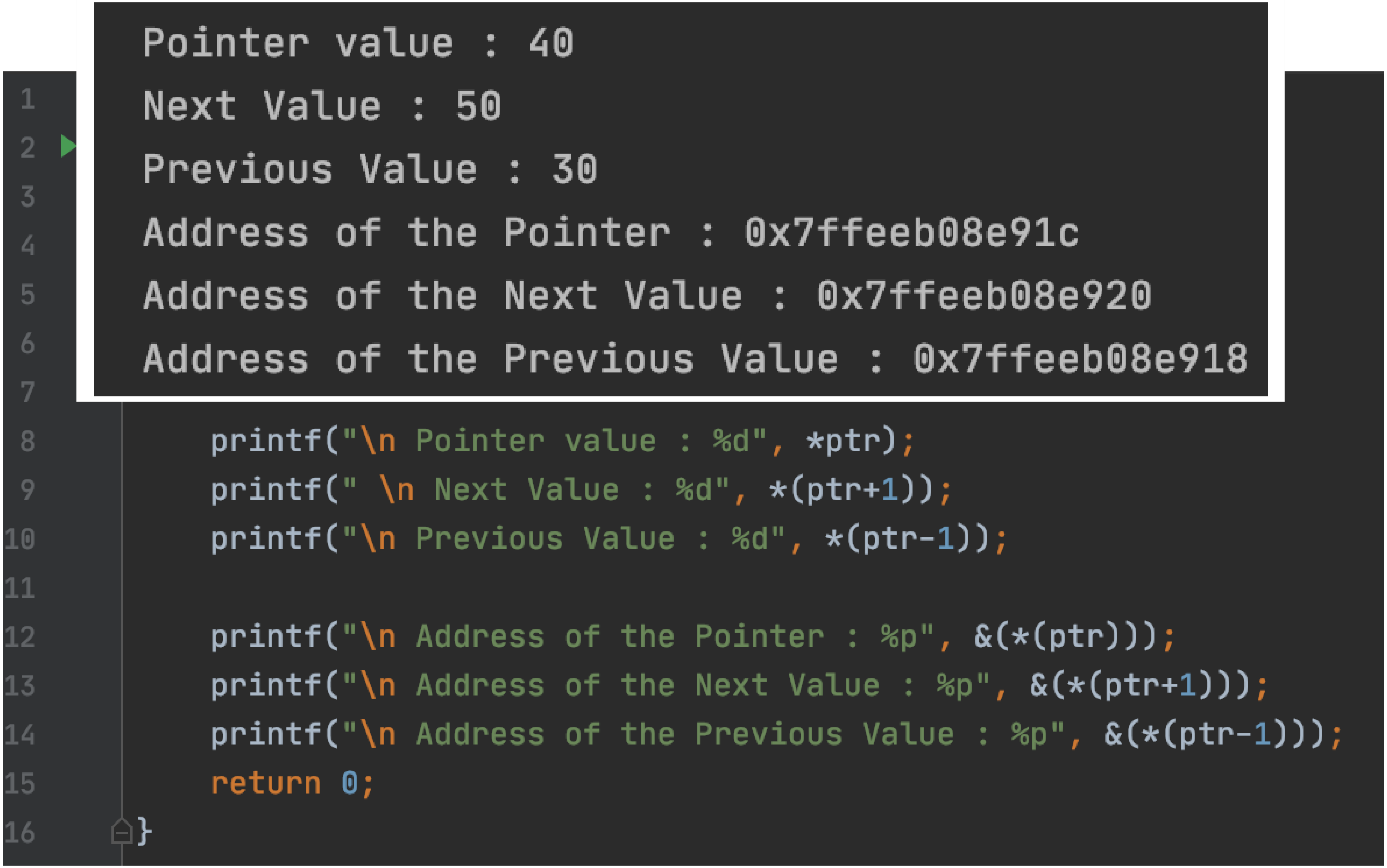
```
address of arr[0] = 0x7ffeeece0910
address of arr[1] = 0x7ffeeece0914
address of arr[2] = 0x7ffeeece0918
address of arr[3] = 0x7ffeeece091c
address of arr[4] = 0x7ffeeece0920
address of arr2[0] = 0x7ffeeece08e0
address of arr2[1] = 0x7ffeeece08e8
address of arr2[2] = 0x7ffeeece08f0
address of arr2[3] = 0x7ffeeece08f8
address of arr2[4] = 0x7ffeeece0900
```

Array Manipulation



```c
#include <stdio.h>
int main() {
    int arr[5] = {10, 20, 30, 40, 50};
    int *ptr;

    ptr = &arr[3]; // address of the fourth element

    printf("\n Pointer value : %d", *ptr);
    printf(" \n Next Value : %d", *(ptr+1));
    printf("\n Previous Value : %d", *(ptr-1));

    printf("\n Address of the Pointer : %p", &(*(ptr)));
    printf("\n Address of the Next Value : %p", &(*(ptr+1)));
    printf("\n Address of the Previous Value : %p", &(*(ptr-1)));
    return 0;
}
```

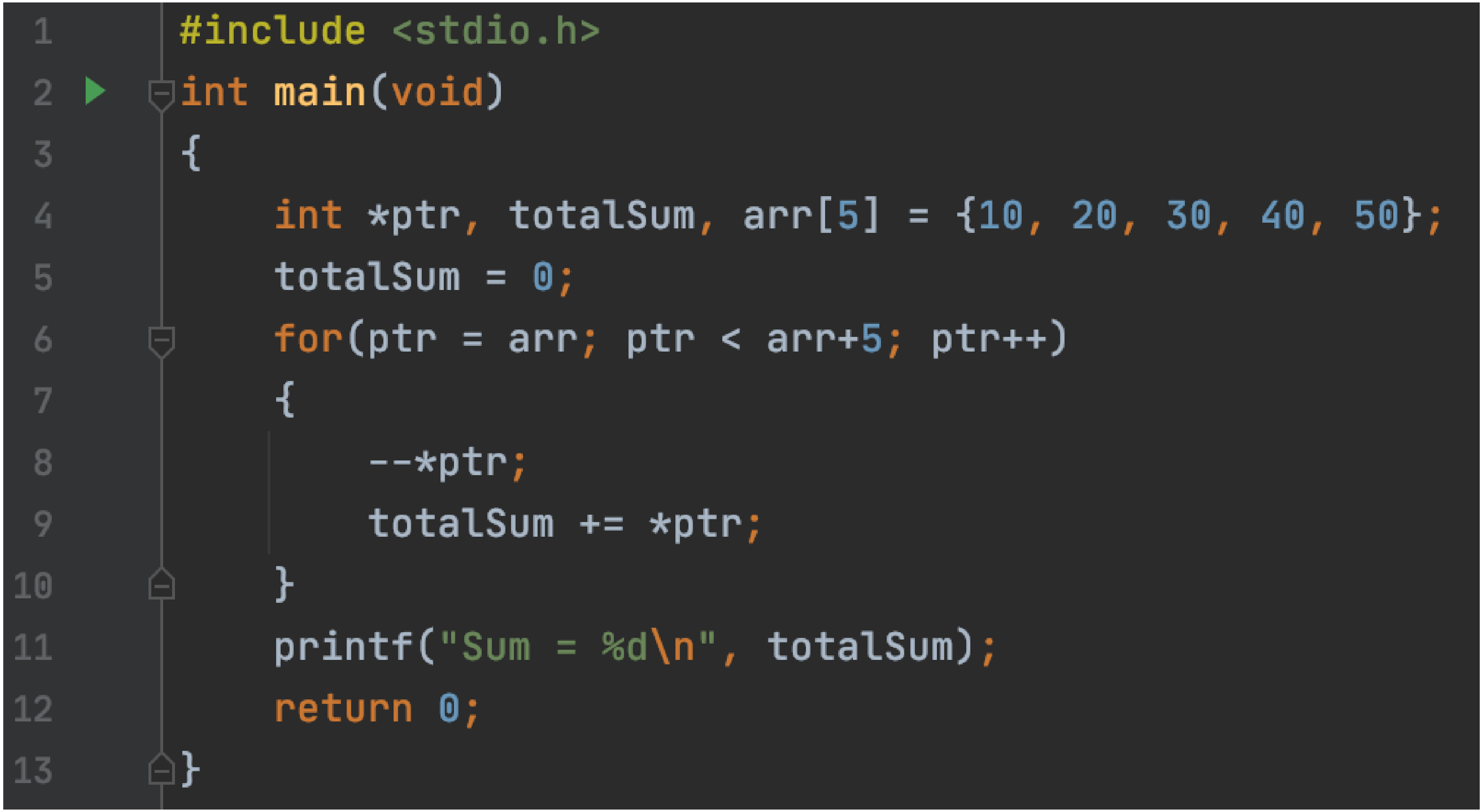
Array Manipulation



```
Pointer value : 40
Next Value : 50
Previous Value : 30
Address of the Pointer : 0x7ffeeb08e91c
Address of the Next Value : 0x7ffeeb08e920
Address of the Previous Value : 0x7ffeeb08e918
```

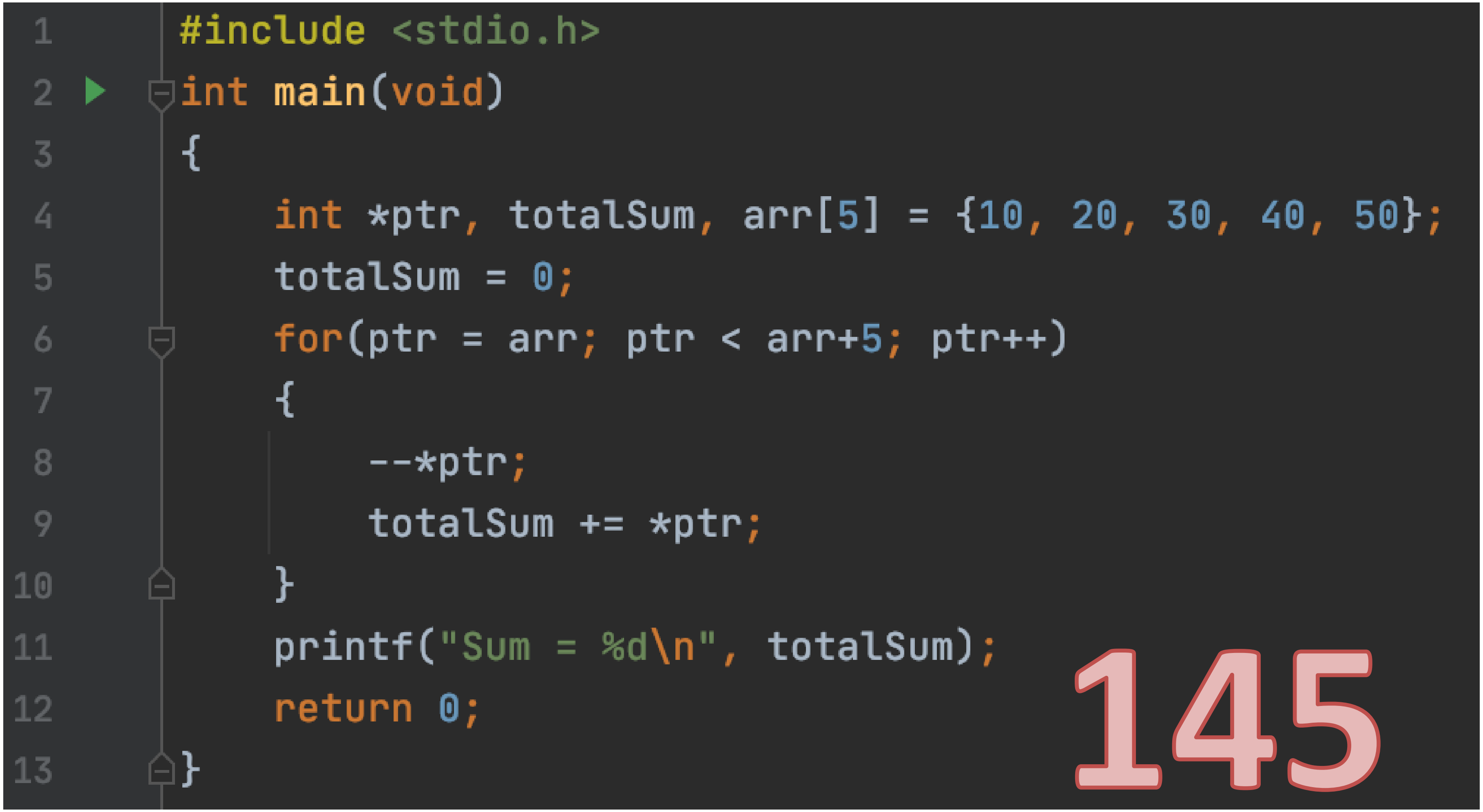
```c
    printf("\n Pointer value : %d", *ptr);
    printf(" \n Next Value : %d", *(ptr+1));
    printf("\n Previous Value : %d", *(ptr-1));

    printf("\n Address of the Pointer : %p", &(*(ptr)));
    printf("\n Address of the Next Value : %p", &(*(ptr+1)));
    printf("\n Address of the Previous Value : %p", &(*(ptr-1)));
    return 0;
}
```

What is the Result ?



```c
#include <stdio.h>
int main(void)
{
    int *ptr, totalSum, arr[5] = {10, 20, 30, 40, 50};
    totalSum = 0;
    for(ptr = arr; ptr < arr+5; ptr++)
    {
        --*ptr;
        totalSum += *ptr;
    }
    printf("Sum = %d\n", totalSum);
    return 0;
}
```

What is the Result ?



```c
#include <stdio.h>
int main(void)
{
    int *ptr, totalSum, arr[5] = {10, 20, 30, 40, 50};
    totalSum = 0;
    for(ptr = arr; ptr < arr+5; ptr++)
    {
        --*ptr;
        totalSum += *ptr;
    }
    printf("Sum = %d\n", totalSum);
    return 0;
}
```

145

References

- https://www.tutorialspoint.com/cprogramming/c_constants.htm
- C From Theory to Practice - 2nd edition, Nikolaos D. Tselikas and George S. Tselikis