Lecture 20

Pointers-2

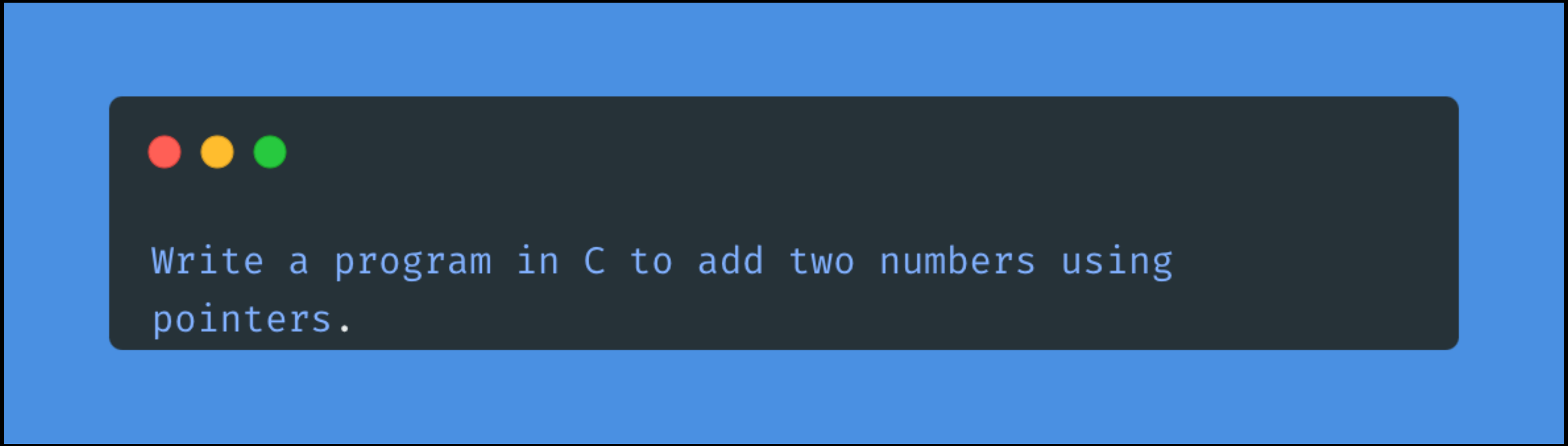
RECURSION 23

SIMPLIFIED CSE COURSE FOR
ALL DEPARTMENTS

C & C++

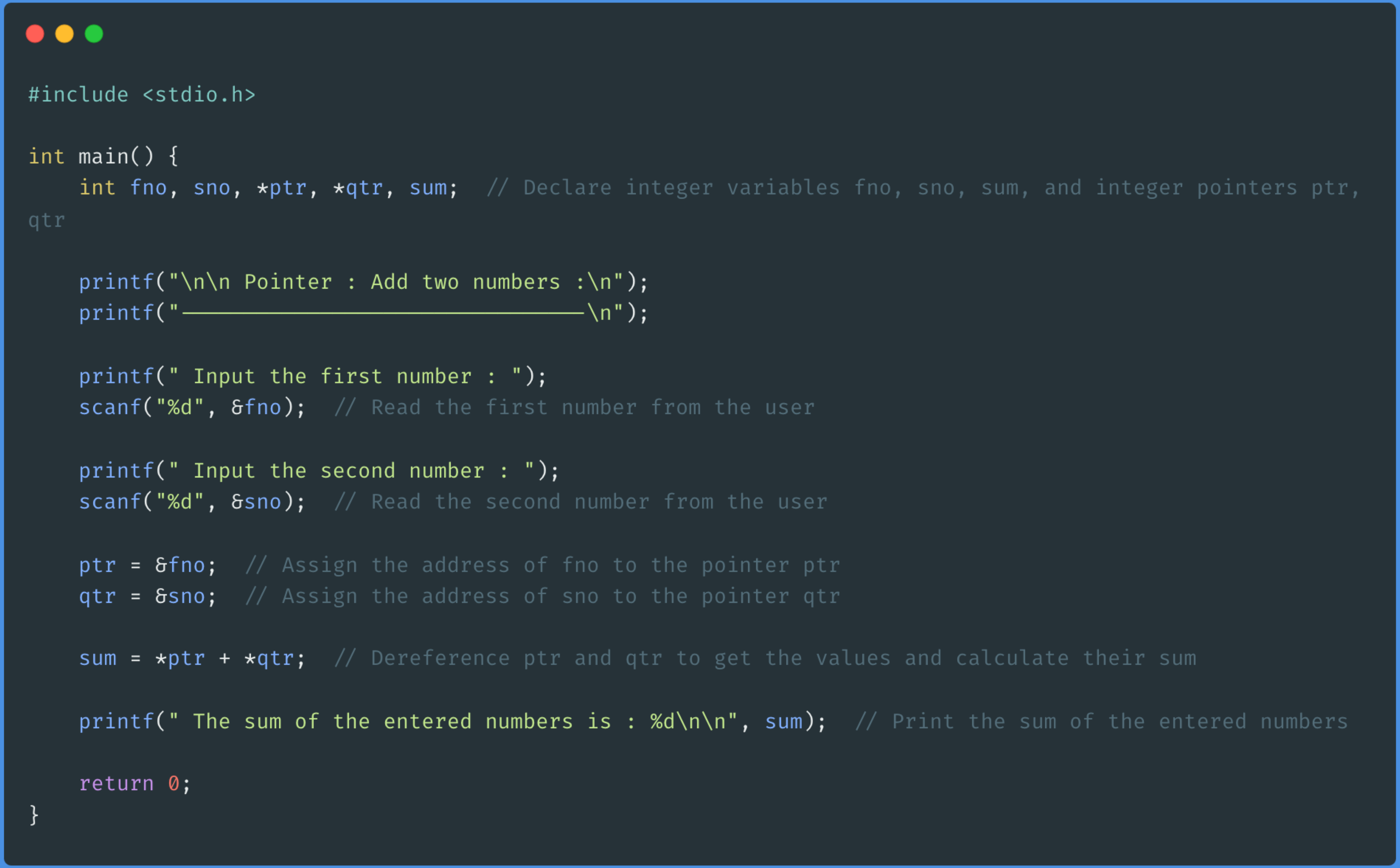
Last Class Review

Problem



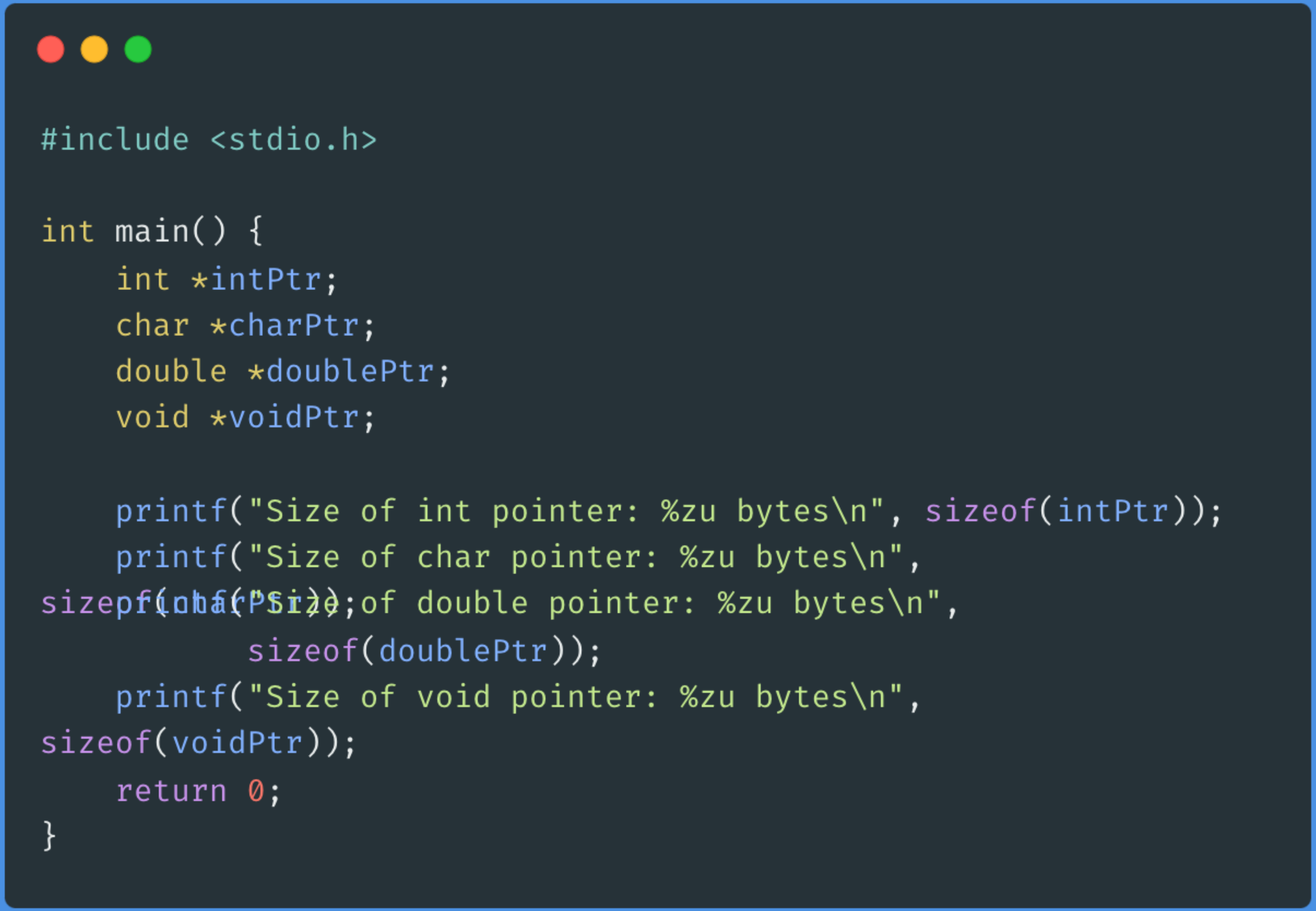
Write a program in C to add two numbers using pointers.

Solution



```c
#include <stdio.h>

int main() {
    int fno, sno, *ptr, *qtr, sum;  // Declare integer variables fno, sno, sum, and integer pointers ptr, qtr

    printf("\n\n Pointer : Add two numbers :\n");
    printf("--------------------------------\n");

    printf(" Input the first number : ");
    scanf("%d", &fno);  // Read the first number from the user

    printf(" Input the second number : ");
    scanf("%d", &sno);  // Read the second number from the user

    ptr = &fno;  // Assign the address of fno to the pointer ptr
    qtr = &sno;  // Assign the address of sno to the pointer qtr

    sum = *ptr + *qtr;  // Dereference ptr and qtr to get the values and calculate their sum

    printf(" The sum of the entered numbers is : %d\n\n", sum);  // Print the sum of the entered numbers

    return 0;
}
```

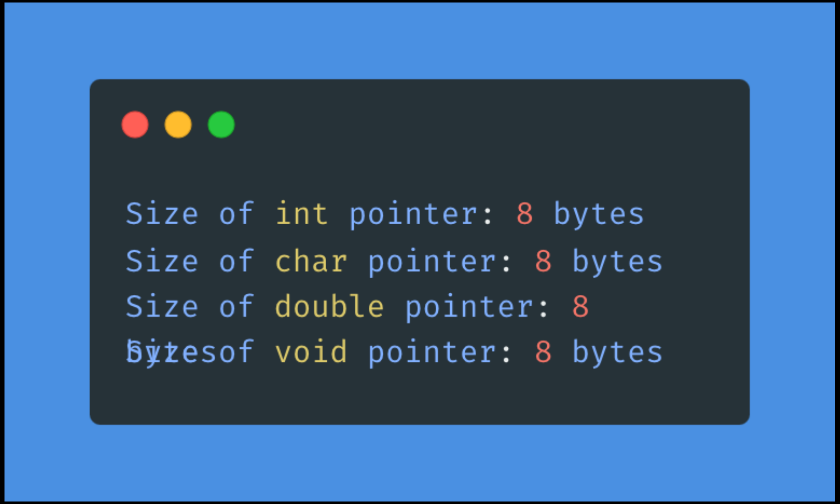
Size of Pointers



```c
#include <stdio.h>

int main() {
    int *intPtr;
    char *charPtr;
    double *doublePtr;
    void *voidPtr;

    printf("Size of int pointer: %zu bytes\n", sizeof(intPtr));
    printf("Size of char pointer: %zu bytes\n",
sizeof(charPtr));
    printf("Size of double pointer: %zu bytes\n",
           sizeof(doublePtr));
    printf("Size of void pointer: %zu bytes\n",
sizeof(voidPtr));
    return 0;
}
```

Output Reason

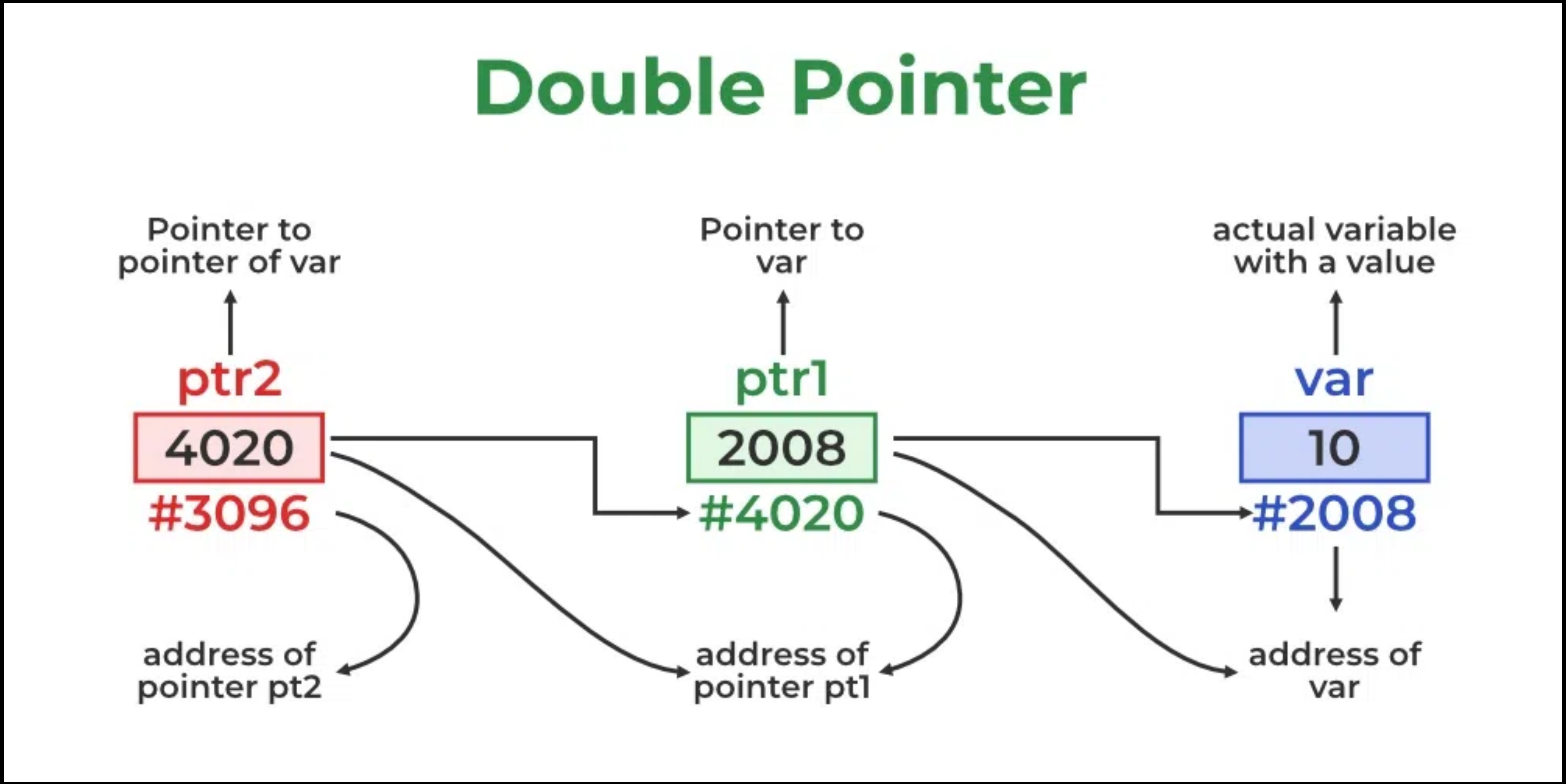
- The size of pointers in C is determined by the system architecture, not the data type they point to. Typically:
    - ○ On a 32-bit system, all pointers are usually 4 bytes (32 bits).
    - ○ On a 64-bit system, all pointers are usually 8 bytes (64 bits).



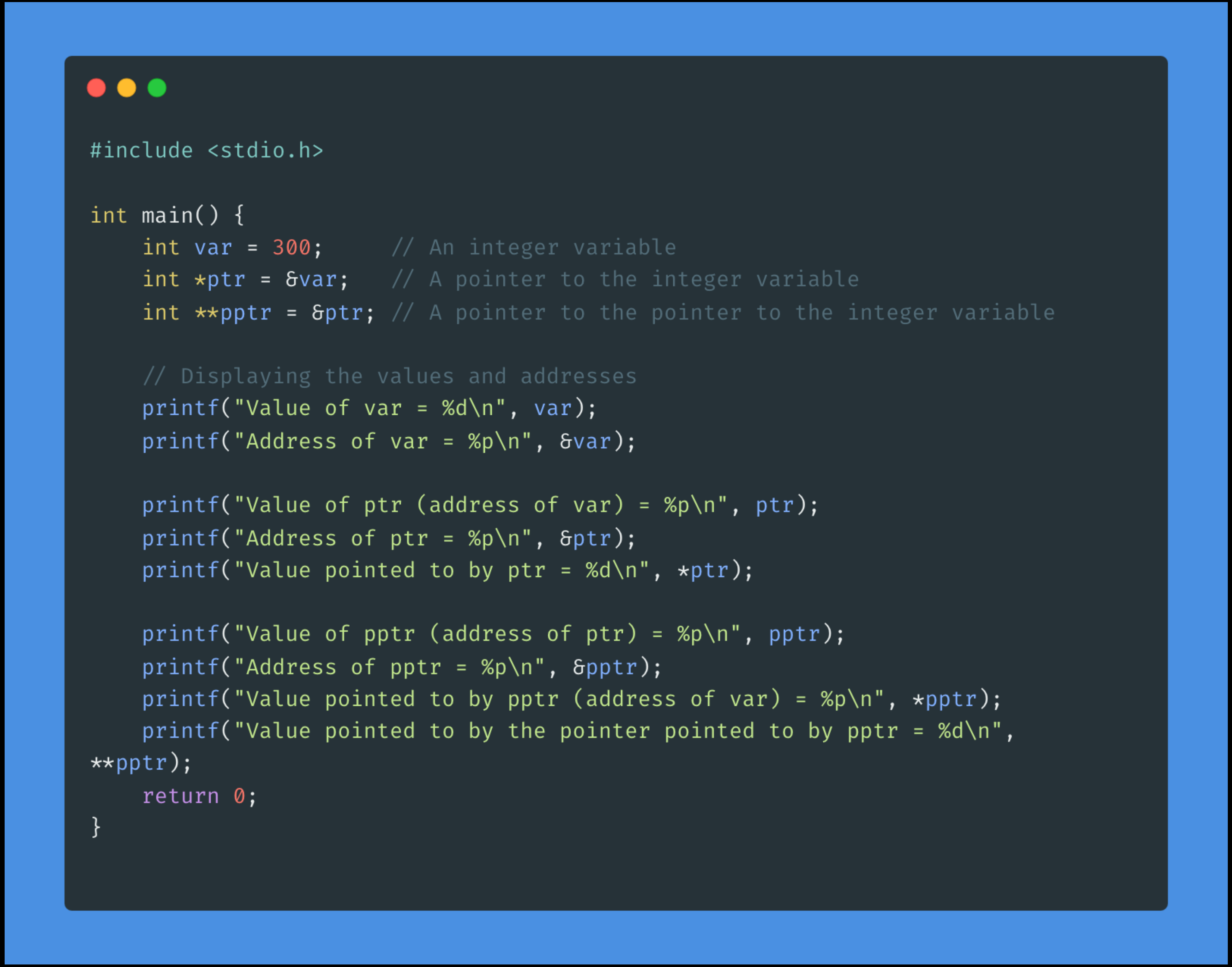
```
Size of int pointer: 8 bytes
Size of char pointer: 8 bytes
Size of double pointer: 8
bytes
Size of void pointer: 8 bytes
```

This is because a pointer holds a memory address, and the size of an address depends on the system architecture.

Pointer of Pointer



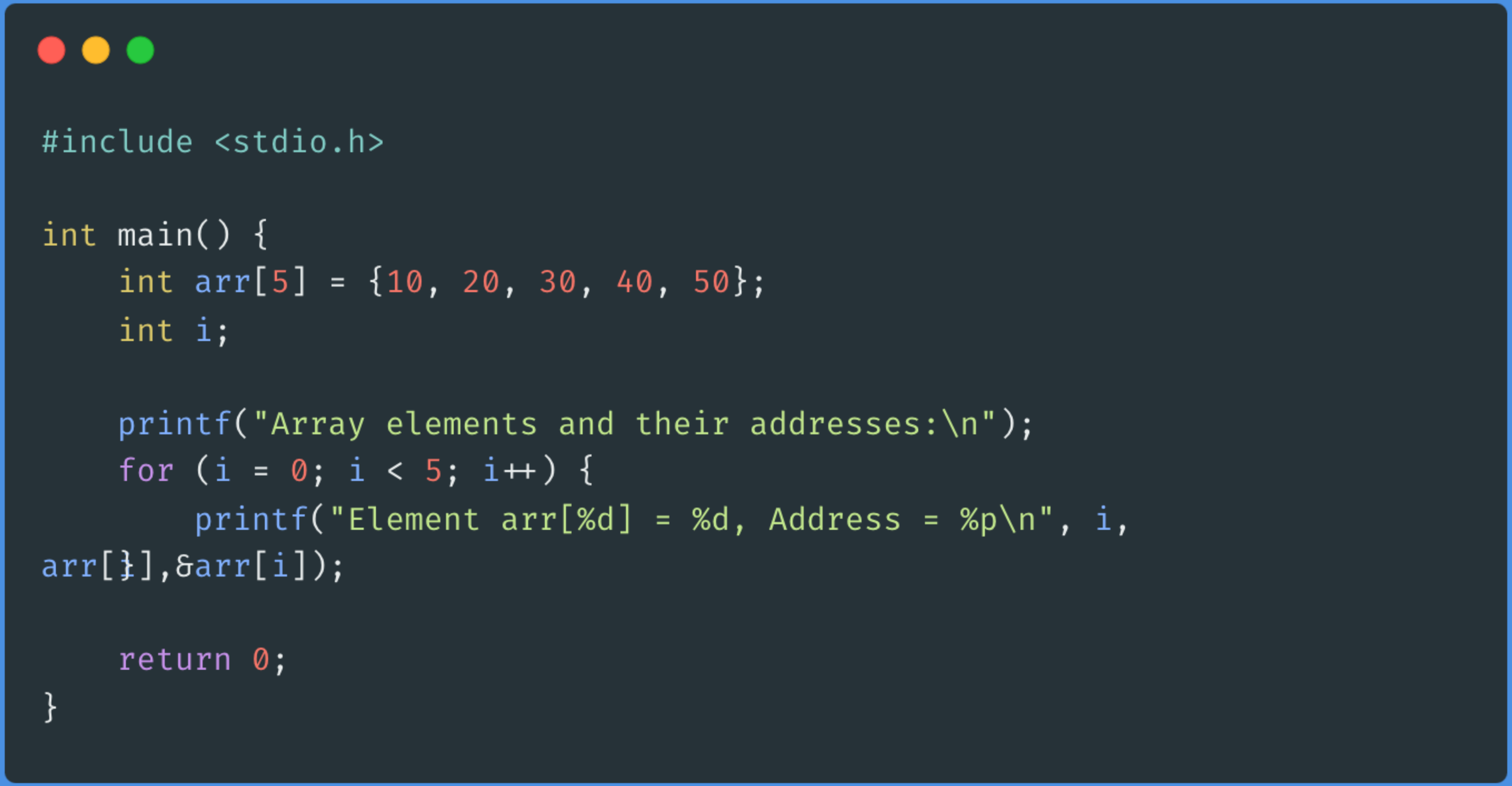
Pointer of Pointer



```c
#include <stdio.h>

int main() {
    int var = 300;     // An integer variable
    int *ptr = &var;   // A pointer to the integer variable
    int **pptr = &ptr; // A pointer to the pointer to the integer variable

    // Displaying the values and addresses
    printf("Value of var = %d\n", var);
    printf("Address of var = %p\n", &var);

    printf("Value of ptr (address of var) = %p\n", ptr);
    printf("Address of ptr = %p\n", &ptr);
    printf("Value pointed to by ptr = %d\n", *ptr);

    printf("Value of pptr (address of ptr) = %p\n", pptr);
    printf("Address of pptr = %p\n", &pptr);
    printf("Value pointed to by pptr (address of var) = %p\n", *pptr);
    printf("Value pointed to by the pointer pointed to by pptr = %d\n",
**pptr);
    return 0;
}
```

Array and Pointers

Address of array elements



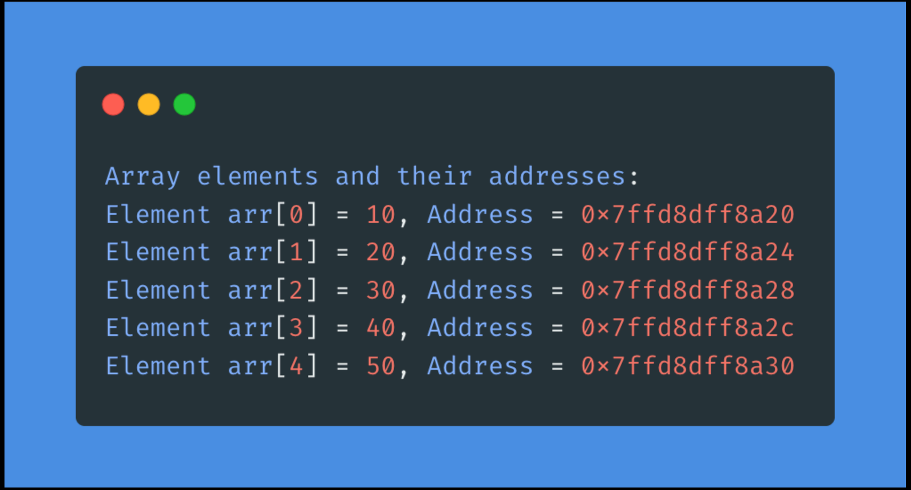
```
#include <stdio.h>

int main() {
    int arr[5] = {10, 20, 30, 40, 50};
    int i;

    printf("Array elements and their addresses:\n");
    for (i = 0; i < 5; i++) {
        printf("Element arr[%d] = %d, Address = %p\n", i,
arr[i],&arr[i]);
    }

    return 0;
}
```

Output Reason

1. Addresses:
   - The addresses are printed in hexadecimal format.
   - Notice the pattern in the addresses: each address increases by 4 bytes.
   - This is because `int` is typically 4 bytes in size on most systems.

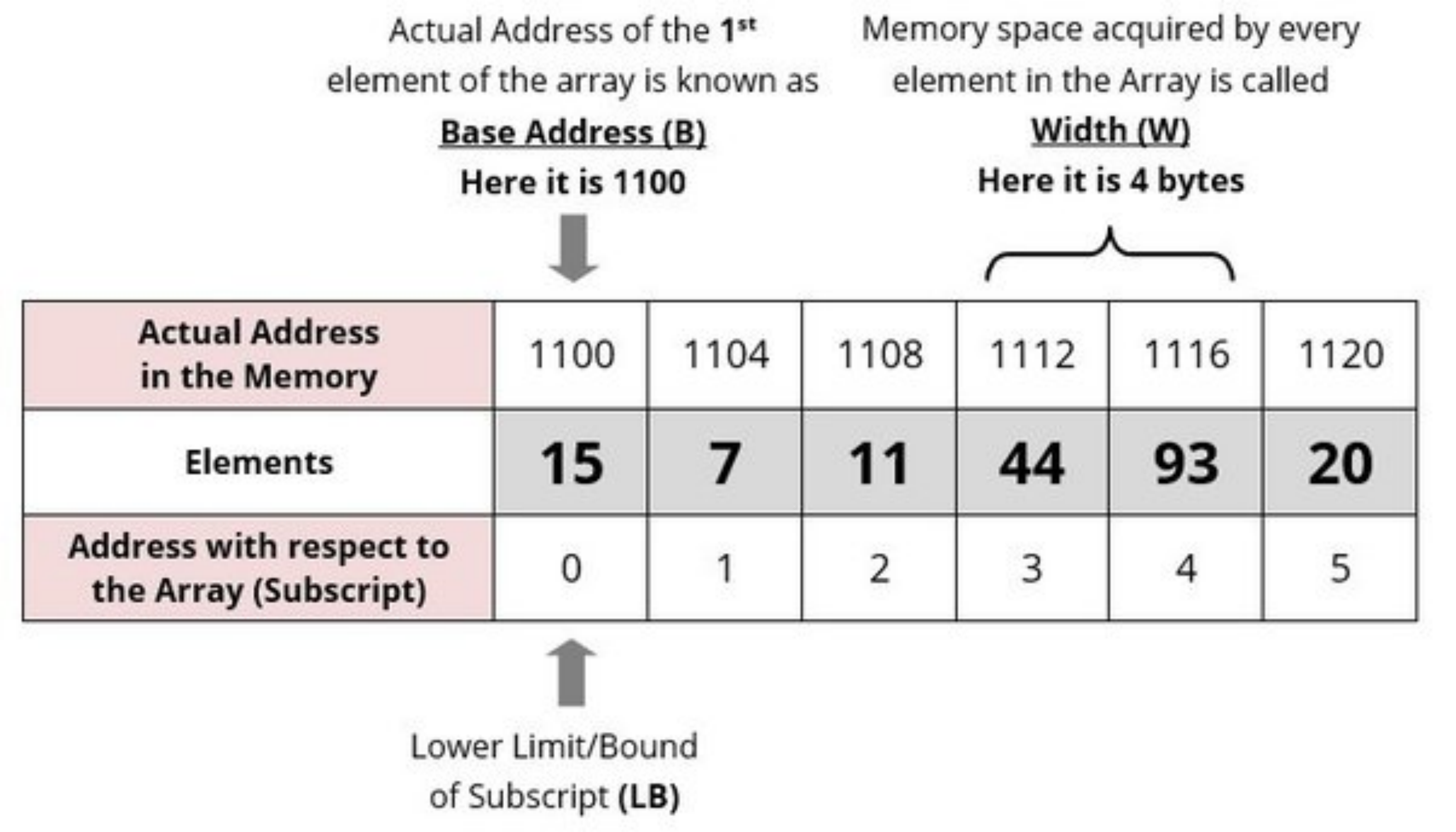
   Detailed Address Calculation:

- Assuming the base address of `arr[0]` is `0x7ffd8dff8a20`:
   - Address of `arr[0]` is `0x7ffd8dff8a20`.
   - Address of `arr[1]` is `0x7ffd8dff8a24` (4 bytes after `arr[0]`).
   - Address of `arr[2]` is `0x7ffd8dff8a28` (8 bytes after `arr[0]`).
   - Address of `arr[3]` is `0x7ffd8dff8a2c` (12 bytes after `arr[0]`).
   - Address of `arr[4]` is `0x7ffd8dff8a30` (16 bytes after `arr[0]`).



```
Array elements and their addresses:
Element arr[0] = 10, Address = 0x7ffd8dff8a20
Element arr[1] = 20, Address = 0x7ffd8dff8a24
Element arr[2] = 30, Address = 0x7ffd8dff8a28
Element arr[3] = 40, Address = 0x7ffd8dff8a2c
Element arr[4] = 50, Address = 0x7ffd8dff8a30
```

Output Reason



Actual Address of the 1st element of the array is known as **Base Address (B)**

Here it is 1100

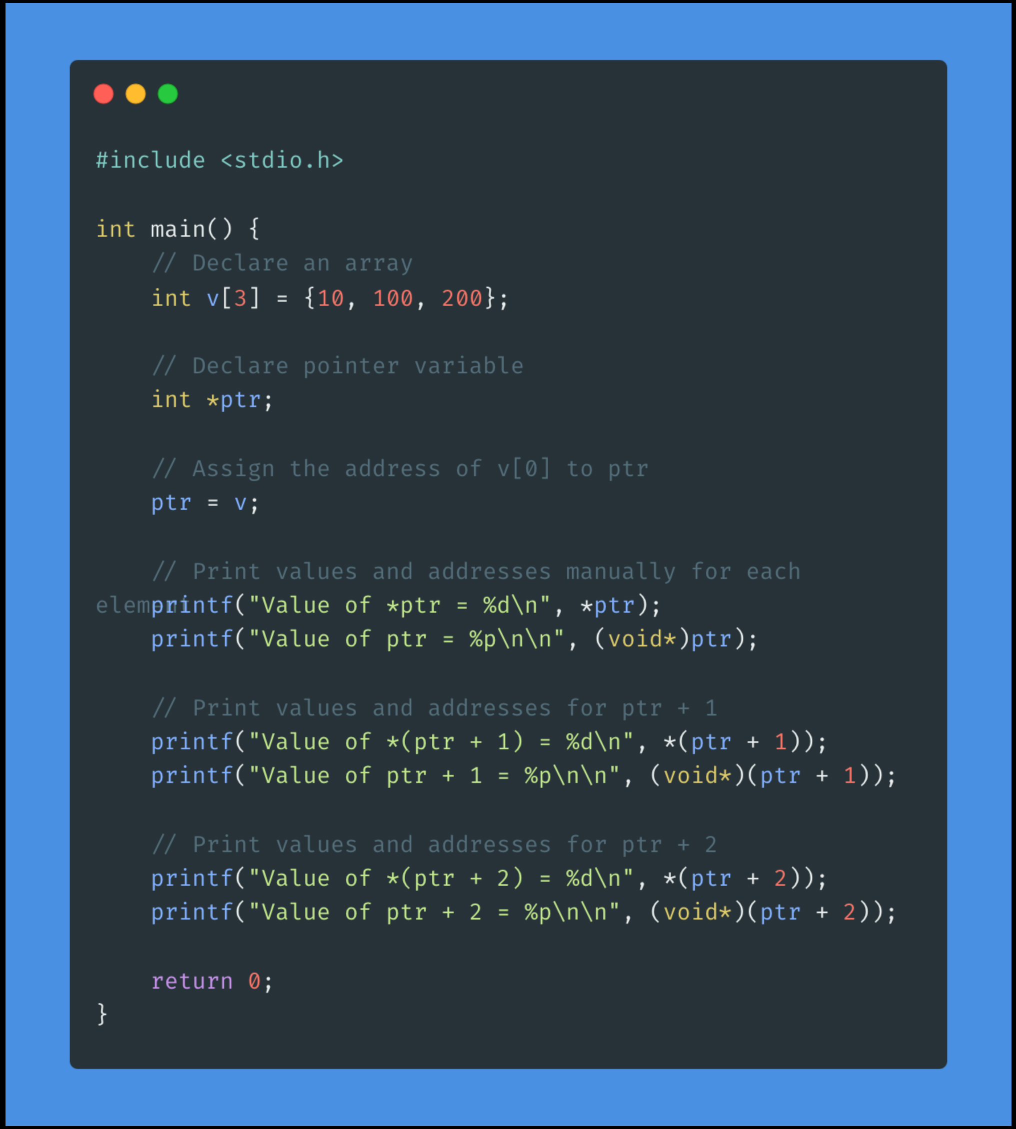
Memory space acquired by every element in the Array is called **Width (W)**

Here it is 4 bytes

Actual Address in the Memory	1100	1104	1108	1112	1116	1120
Elements	**15**	**7**	**11**	**44**	**93**	**20**
Address with respect to the Array (Subscript)	0	1	2	3	4	5

Lower Limit/Bound of Subscript **(LB)**

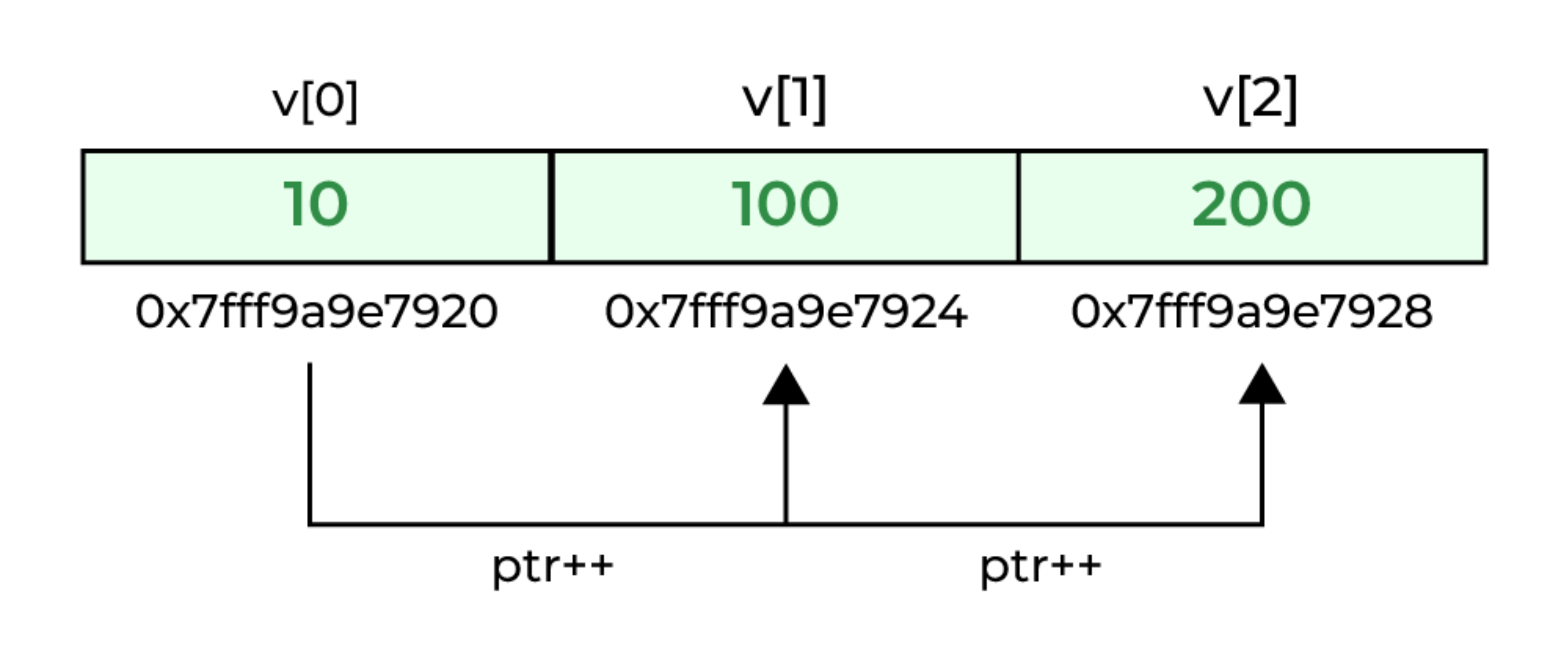
Accessing array with pointer



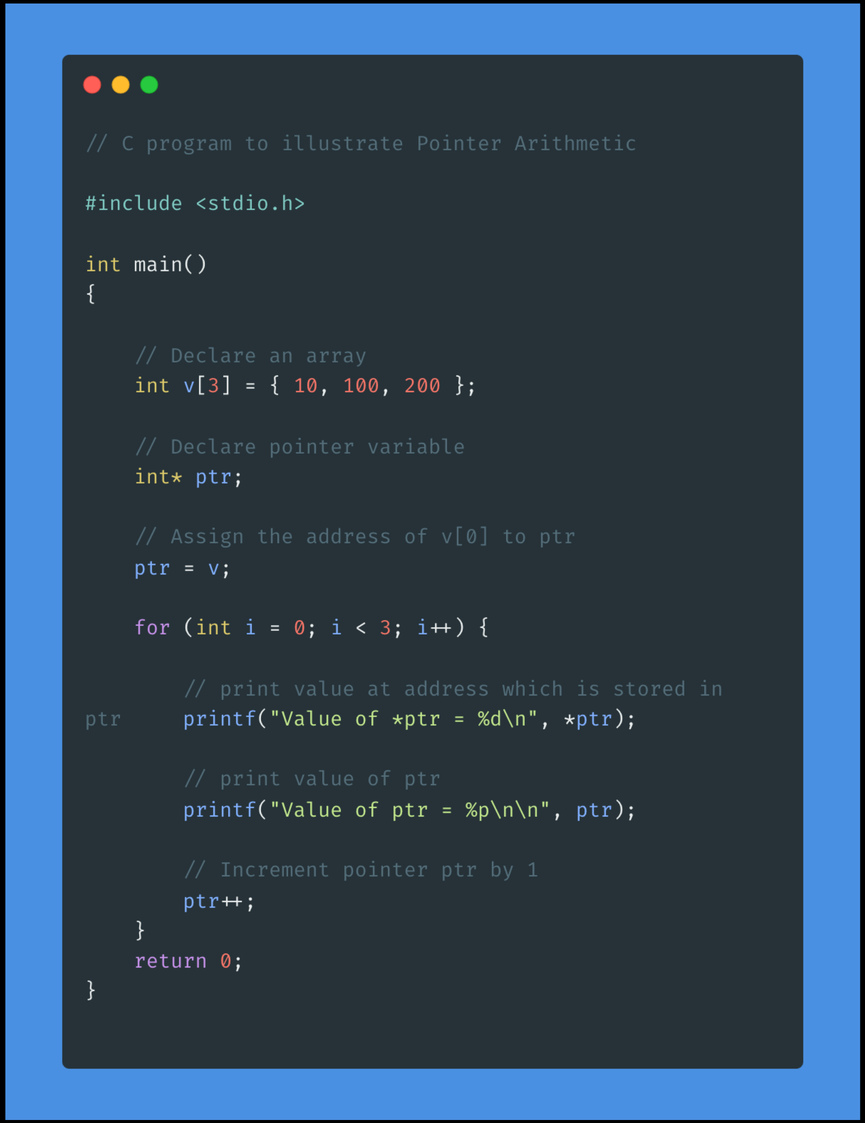
```c
#include <stdio.h>

int main() {
    // Declare an array
    int v[3] = {10, 100, 200};

    // Declare pointer variable
    int *ptr;

    // Assign the address of v[0] to ptr
    ptr = v;

    // Print values and addresses manually for each
elemprintf("Value of *ptr = %d\n", *ptr);
    printf("Value of ptr = %p\n\n", (void*)ptr);

    // Print values and addresses for ptr + 1
    printf("Value of *(ptr + 1) = %d\n", *(ptr + 1));
    printf("Value of ptr + 1 = %p\n\n", (void*)(ptr + 1));

    // Print values and addresses for ptr + 2
    printf("Value of *(ptr + 2) = %d\n", *(ptr + 2));
    printf("Value of ptr + 2 = %p\n\n", (void*)(ptr + 2));

    return 0;
}
```

Meaning of ptr+1

pointer+1 = (address stored in pointer) + size of dataType of pointer

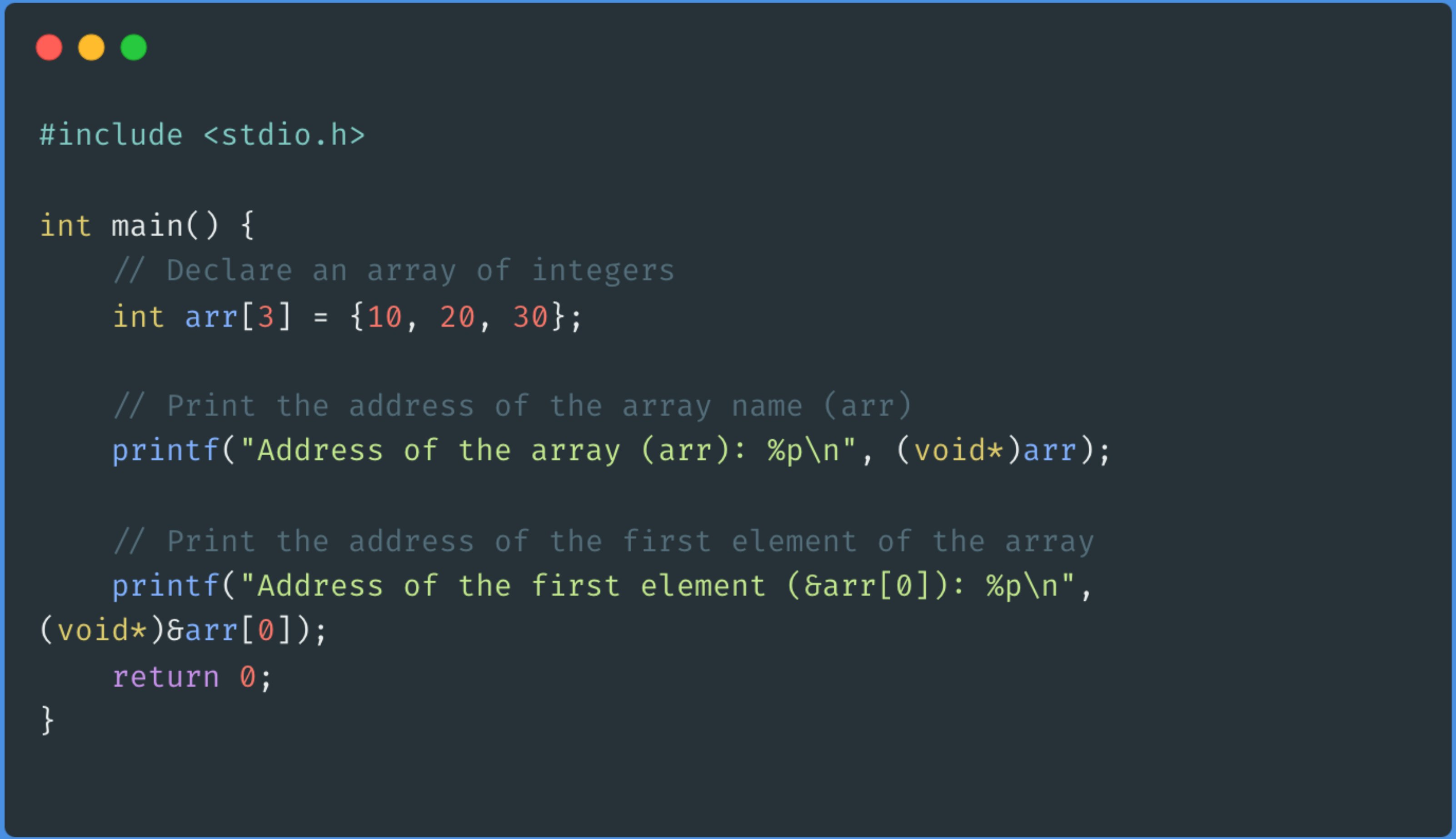


Accessing array with pointer



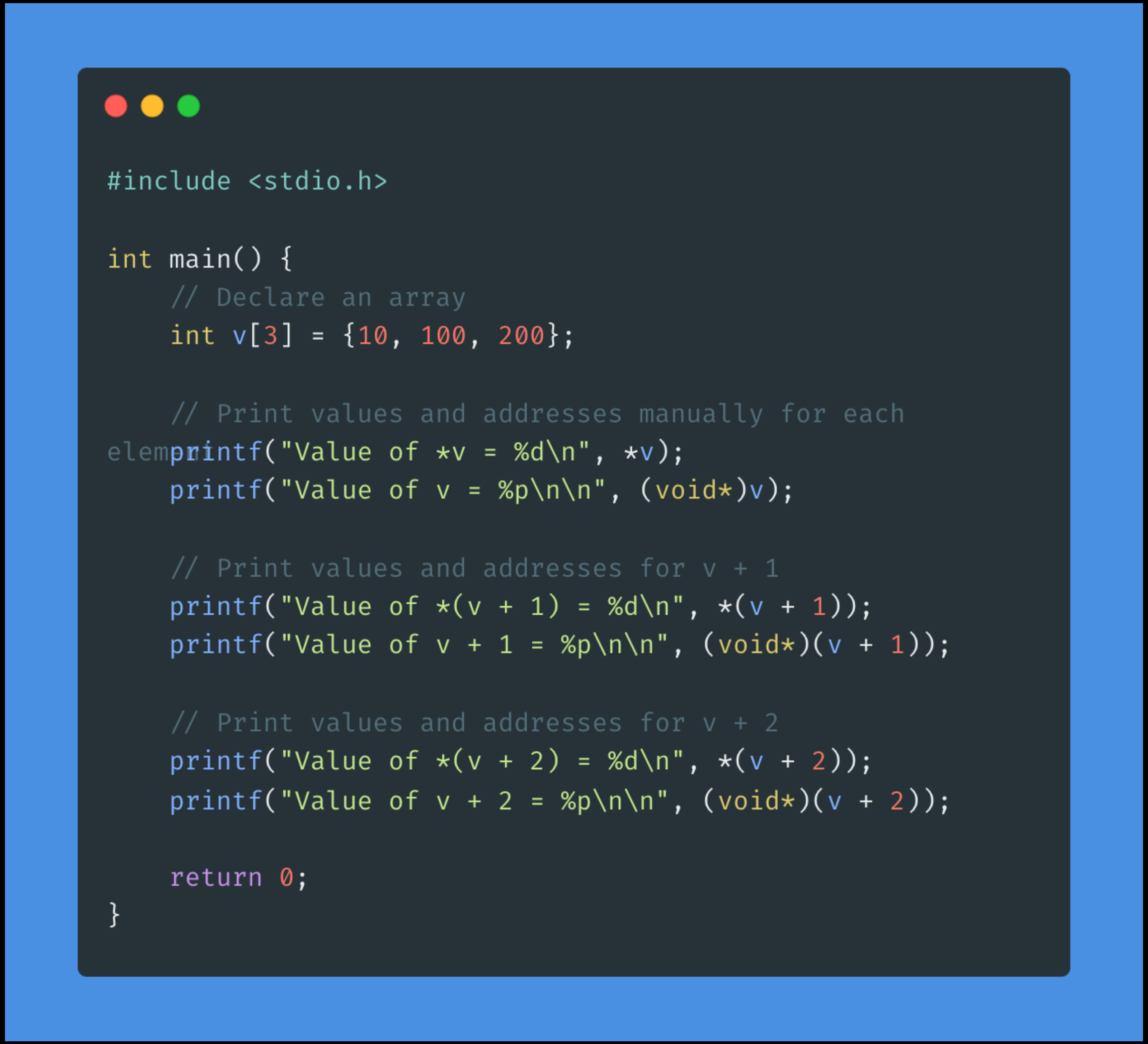
```c
// C program to illustrate Pointer Arithmetic

#include <stdio.h>

int main()
{

    // Declare an array
    int v[3] = { 10, 100, 200 };

    // Declare pointer variable
    int* ptr;

    // Assign the address of v[0] to ptr
    ptr = v;

    for (int i = 0; i < 3; i++) {

        // print value at address which is stored in
ptr     printf("Value of *ptr = %d\n", *ptr);

        // print value of ptr
        printf("Value of ptr = %p\n\n", ptr);

        // Increment pointer ptr by 1
        ptr++;
    }
    return 0;
}
```

Array name contains the address for first variable



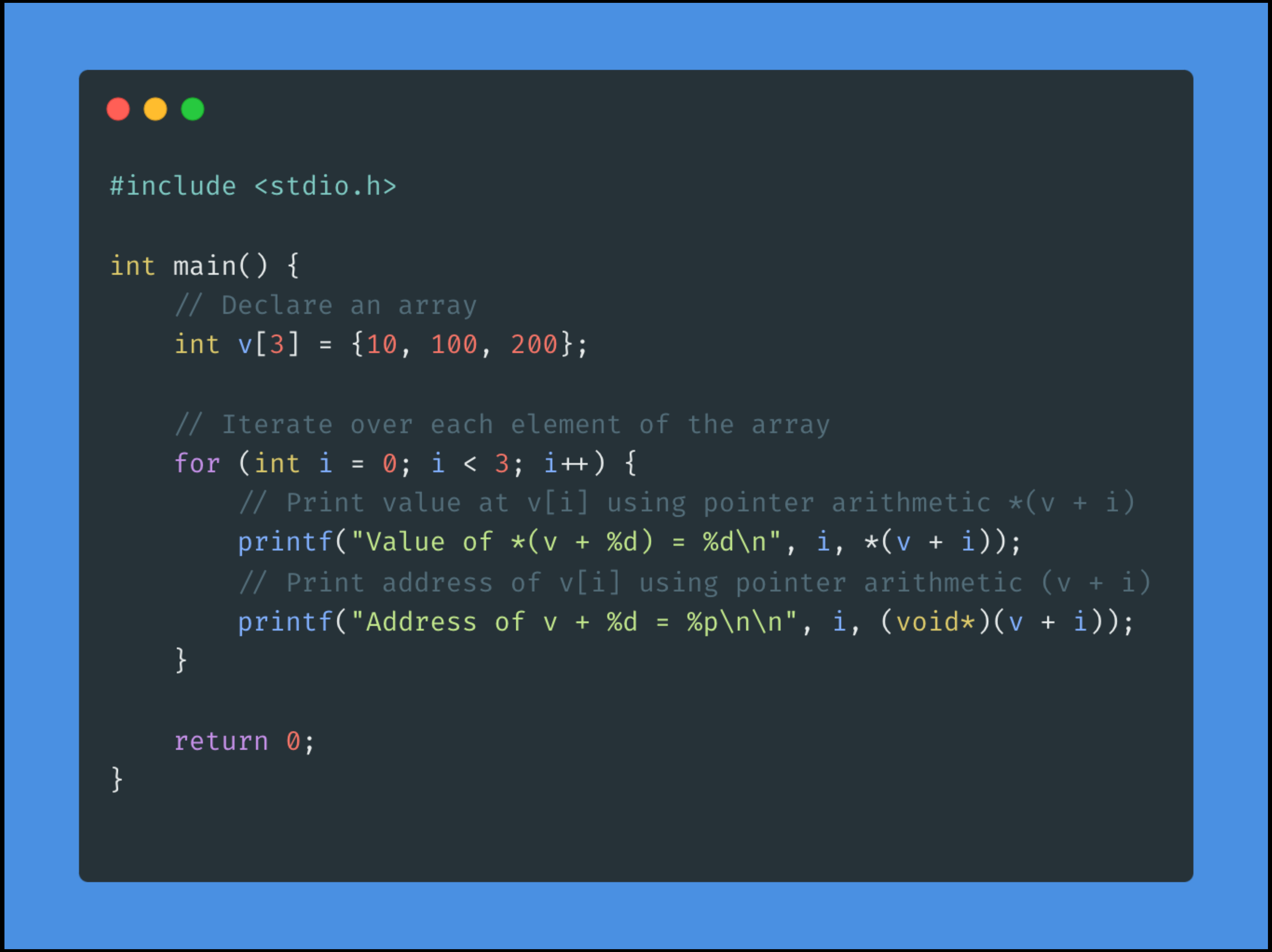
```c
#include <stdio.h>

int main() {
    // Declare an array of integers
    int arr[3] = {10, 20, 30};

    // Print the address of the array name (arr)
    printf("Address of the array (arr): %p\n", (void*)arr);

    // Print the address of the first element of the array
    printf("Address of the first element (&arr[0]): %p\n",
(void*)&arr[0]);
    return 0;
}
```

Accessing array with pointer



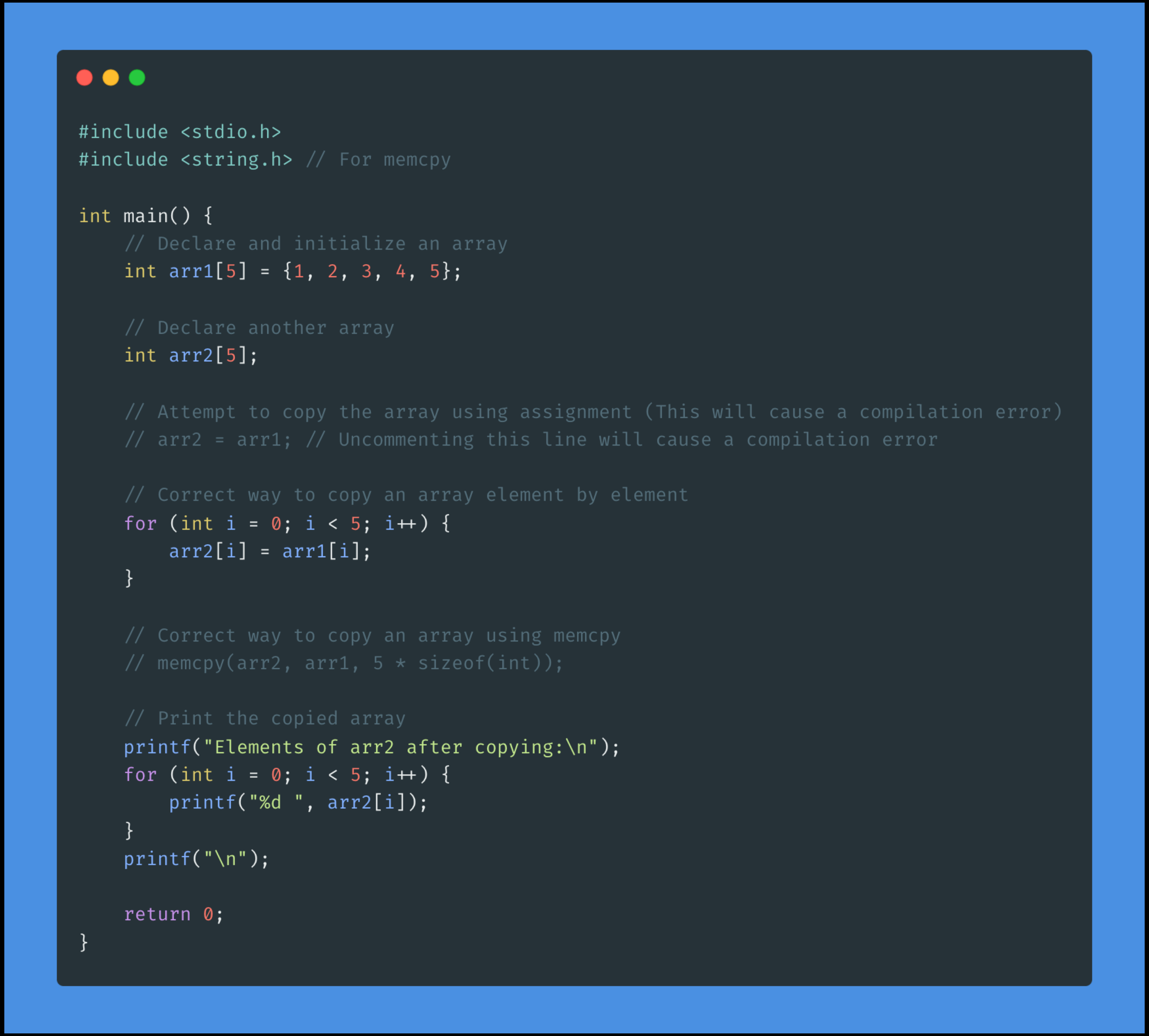
```c
#include <stdio.h>

int main() {
    // Declare an array
    int v[3] = {10, 100, 200};

    // Print values and addresses manually for each
elemprintf("Value of *v = %d\n", *v);
    printf("Value of v = %p\n\n", (void*)v);

    // Print values and addresses for v + 1
    printf("Value of *(v + 1) = %d\n", *(v + 1));
    printf("Value of v + 1 = %p\n\n", (void*)(v + 1));

    // Print values and addresses for v + 2
    printf("Value of *(v + 2) = %d\n", *(v + 2));
    printf("Value of v + 2 = %p\n\n", (void*)(v + 2));

    return 0;
}
```

Accessing array with pointer



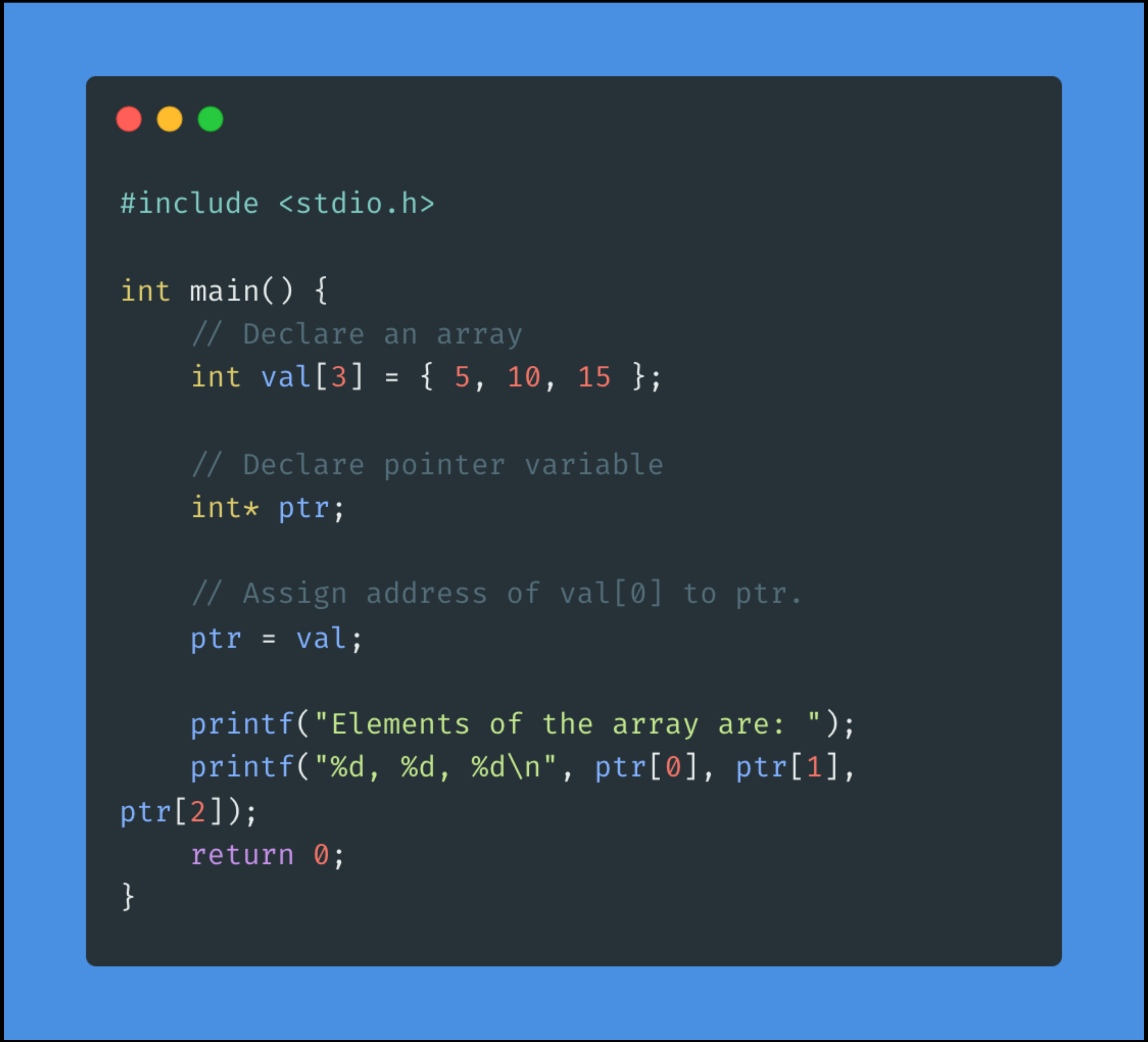
```c
#include <stdio.h>

int main() {
    // Declare an array
    int v[3] = {10, 100, 200};

    // Iterate over each element of the array
    for (int i = 0; i < 3; i++) {
        // Print value at v[i] using pointer arithmetic *(v + i)
        printf("Value of *(v + %d) = %d\n", i, *(v + i));
        // Print address of v[i] using pointer arithmetic (v + i)
        printf("Address of v + %d = %p\n\n", i, (void*)(v + i));
    }

    return 0;
}
```

You cannot copy an array



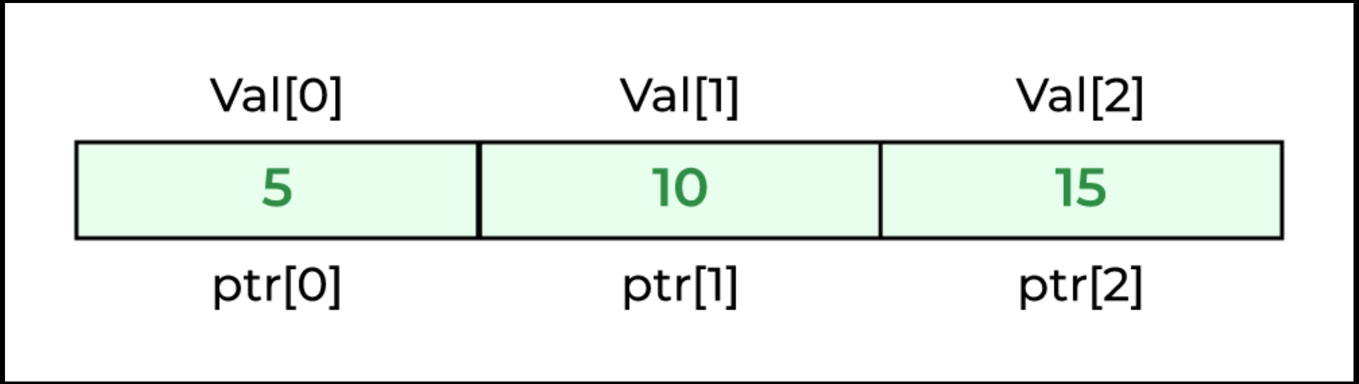
```c
#include <stdio.h>
#include <string.h> // For memcpy

int main() {
    // Declare and initialize an array
    int arr1[5] = {1, 2, 3, 4, 5};

    // Declare another array
    int arr2[5];

    // Attempt to copy the array using assignment (This will cause a compilation error)
    // arr2 = arr1; // Uncommenting this line will cause a compilation error

    // Correct way to copy an array element by element
    for (int i = 0; i < 5; i++) {
        arr2[i] = arr1[i];
    }

    // Correct way to copy an array using memcpy
    // memcpy(arr2, arr1, 5 * sizeof(int));

    // Print the copied array
    printf("Elements of arr2 after copying:\n");
    for (int i = 0; i < 5; i++) {
        printf("%d ", arr2[i]);
    }
    printf("\n");

    return 0;
}
```

But pointers can

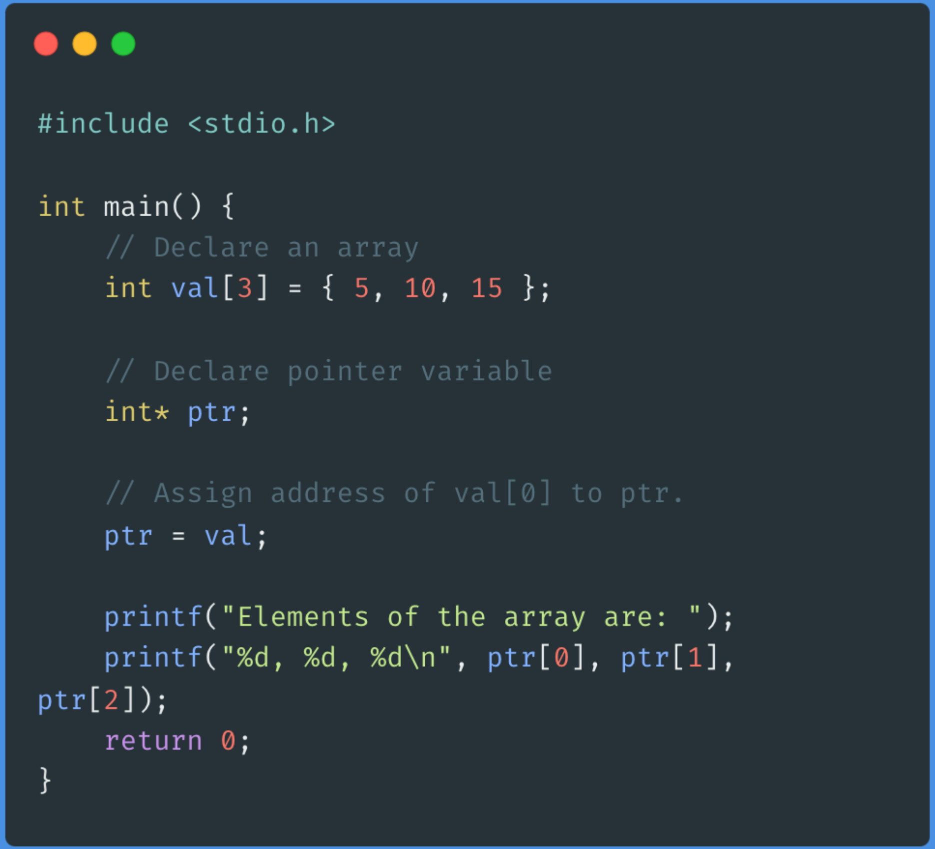


```c
#include <stdio.h>

int main() {
    // Declare an array
    int val[3] = { 5, 10, 15 };

    // Declare pointer variable
    int* ptr;

    // Assign address of val[0] to ptr.
    ptr = val;

    printf("Elements of the array are: ");
    printf("%d, %d, %d\n", ptr[0], ptr[1],
ptr[2]);
    return 0;
}
```

Pointer with array subscript



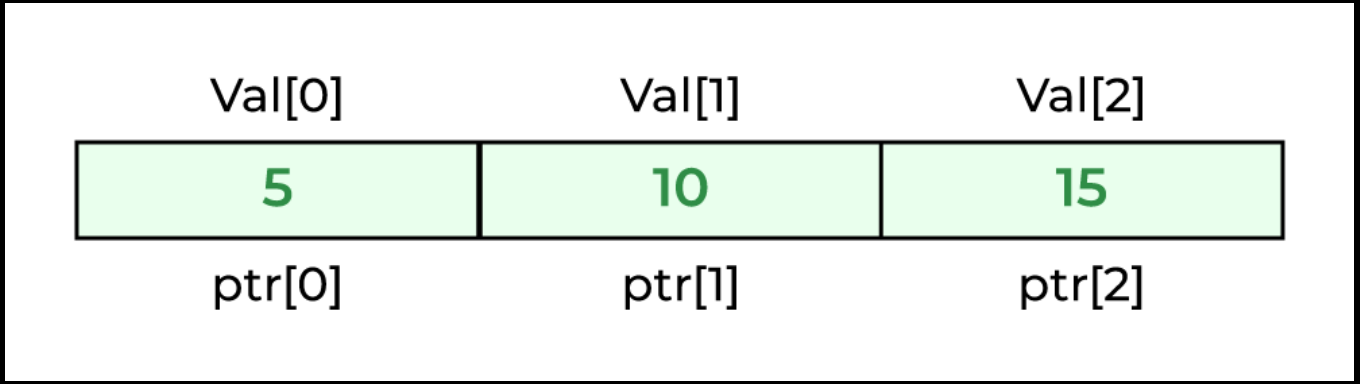
Val[0]	Val[1]	Val[2]
5	10	15
ptr[0]	ptr[1]	ptr[2]



```
#include <stdio.h>

int main() {
    // Declare an array
    int val[3] = { 5, 10, 15 };

    // Declare pointer variable
    int* ptr;

    // Assign address of val[0] to ptr.
    ptr = val;

    printf("Elements of the array are: ");
    printf("%d, %d, %d\n", ptr[0], ptr[1],
ptr[2]);
    return 0;
}
```

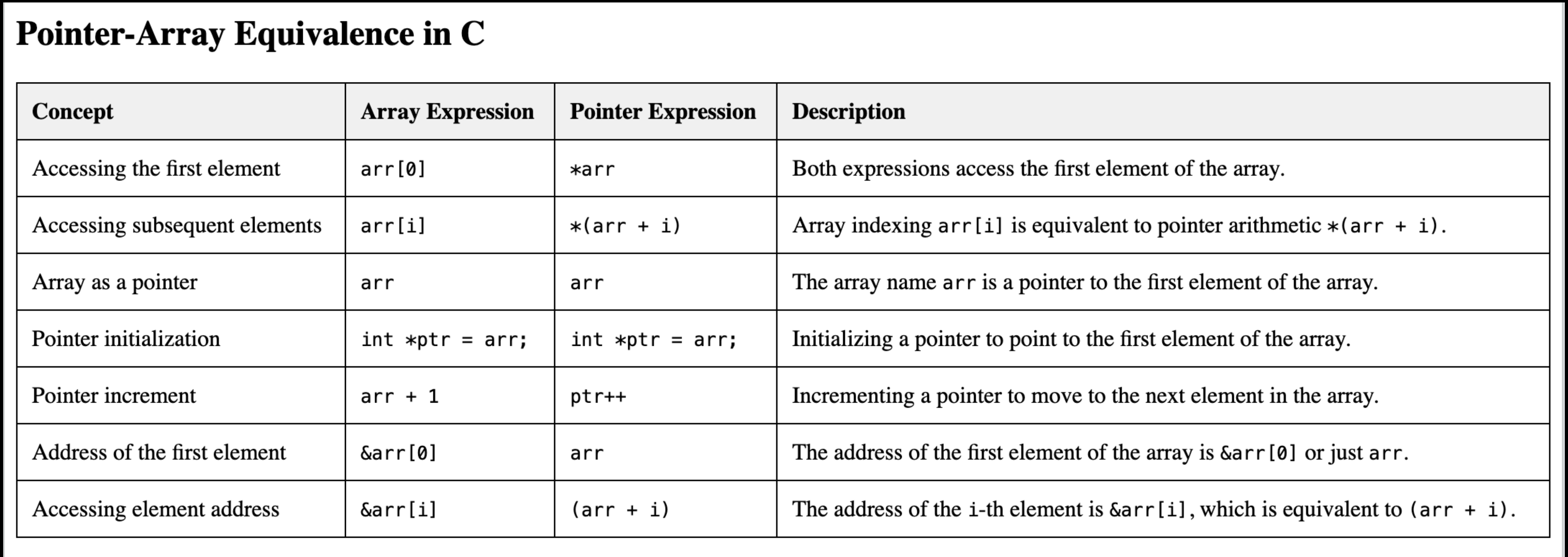
Pointer with array subscript

ptr[1] = *(ptr+1)
arr[1] = *(arr+1)



Val[0]	Val[1]	Val[2]
5	10	15
ptr[0]	ptr[1]	ptr[2]

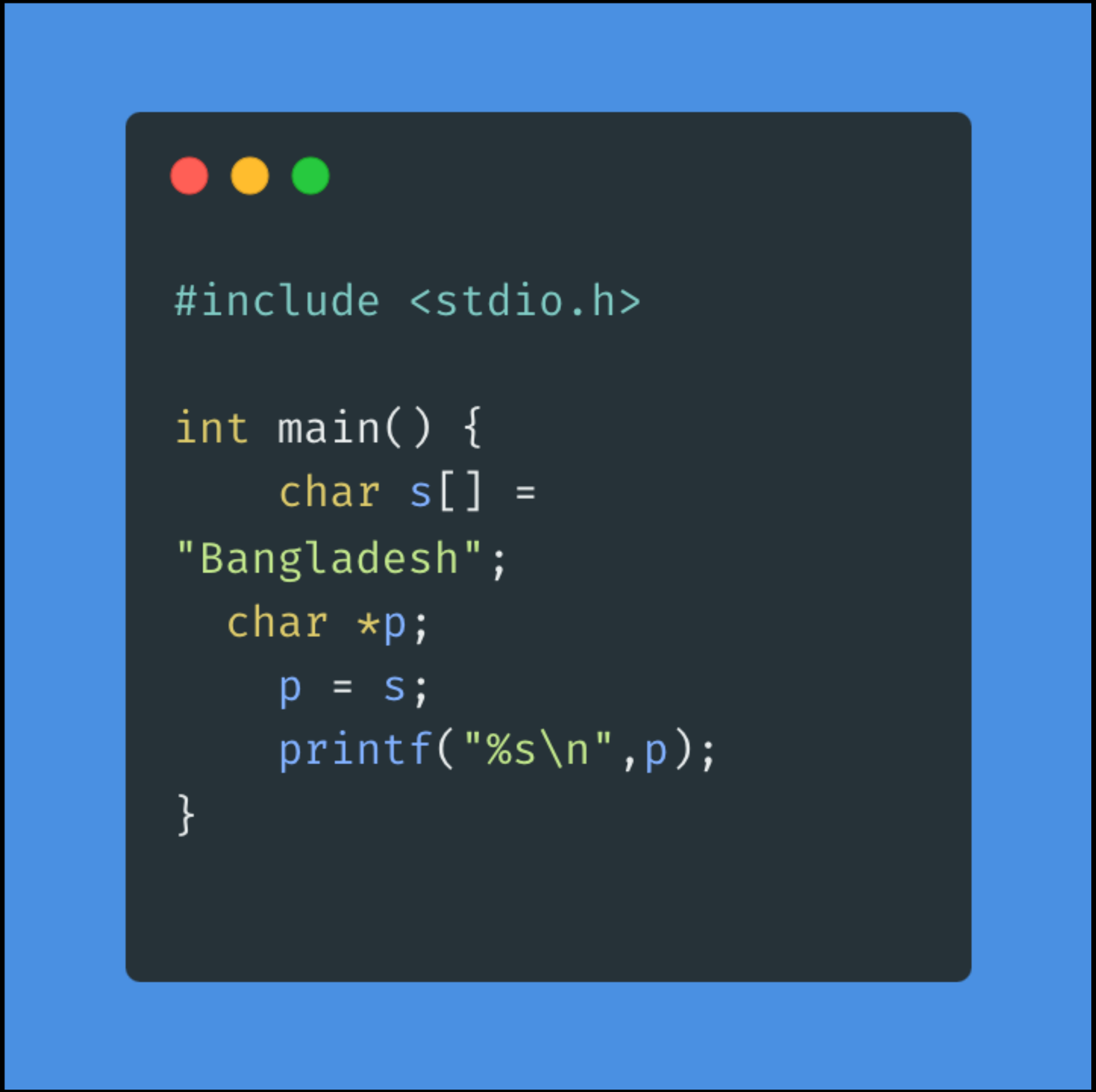
Pointer array similarity



Pointer-Array Equivalence in C

Concept	Array Expression	Pointer Expression	Description
Accessing the first element	`arr[0]`	`*arr`	Both expressions access the first element of the array.
Accessing subsequent elements	`arr[i]`	`*(arr + i)`	Array indexing `arr[i]` is equivalent to pointer arithmetic `*(arr + i)`.
Array as a pointer	`arr`	`arr`	The array name `arr` is a pointer to the first element of the array.
Pointer initialization	`int *ptr = arr;`	`int *ptr = arr;`	Initializing a pointer to point to the first element of the array.
Pointer increment	`arr + 1`	`ptr++`	Incrementing a pointer to move to the next element in the array.
Address of the first element	`&arr[0]`	`arr`	The address of the first element of the array is `&arr[0]` or just `arr`.
Accessing element address	`&arr[i]`	`(arr + i)`	The address of the `i`-th element is `&arr[i]`, which is equivalent to `(arr + i)`.

String and Pointer



```
#include <stdio.h>

int main() {
    char s[] =
"Bangladesh";
  char *p;
    p = s;
    printf("%s\n",p);
}
```

What are the benefits of using pointers?