C

Passing arrays to a Function

Pointers with arrays

Pointers with functions

Pass arrays to a function

- Single element of an array can be passed in similar manner as passing variable to a function.

```c
#include <stdio.h>
void display(int age)
{
    printf("%d", age);
}

int main()
{
    int ageArray[] = { 2, 3, 4 };
    display(ageArray[2]); //Passing array element ageArray[2] only.
    return 0;
}
```

Passing an entire one-dimensional array to a function

While passing arrays as arguments to the function, only the name of the array is passed (i.e., starting address of memory area is passed as argument).

```c
#include <stdio.h>
float average(float age[]);

int main()
{
     float avg, age[] = { 23.4, 55, 22.6, 3, 40.5, 18 };
     avg = average(age); /* Only name of array is passed as argument. *
     printf("Average age=%.2f", avg);
     return 0;
}

float average(float age[])
{
     int i;
     float avg, sum = 0.0;
     for (i = 0; i < 6; ++i) {
          sum += age[i];
     }
     avg = (sum / 6);
     return avg;
}
```

Passing Multi-dimensional Arrays to Function

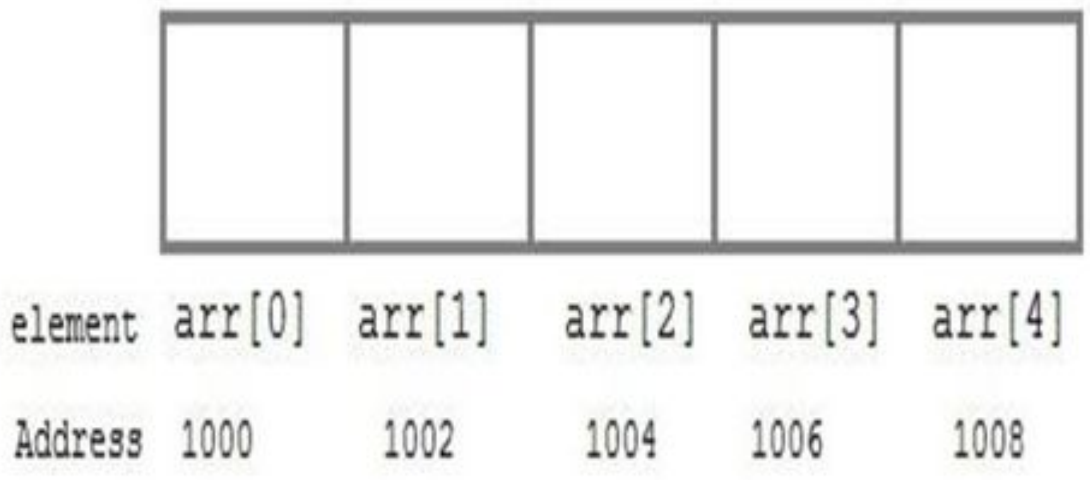
To pass two-dimensional array to a function as an argument, starting address of memory area reserved is passed as in one dimensional array

```
#include <stdio.h>
void displayNumbers(int num[2][2]);
int main()
{
     int num[2][2], i, j;
     printf("Enter 4 numbers:\n");
     for (i = 0; i < 2; ++i)
          for (j = 0; j < 2; ++j)
               scanf("%d", &num[i][j]);
     // passing multi-dimensional array to displayNumbers function
     displayNumbers(num);
     return 0;
}

void displayNumbers(int num[2][2])
{
     // Instead of the above line,
     // void displayNumbers(int num[][2]) is also valid
     int i, j;
     printf("Displaying:\n");
     for (i = 0; i < 2; ++i)
          for (j = 0; j < 2; ++j)
               printf("%d\n", num[i][j]);
}
```

Pointer to an Array

Declaration of array

int arr[5] = { 1, 2, 3, 4, 5 };



Compiler allocates sufficient amount of memory to contain all the elements of the array.

Base address: Base address i.e. address of the first element of the array (also allocated by the compiler).

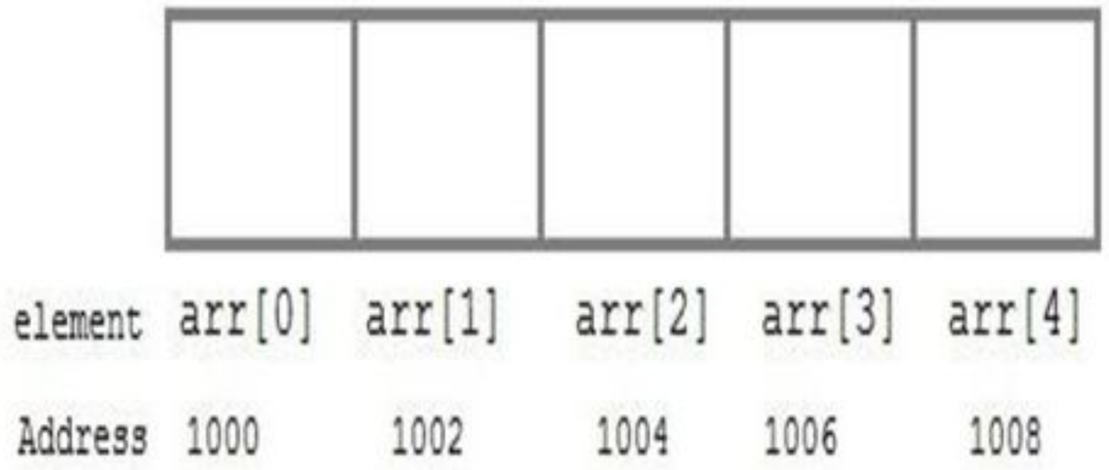
From the above example

Base address is 1000 i.e., arr[0]

Pointer to an Array

Declaration of array

int arr[5] = { 1, 2, 3, 4, 5 };



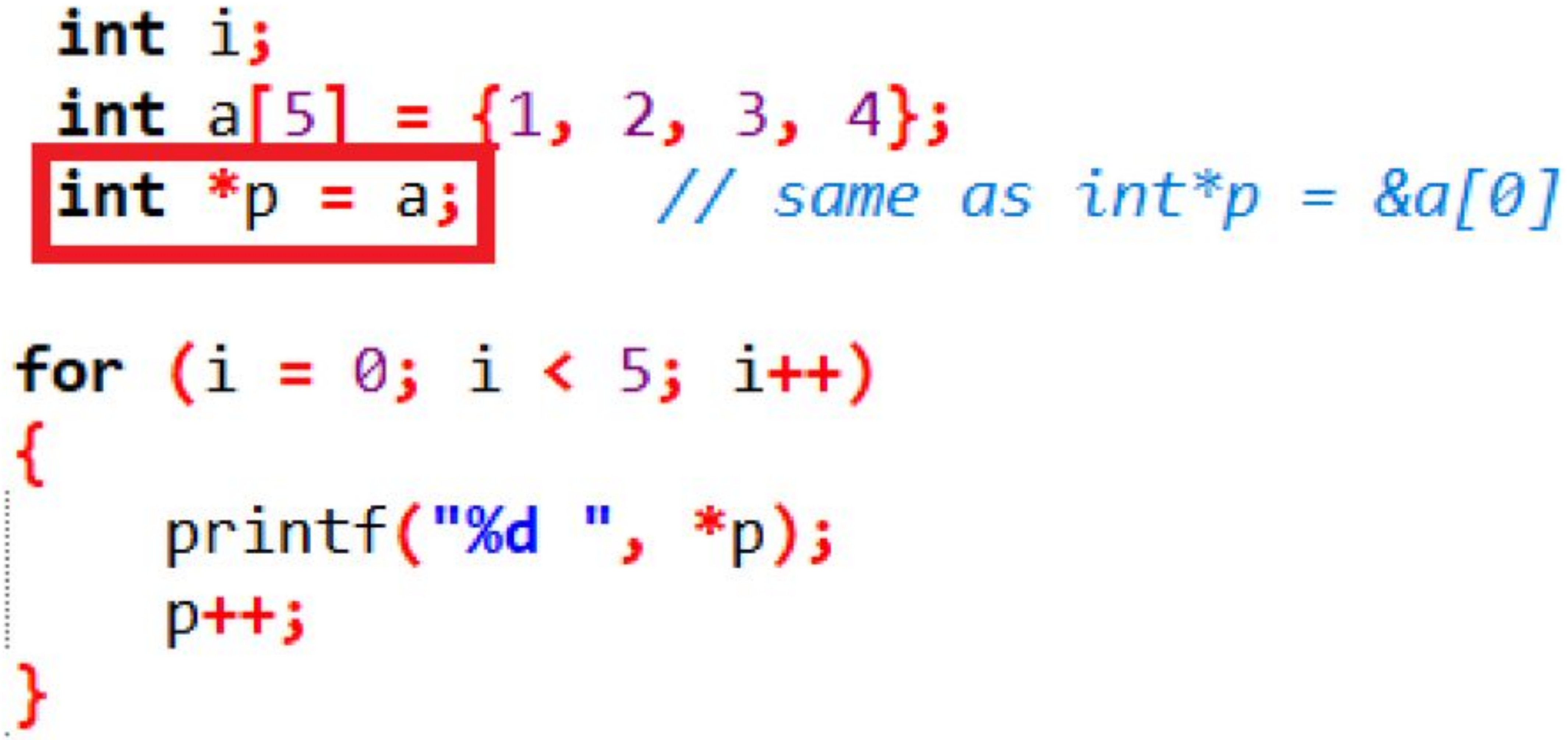
arr will give the base address, which is a constant pointer pointing to the first element of the array, arr[0].

Hence arr contains the address of arr[0] i.e 1000

Printf("address of array is =%u \n",arr);

Printf("address of array is =%u \n",&arr[0]);

Pointer to an Array (Examples)



```
int i;
int a[5] = {1, 2, 3, 4};
int *p = a;        // same as int*p = &a[0]

for (i = 0; i < 5; i++)
{
    printf("%d ", *p);
    p++;
}
```

Pointer to an Array (Examples)

Replacing the **printf("%d", *p);** statement of above example, with below mentioned statements. Lets see what will be the result.

printf("%d", a[i]); $\longrightarrow$ **prints the array, by incrementing index**

printf("%d", i[a]); $\longrightarrow$ **this will also print elements of array**

printf("%d", a+i); $\longrightarrow$ **This will print address of all the array elements**

printf("%d", *(a+i)); $\longrightarrow$ **Will print value of array element.**

printf("%d", *a); $\longrightarrow$ **will print value of a[0] only**

a++; $\longrightarrow$ **Compile time error, we cannot change base address of the array.**

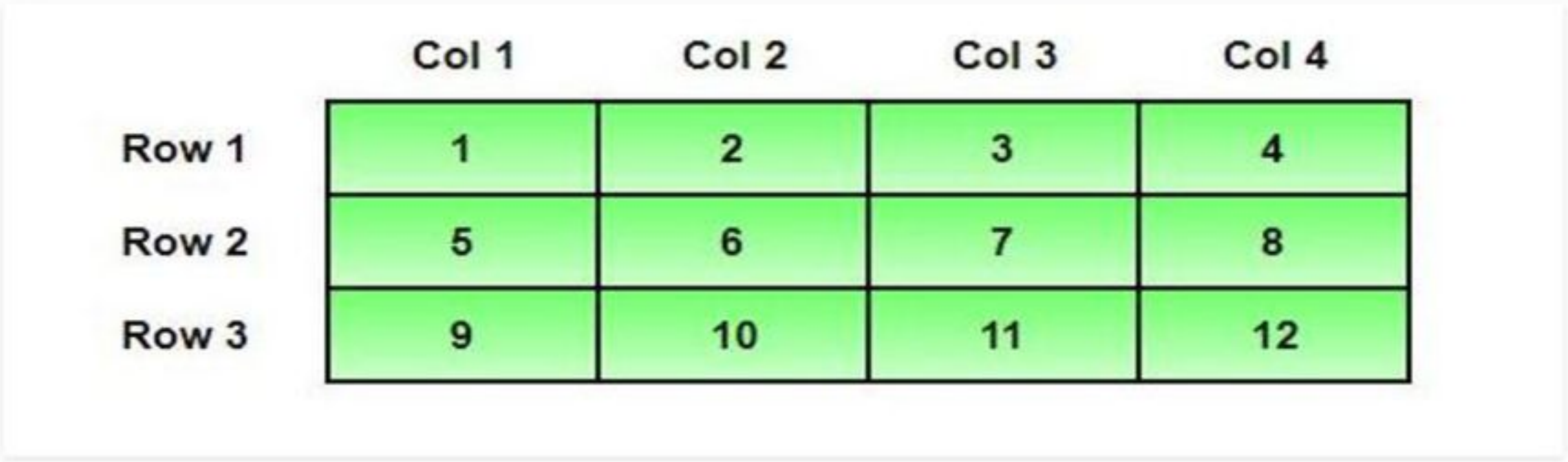
```
*(a+i)
```

is same as:

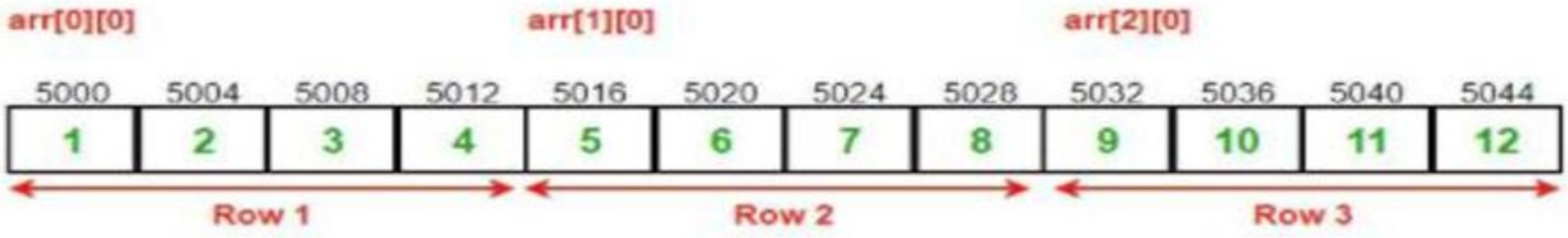
```
a[i]
```

Pointer to Multidimensional Arrav

```
int arr[3][4] = { {1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12} };
```



	Col 1	Col 2	Col 3	Col 4
Row 1	1	2	3	4
Row 2	5	6	7	8
Row 3	9	10	11	12



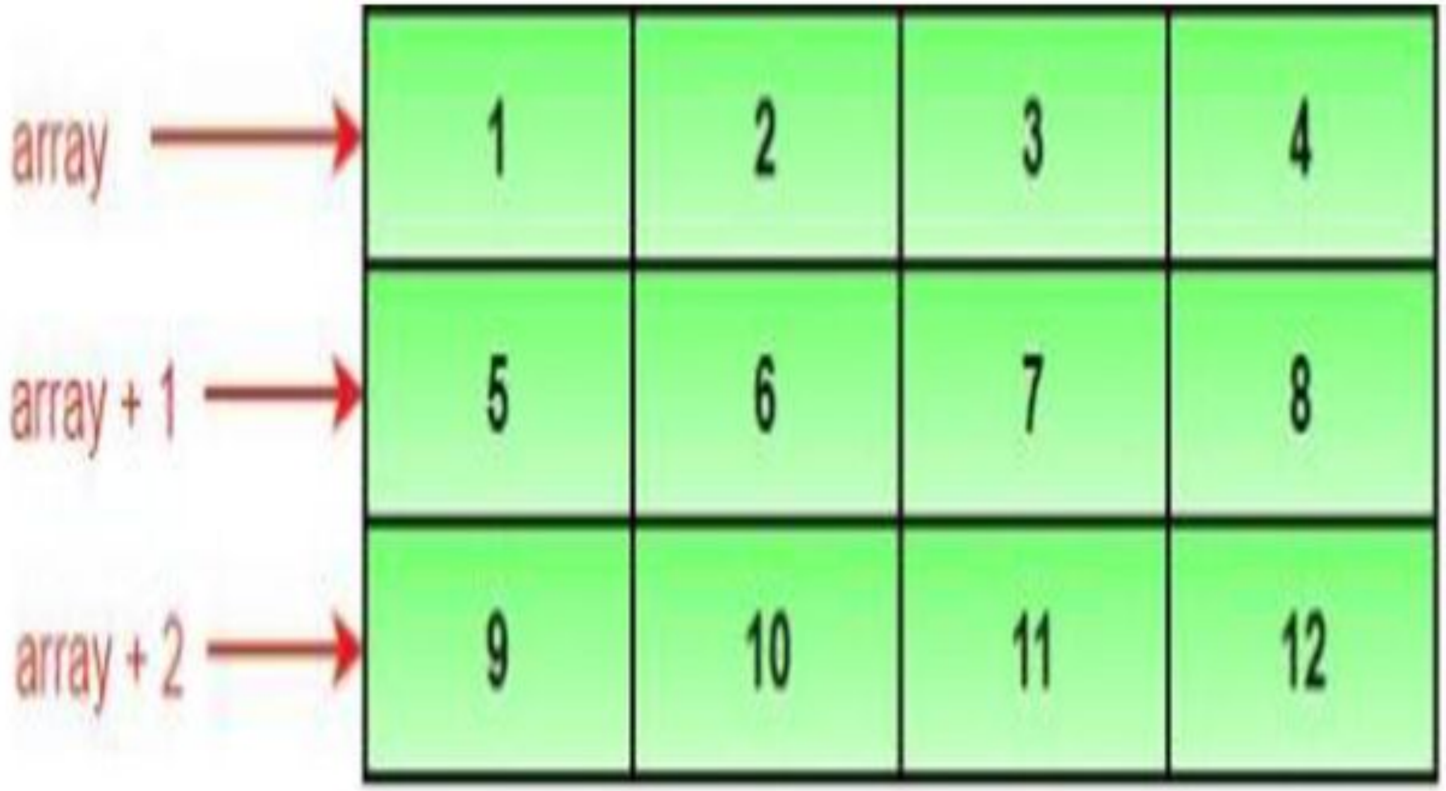
Pointer to Multidimensional Array

A multidimensional array is of form, a[i][j]
In a[i][j], a will give the base address of this array, even a + 0 + 0 will also give the base address, that is the address of a[0][0] element.

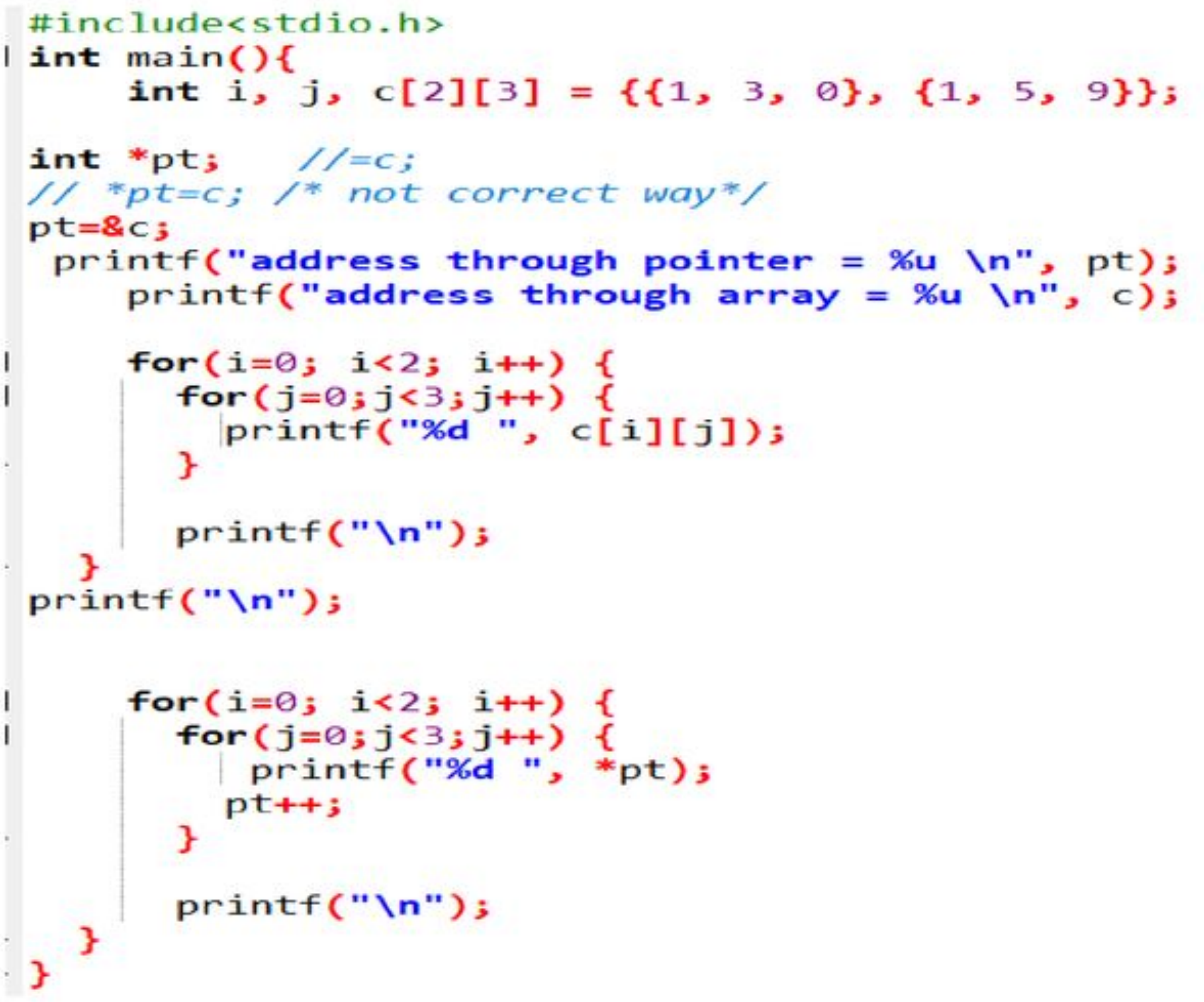
```
*(*(a + i) + j)
```

which is same as,

```
a[i][j]
```



Pointer to Multidimensional Array



```c
#include<stdio.h>
int main(){
    int i, j, c[2][3] = {{1, 3, 0}, {1, 5, 9}};

int *pt;   //=c;
// *pt=c; /* not correct way*/
pt=&c;
 printf("address through pointer = %u \n", pt);
    printf("address through array = %u \n", c);

    for(i=0; i<2; i++) {
      for(j=0;j<3;j++) {
        printf("%d ", c[i][j]);
      }

      printf("\n");
  }
printf("\n");


    for(i=0; i<2; i++) {
      for(j=0;j<3;j++) {
         printf("%d ", *pt);
        pt++;
      }

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

Passing Pointer to a function

Just like any other argument, pointers can also be passed to a function as an argument.

There are two ways to pass arguments/parameters to function calls –

- *call by value*
- *call by reference*.

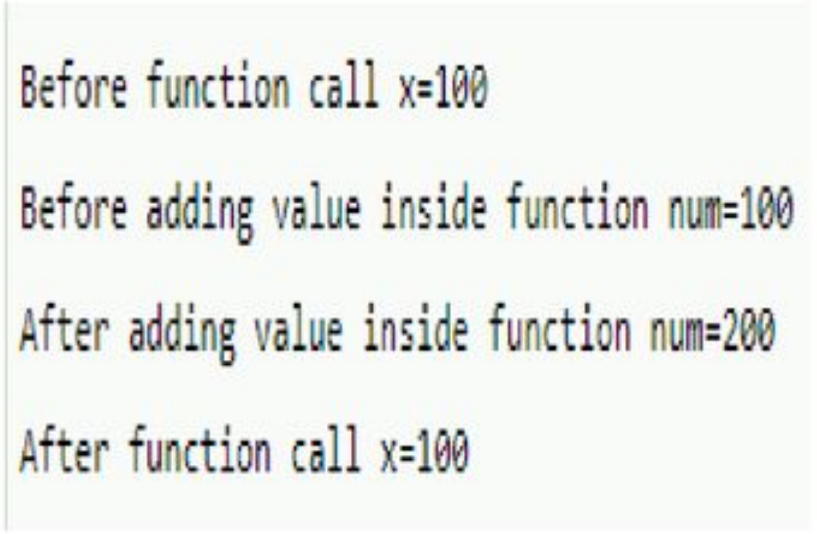
Call by values

In call by value, If you change the value of function parameter, it is changed for the current function only.

It will not change the value of variable inside the caller method such as main()

Call by value (Example)

```c
#include<stdio.h>
void change(int num) {
   printf("Before adding value inside function num=%d \n",num);
   num=num+100;
   printf("After adding value inside function num=%d \n", num);
}
int main() {
   int x=100;
   printf("Before function call x=%d \n", x);
   change(x);//passing value in function
   printf("After function call x=%d \n", x);
return 0;
}
```

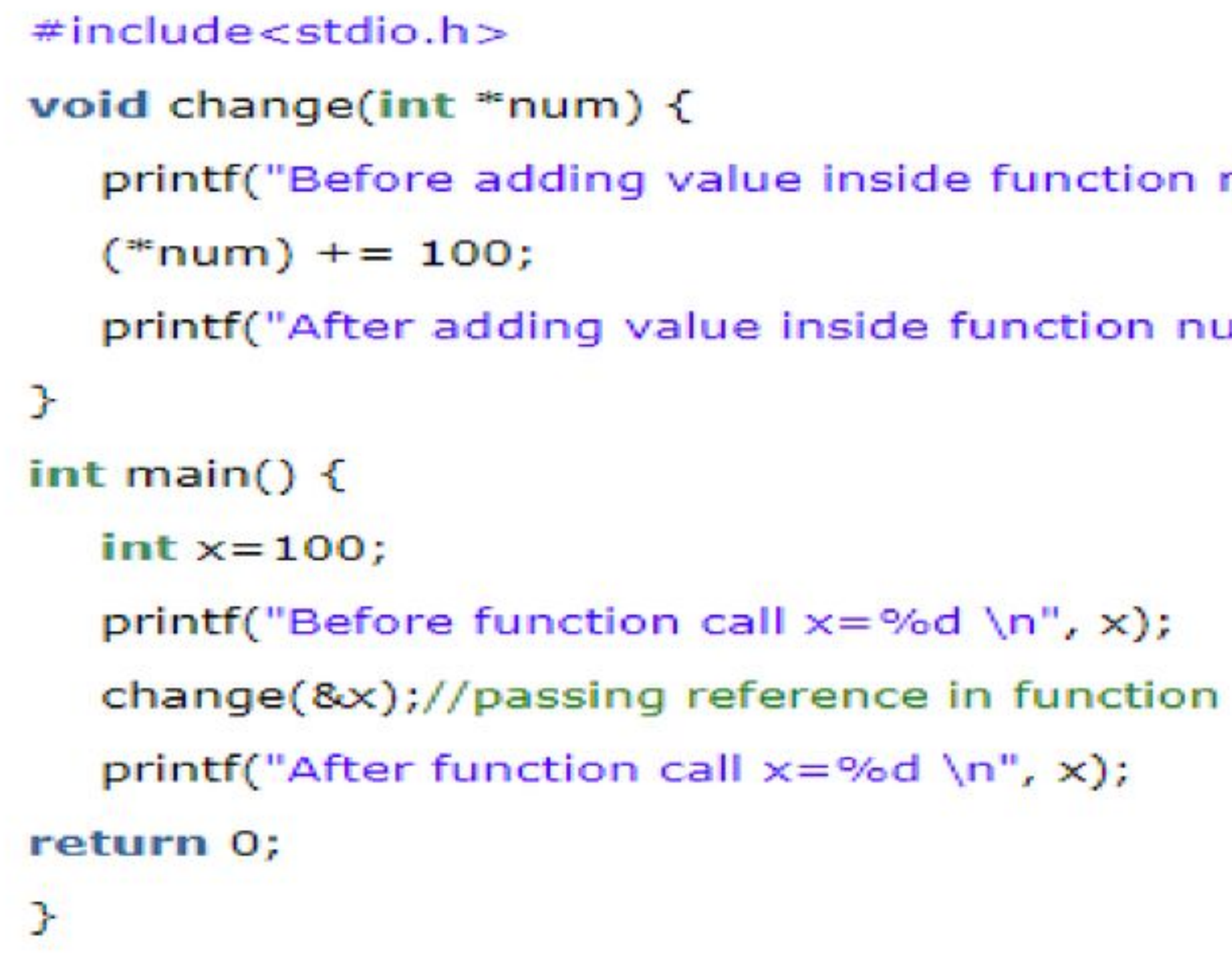


```
Before function call x=100

Before adding value inside function num=100

After adding value inside function num=200

After function call x=100
```

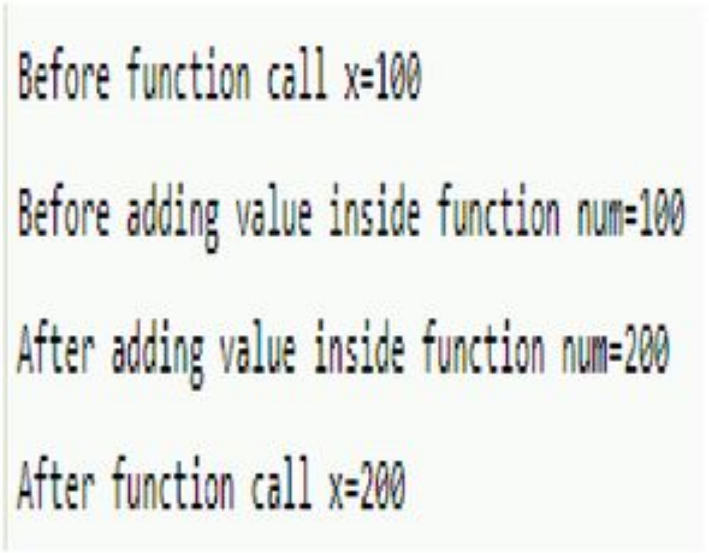
call by reference

- When we pass a pointer as an argument instead of a variable then the address of the variable is passed instead of the value.
- So actual and formal arguments shares the same address space. Hence, value changed inside the function, is reflected inside as well as outside the function.
- This technique is known as call by reference in C.

call by reference (Example)

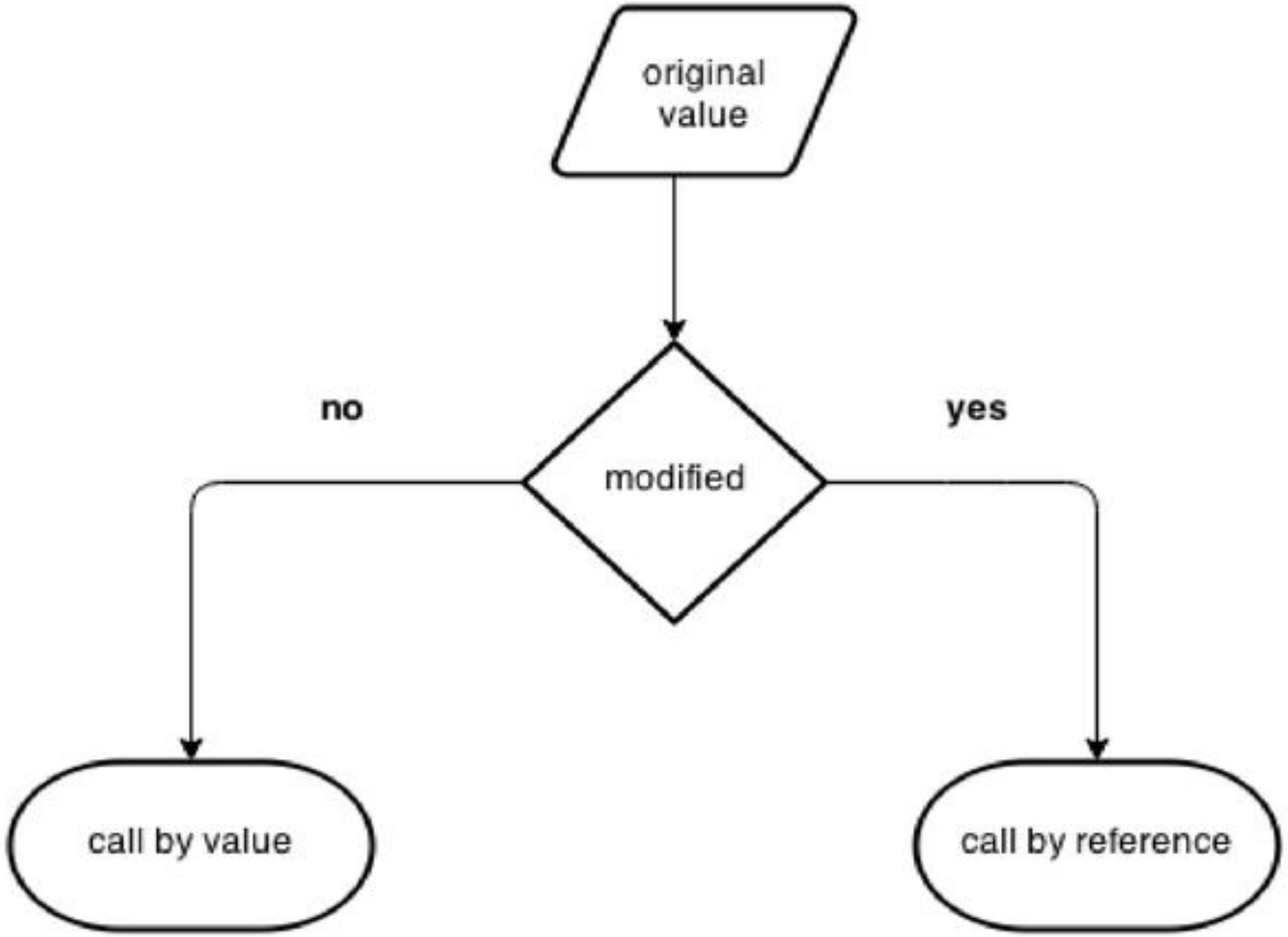


```
#include<stdio.h>
void change(int *num) {
   printf("Before adding value inside function num=%d \n",*num);
   (*num) += 100;
   printf("After adding value inside function num=%d \n", *num);
}
int main() {
   int x=100;
   printf("Before function call x=%d \n", x);
   change(&x);//passing reference in function
   printf("After function call x=%d \n", x);
return 0;
}
```



```
Before function call x=100

Before adding value inside function num=100

After adding value inside function num=200

After function call x=200
```

Difference between call by reference and call by reference (Example)



Difference between call by reference and call by reference (Example)

	Call by value	**Call by reference**
1	A copy of value is passed to the function	An address of value is passed to the function
2	Changes made inside the function is not reflected on other functions	Changes made inside the function is reflected outside the function also
3	Actual and formal arguments will be created in different memory location	Actual and formal arguments will be created in same memory location

Assignments

1. Write a c program for Student using functions concept with following parameter
   a) Take Name and marks as arrays
   b) Create a function **AvgM** to calculate the average of 5 marks where were given in array
   c) Create a function **displayS** to display student name and average marks and their grade
2. Write a c program for finding multiplication of two matrix using functions
3. Write a c program for sum of array elements using pointer to array concepts
4. Write a c program for sum two matrices using pointer to array concepts
5. Write a C program for swapping of two number using call by value and call by reference.