Pointers in C

By: Atiya Jokhio

Difference between pointers and arrays

- Assigning any address to an array variable is not allowed while address can be assigned to a pointer variable using address operator.
- A pointer is a place in memory that keeps address of another place inside while an array is a single, pre allocated chunk of contiguous elements (all of the same type), fixed in size and location.
- Pointer is dynamic in nature. The memory allocation can be resized or freed later while the arrays are static in nature. Once memory is allocated, it cannot be resized or freed dynamically.

Passing Array Elements to a Function by reference

```c
#include <stdio.h>
int main()
{
    int array[]= {10,20,30,40,50};
    display(&array[0],5);
}

display (int *j, int n)
{
    int i;
    for(i=0; i<n; i++)
    {

        printf("\nelement= %d", *j);
        j++; // increment pointer to point to next element
    }}
```

Passing Array Elements to a Function by reference

```c
#include <stdio.h>
#include <string.h>

void Sentence(char *arr)
{
    int i;
    int n = strlen(arr);

    printf("n = %d\n", n);

    for (i=0; i<n; i++)
      printf("%c", arr[i]);
}

// Driver program
int main()
{
    char arr[] = "Fast is National University";
    Sentence(arr);
    return 0;
}
```

Pointer to Pointer (Double Pointer)

- We already know that a pointer points to a location in memory and thus used to store the address of variables. So, when we define a pointer to pointer.
- The first pointer is used to store the address of the variable. And the second pointer is used to store the address of the first pointer. That is why they are also known as double pointers.

Pointer to Pointer (Double Pointer)

Pointer to pointer declaration:

Syntax:

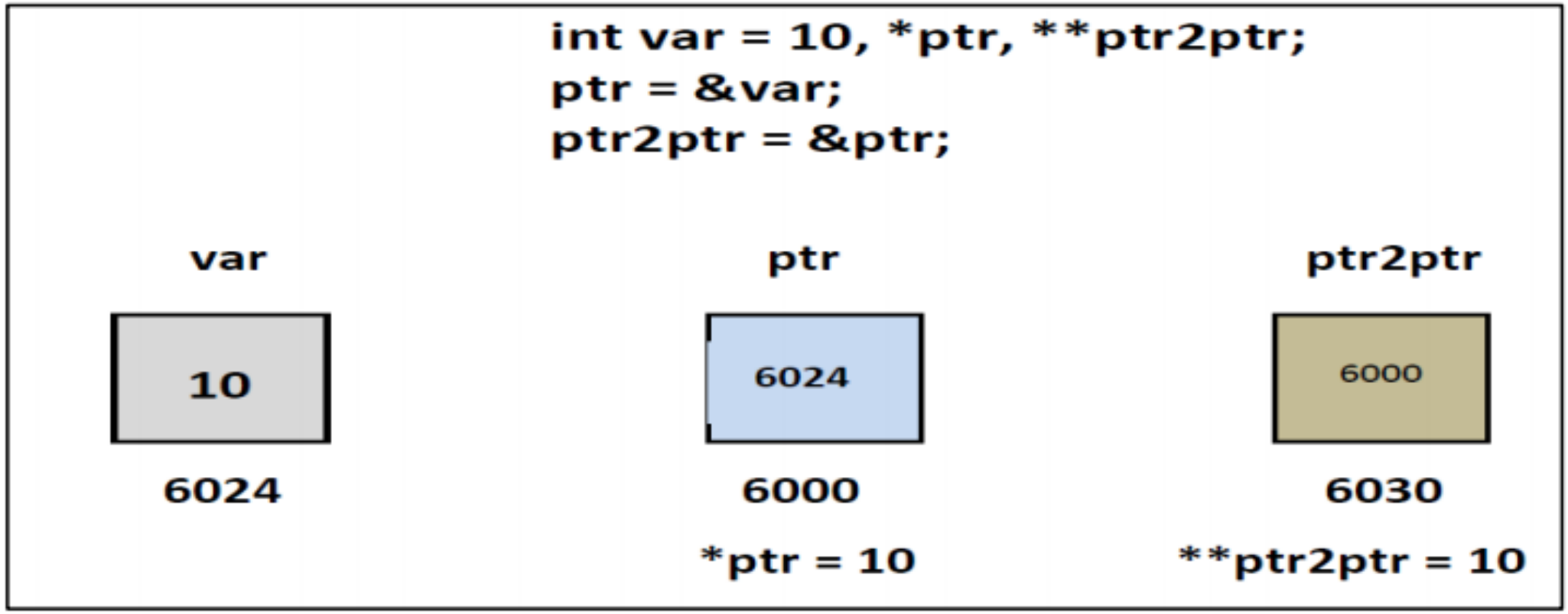
type ** variable ;

Example:

int **ptr2ptr;

Pointer to Pointer (Double Pointer)

INTERPRETATION:
The value of the pointer variable ptr2ptr is a memory address of another pointer.



Pointer to Pointer (Double Pointer)

Example:

```c
#include <stdio.h>
int main() {
    int  a = 10;
    int  *p1;        //this can store the address of variable a
    int  **p2;
    /*
        this can store the address of pointer variable p1 only.
        It cannot store the address of variable 'a'
    */
    p1 = &a;
    p2 = &p1;
    printf("Address of a = %u\n", &a);
    printf("Address of p1 = %u\n", &p1);
    printf("Address of p2 = %u\n\n", &p2);
    // below print statement will give the address of 'a'
    printf("Value at the address stored by p2 = %u\n", *p2);
    printf("Value at the address stored by p1 = %d\n", *p1);
    printf("Value of **p2 = %d\n", **p2); //read this *(*p2)
    return 0;
}
```

2D- Array using Pointers

- To access a two dimensional array using pointer, let us recall basics from one dimensional array. Since it is just an array of one dimensional array.

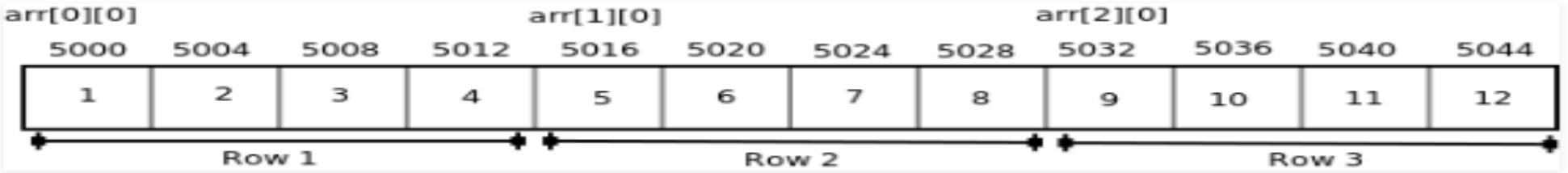
- Suppose I have a pointer array_ptr pointing at base address of one dimensional array. To access nth element of array using pointer we use *(array_ptr+n) (where array_ptr points to 0^{th} element of array, n is the nth element to access and nth element starts from 0).

2D- Array using Pointers

Let us take a two dimensional array *arr[3][4]*:
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

The following figure shows how the above 2-D array will be stored in memory



2D- Array using Pointers

- Now we know two dimensional array is array of one dimensional array. Hence let us see how to access a two dimensional array through pointer.
- Let us suppose a two-dimensional array

Int matrix[3][3];

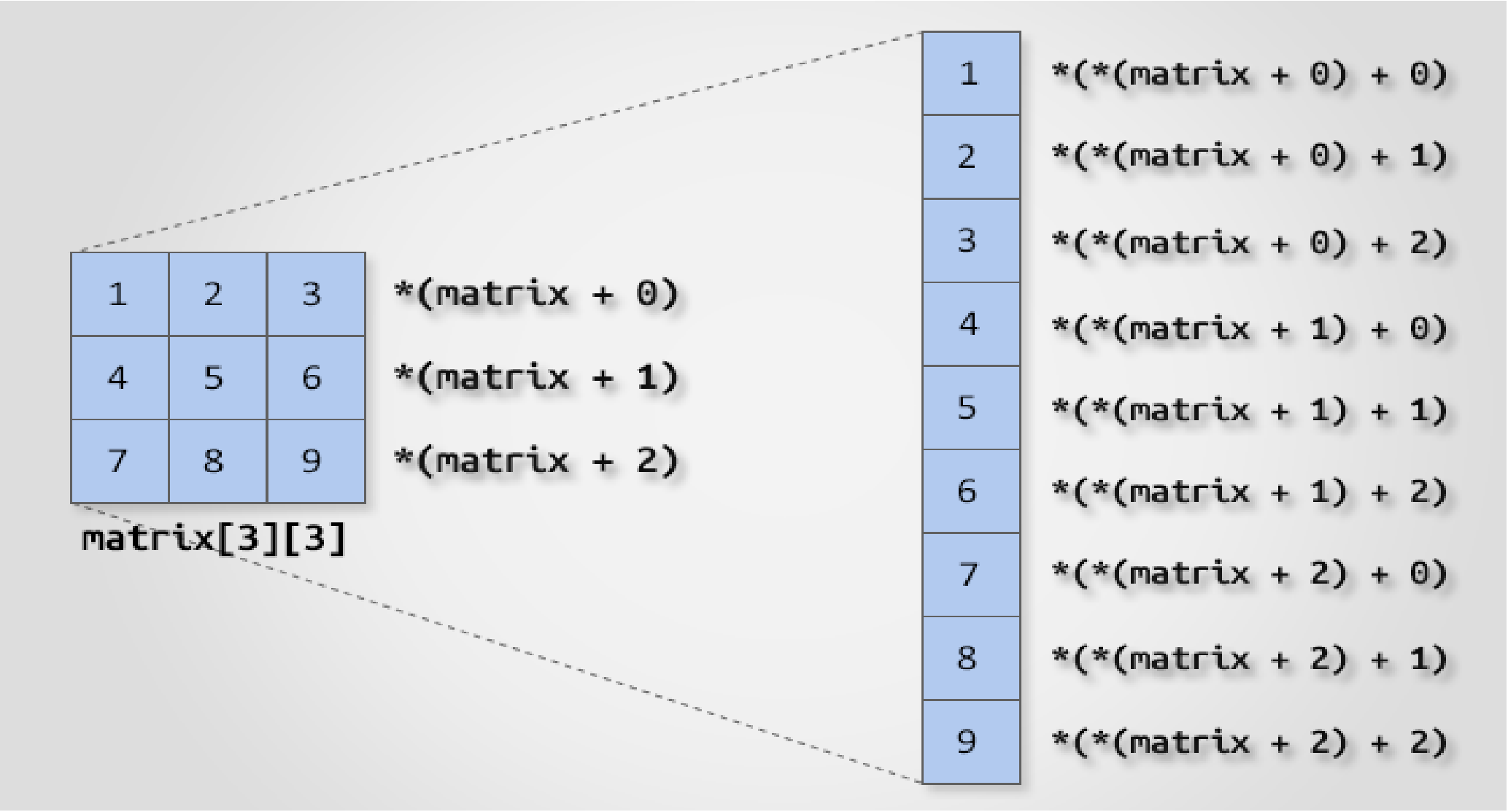
For the above array,

```
matrix                  =>    Points to base address of two-dimensional array.
                              Since array decays to pointer.

*(matrix)               =>    Points to first row of two-dimensional array.
*(matrix + 0)           =>    Points to first row of two-dimensional array.
*(matrix + 1)           =>    Points to second row of two-dimensional array.

**matrix                =>    Points to matrix[0][0]
*(*(matrix + 0))        =>    Points to matrix[0][0]
*(*(matrix + 0) + 0)    =>    Points to matrix[0][0]
*(*matrix + 1)          =>    Points to matrix[0][1]
*(*(matrix + 0) + 1)    =>    Points to matrix[0][1]
*(*(matrix + 2) + 2)    =>    Points to matrix[2][2]
```

2D- Array using Pointers



2D- Array using Pointers

Example:

```
#include <stdio.h>
int main( )
{
int s[4][2] = {
{ 1, 2 },
{ 3, 4 },
{ 5, 6 },
{ 7, 8 }
} ;
int i, j ;
for ( i = 0 ; i <= 3 ; i++ )
{
printf ( "\n" ) ;
//printf("Adress of i %u", *(s+i));
//printf ( "\n" ) ;
for ( j = 0 ; j <= 1 ; j++ )
{
//printf ( "Adress of j %d\n", ( *( s + i ) + j ) ) ;
printf ( "%d ", *( *( s + i ) + j ) ) ;
}}
}
```

Pointers to structures

- We have already learned that a pointer is a variable which points to the address of another variable of any data type like int, char, float etc.
- Similarly, we can have a pointer to structures, where a pointer variable can point to the address of a structure variable.
- Here is how we can declare a pointer to a structure variable.

```
struct name {
member1;
member2; . . };

 int main()
{
struct name *ptr, Michal;
}
```

Pointers to structures

- Here is how we can declare a pointer to a structure variable.

```c
#include <stdio.h>

struct Book
{
    char name[10];
    int price;
}

int main()
{
    struct Book a;          //Single structure variable
    struct Book* ptr;       //Pointer of Structure type
    ptr = &a;

    struct Book b[10];      //Array of structure variables
    struct Book* p;         //Pointer of Structure type
    p = &b;

    return 0;
}
```

Pointers to structures

- **Accessing Structure Members with Pointer**

  To access members of structure using the structure variable, we used the dot . operator.
  But when we have a pointer of structure type, we use arrow -> to access structure members.
  Example:

```c
#include <stdio.h>
struct person
{
    int age;
    float weight;
};
int main(){
     struct person *personPtr, person1;
     personPtr = &person1;

     printf("Enter age: ");
     scanf("%d", &personPtr->age);

     printf("Enter weight: ");
     scanf("%f", &personPtr->weight);

     printf("Displaying:\n");
     printf("Age: %d\n", personPtr->age);
     printf("weight: %f", personPtr->weight);
     return 0;
}
```

Pointers to structures

- **Passing pointer structure to function**

```
# include <stdio.h>
struct book
{
char name[50] ;
char author[25] ;
int volno ;
} ;

main()
{
struct book b1 = { "Introduction to Computers", "Peter Norton", 100 } ;
display ( &b1 ) ;
}

display ( struct book *b )
{
 printf ( "\n%s \n%s \n%d", b->name, b->author, b->volno ) ;
}
```

Void Pointers

- In the c programming language, pointer to void is the concept of defining a pointer variable that is independent of data type.
- In C programming language, a void pointer is a pointer variable used to store the address of a variable of any datatype.
- That means single void pointer can be used to store the address of integer variable, float variable, character variable, double variable or any structure variable.
- We use the keyword **void** to create void pointer.

```
void  *ptr;
```

Void Pointers

```c
#include<stdio.h>

int main()
{
    int a ;
    float b ;
    char c ;

    void *ptr ;

    ptr = &a ;
    printf("Address of integer variable 'a' = %u\n", ptr) ;

    ptr = &b ;
    printf("Address of float variable 'b' = %u\n", ptr) ;

    ptr = &c ;
    printf("Address of character variable 'c' = %u\n", ptr) ;
    return 0;
}
```