## Pointer

The pointer is a variable which stores the address of another variable. This variable can be of type int, char, array, function or any other pointer.

**Declaring a Pointer**

int *a;//pointer to int

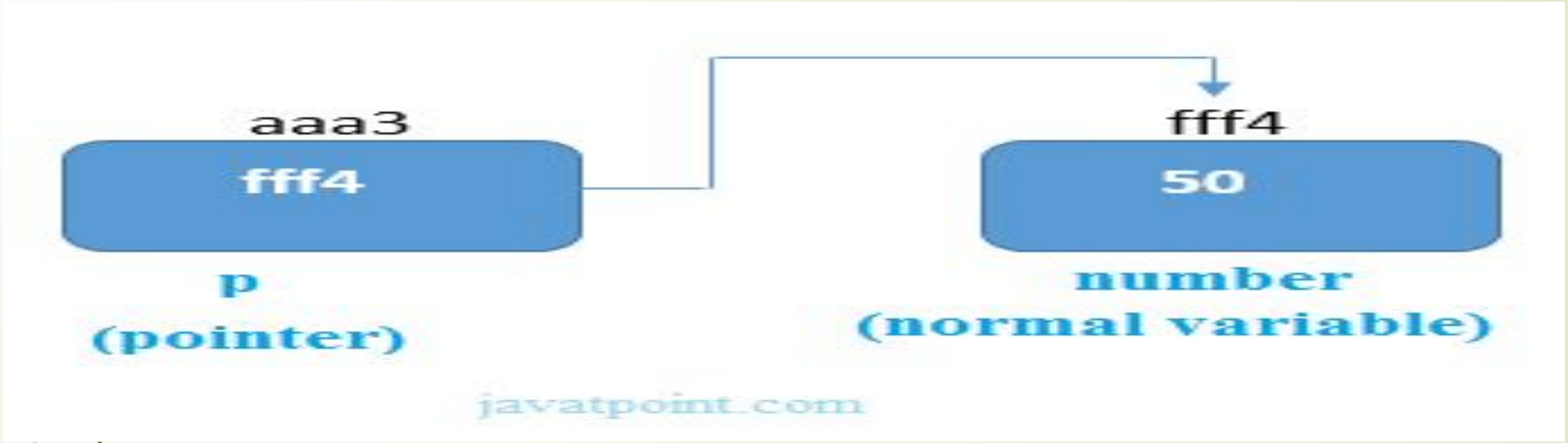
char *c;//pointer to char

**Pointer to array**

int arr[10];

int *p[10]=&arr; // Variable p of type pointer is pointing to the address of an integer array arr.

## Pointer



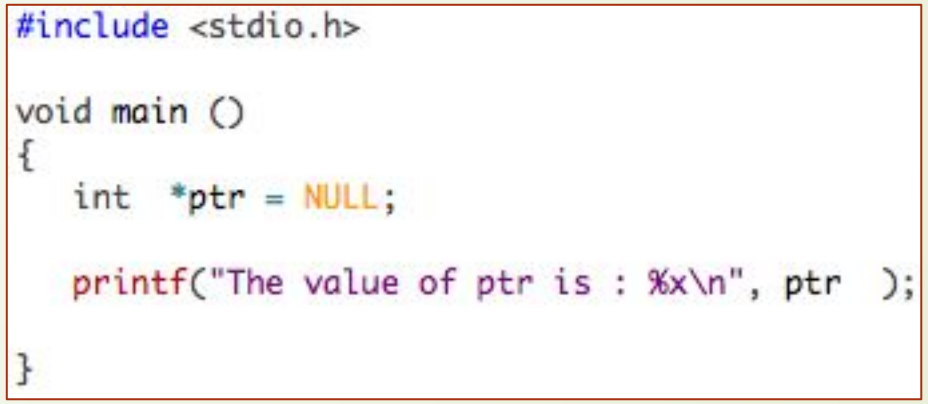
Pointer variable stores the address of number variable, i.e., fff4. The value of number variable is 50. But the address of pointer variable p is aaa3

## Example

```
#include<stdio.h>
int main()
{
int number=50;
int *p;
p=&number;//stores the address of number variable
printf("Address of p variable is %x \n",p); // p contains the address of the number therefore print
printf("Value of p variable is %d \n",*p); // As we know that * is used to dereference a pointer th
return 0;
}
```

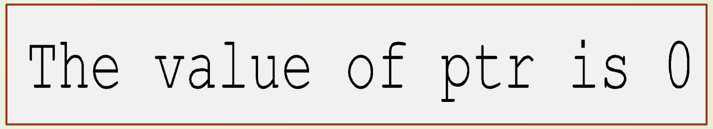
## NULL Pointers

-  It is always a good practice to assign a NULL value to a pointer variable in case you do not have exact address to be assigned. This is done at the time of variable declaration. A pointer that is assigned NULL is called a **null** pointer.
-  The NULL pointer is a constant with a value of zero defined in several standard libraries. Consider the following program:



```
#include <stdio.h>

void main ()
{
   int  *ptr = NULL;

   printf("The value of ptr is : %x\n", ptr  );

}
```

-  When the above code is compiled and executed, it produces the following result:



```
The value of ptr is 0
```

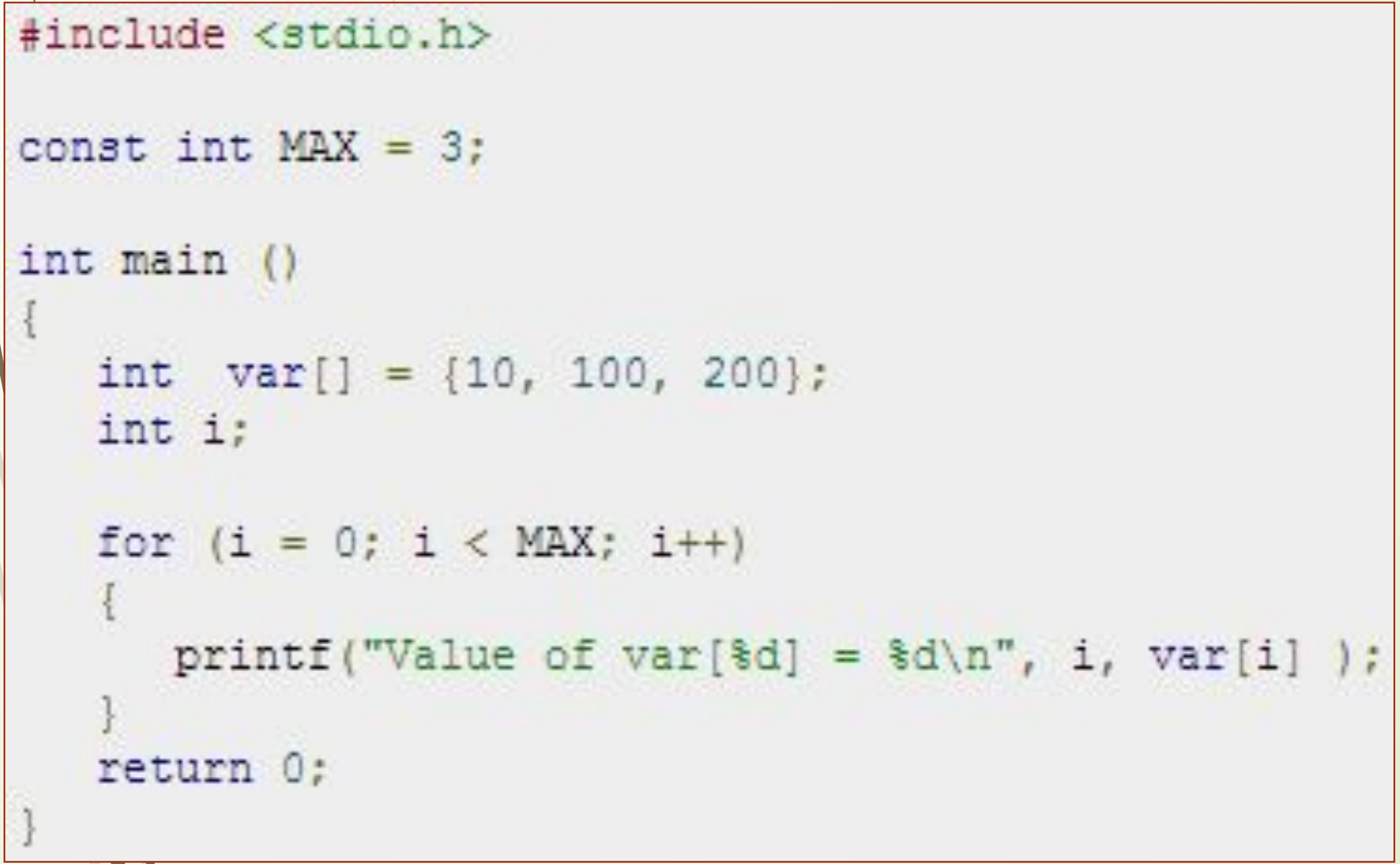
## NULL Pointers

- On most of the operating systems, programs are not permitted to access memory at address 0 because that memory is reserved by the operating system. However, the memory address 0 has special significance; it signals that the pointer is not intended to point to an accessible memory location. But by convention, if a pointer contains the null (zero) value, it is assumed to point to nothing.
- To check for a null pointer you can use an if statement as follows:

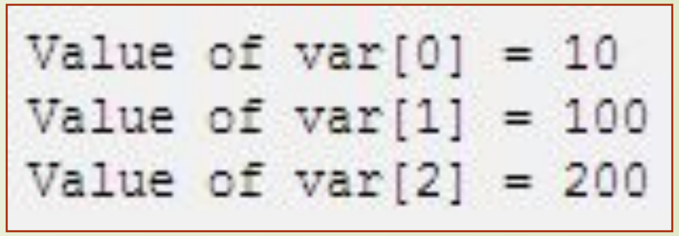
```
if(ptr)     /* succeeds if p is not null */
if(!ptr)    /* succeeds if p is null */
```

## Array of pointers

- Before we understand the concept of arrays of pointers, let us consider the following example, which makes use of an array of 3 integers:



```c
#include <stdio.h>

const int MAX = 3;

int main ()
{
   int  var[] = {10, 100, 200};
   int i;

   for (i = 0; i < MAX; i++)
   {
      printf("Value of var[%d] = %d\n", i, var[i] );
   }
   return 0;
}
```



```
Value of var[0] = 10
Value of var[1] = 100
Value of var[2] = 200
```

## Array of pointers

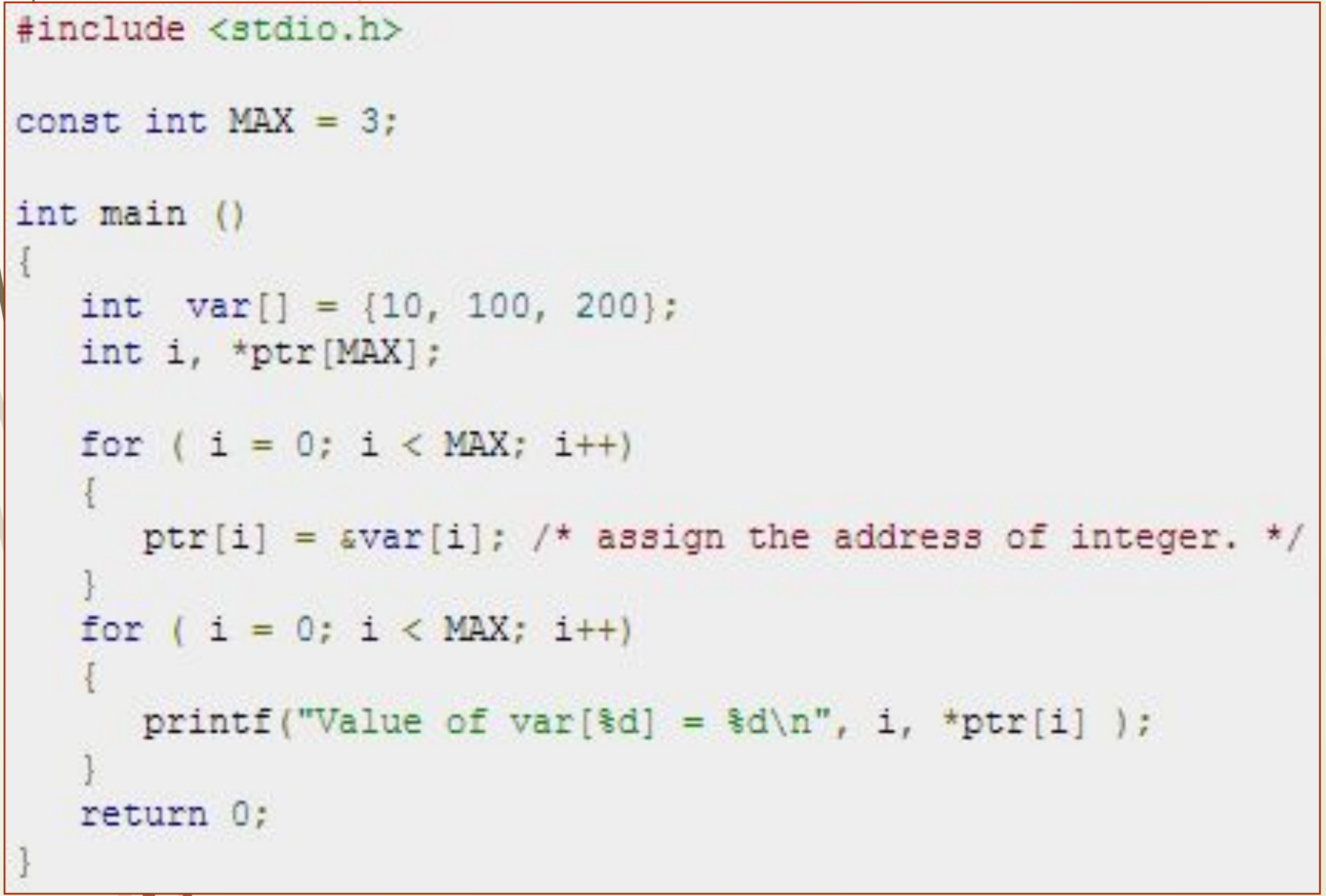
There may be a situation when we want to maintain an array, which can store pointers to an int or char or any other data type available. Following is the declaration of an array of pointers to an integer:

```
int *ptr[ ];
```

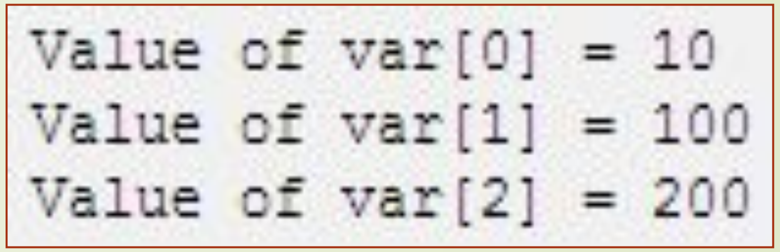
This declares **ptr** as an array of integer pointers. Thus, each element in ptr, now holds a pointer to an int value. Following example makes use of three integers, which will be stored in an array of pointers as follows:

## Array of pointers

Following example makes use of three integers, which will be stored in an array of pointers as follows:



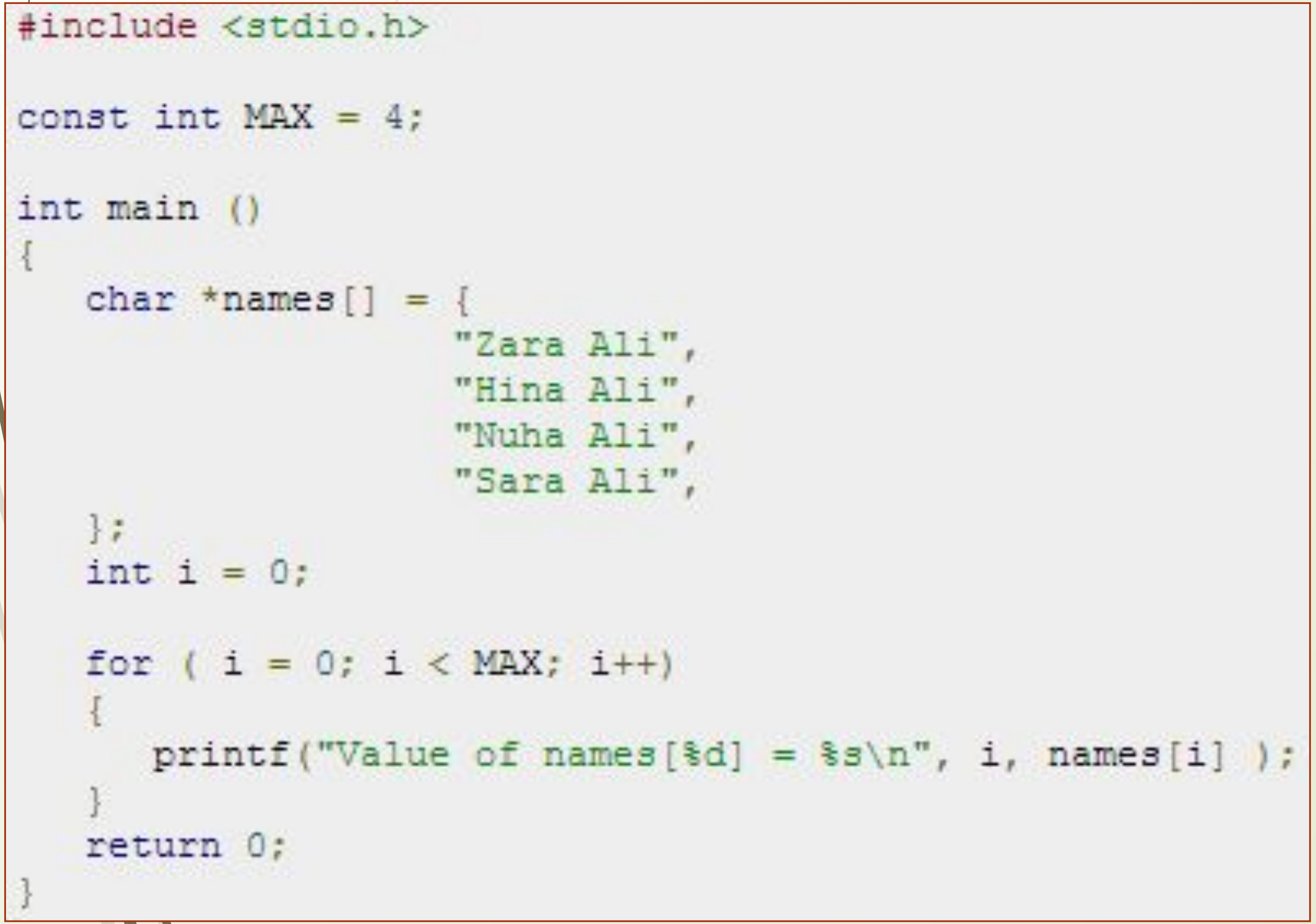
```
#include <stdio.h>

const int MAX = 3;

int main ()
{
   int  var[] = {10, 100, 200};
   int i, *ptr[MAX];

   for ( i = 0; i < MAX; i++)
   {
      ptr[i] = &var[i]; /* assign the address of integer. */
   }
   for ( i = 0; i < MAX; i++)
   {
      printf("Value of var[%d] = %d\n", i, *ptr[i] );
   }
   return 0;
}
```



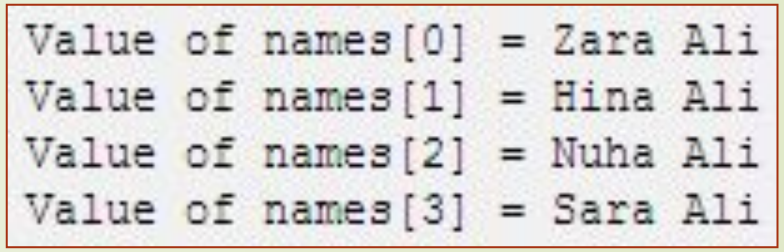
```
Value of var[0] = 10
Value of var[1] = 100
Value of var[2] = 200
```

## Array of pointers

- You can also use an array of pointers to character to store a list of strings as follows:



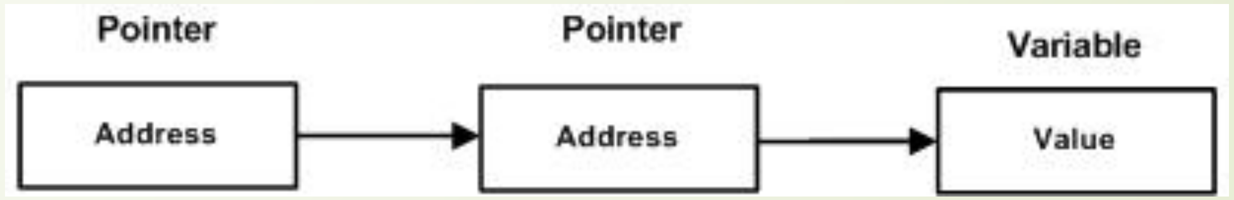
```c
#include <stdio.h>

const int MAX = 4;

int main ()
{
   char *names[] = {
                   "Zara Ali",
                   "Hina Ali",
                   "Nuha Ali",
                   "Sara Ali",
   };
   int i = 0;

   for ( i = 0; i < MAX; i++)
   {
      printf("Value of names[%d] = %s\n", i, names[i] );
   }
   return 0;
}
```



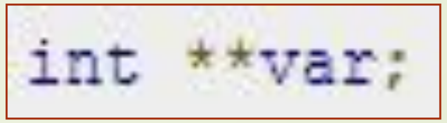
```
Value of names[0] = Zara Ali
Value of names[1] = Hina Ali
Value of names[2] = Nuha Ali
Value of names[3] = Sara Ali
```

## Pointer to Pointer

A pointer to a pointer is a form of multiple indirection, or a chain of pointers. Normally, a pointer contains the address of a variable. When we define a pointer to a pointer, the first pointer contains the address of the second pointer, which points to the location that contains the actual value as shown below.



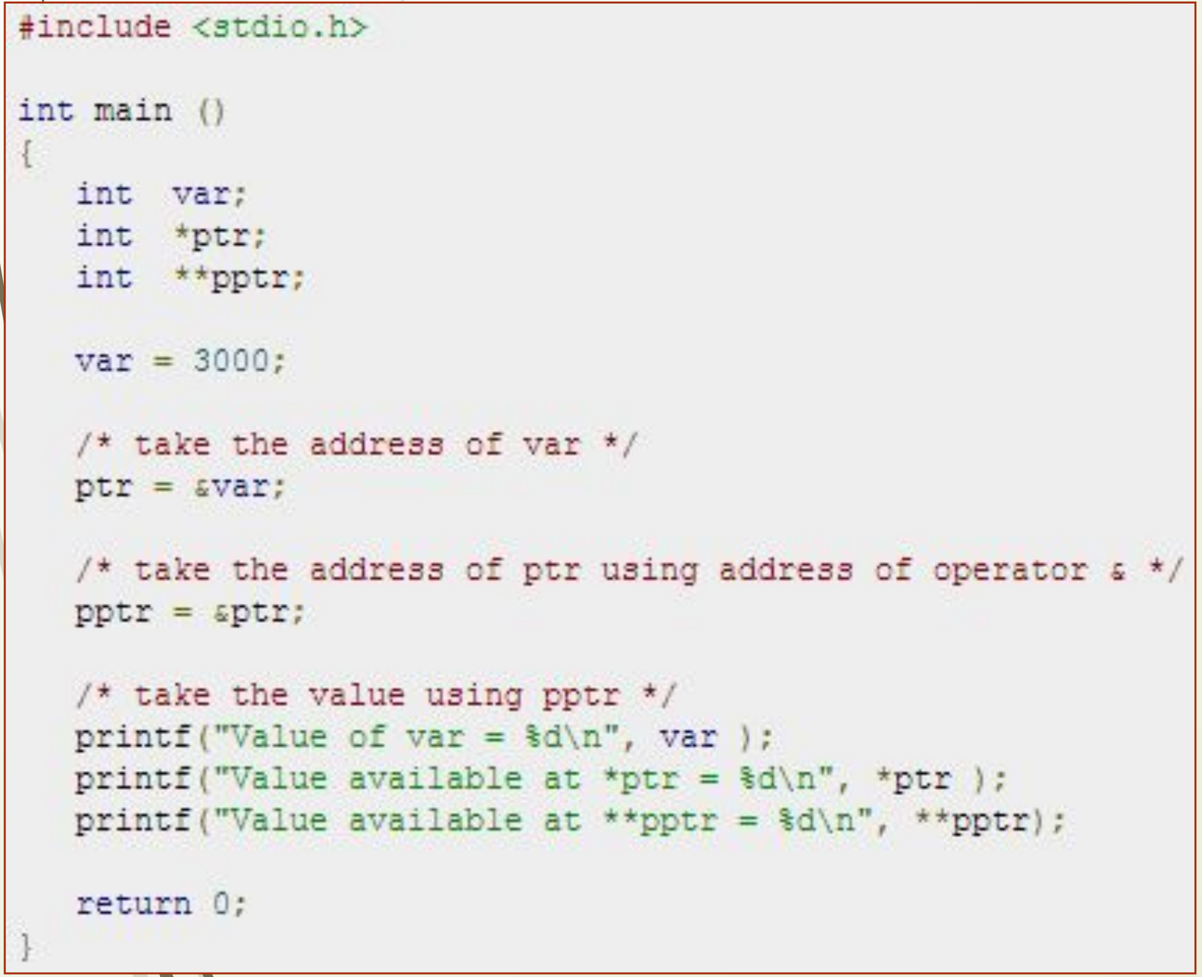
A variable that is a pointer to a pointer must be declared as such. This is done by placing an additional asterisk in front of its name. For example, following is the declaration to declare a pointer to a pointer of type *int*:



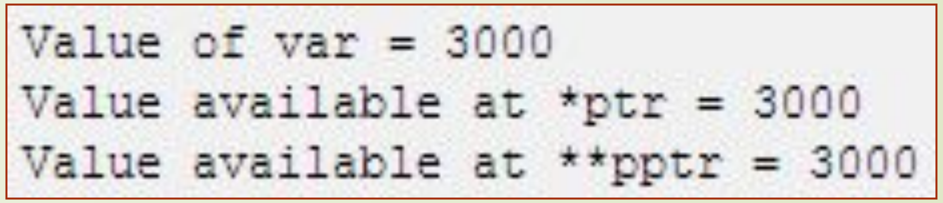
```
int **var;
```

## Pointer to Pointer

When a target value is indirectly pointed to by a pointer to a pointer, accessing that value requires that the asterisk operator be applied twice, as is shown below in the example:



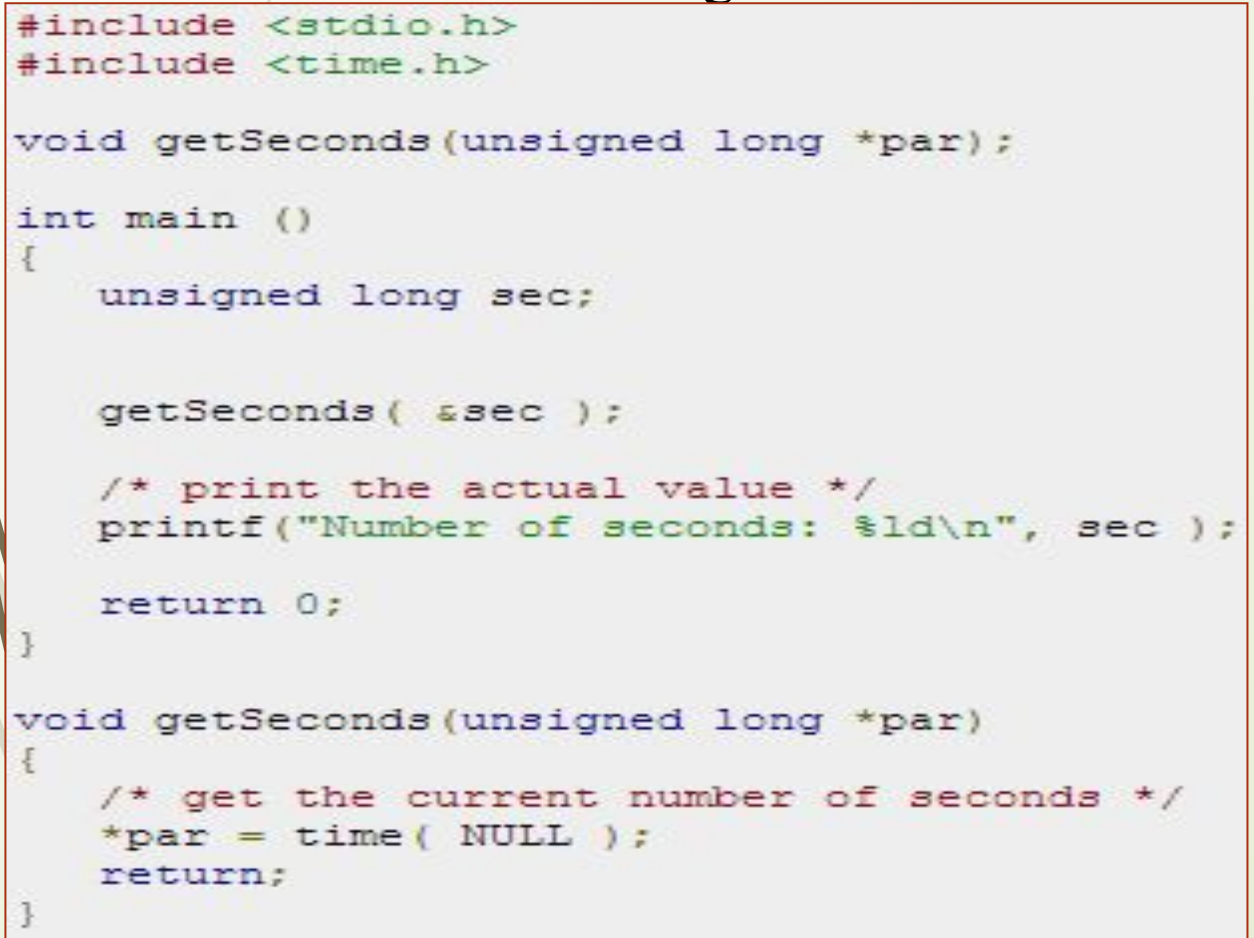
```c
#include <stdio.h>

int main ()
{
   int  var;
   int  *ptr;
   int  **pptr;

   var = 3000;

   /* take the address of var */
   ptr = &var;

   /* take the address of ptr using address of operator & */
   pptr = &ptr;

   /* take the value using pptr */
   printf("Value of var = %d\n", var );
   printf("Value available at *ptr = %d\n", *ptr );
   printf("Value available at **pptr = %d\n", **pptr);

   return 0;
}
```



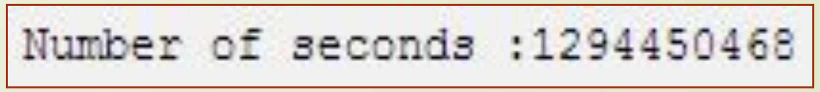
```
Value of var = 3000
Value available at *ptr = 3000
Value available at **pptr = 3000
```

## Passing pointers to functions

- C programming language allows you to pass a pointer to a function. To do so, simply declare the function parameter as a pointer type.
- Following is a simple example where we pass an unsigned long pointer to a function and change the value inside the function which reflects back in the calling



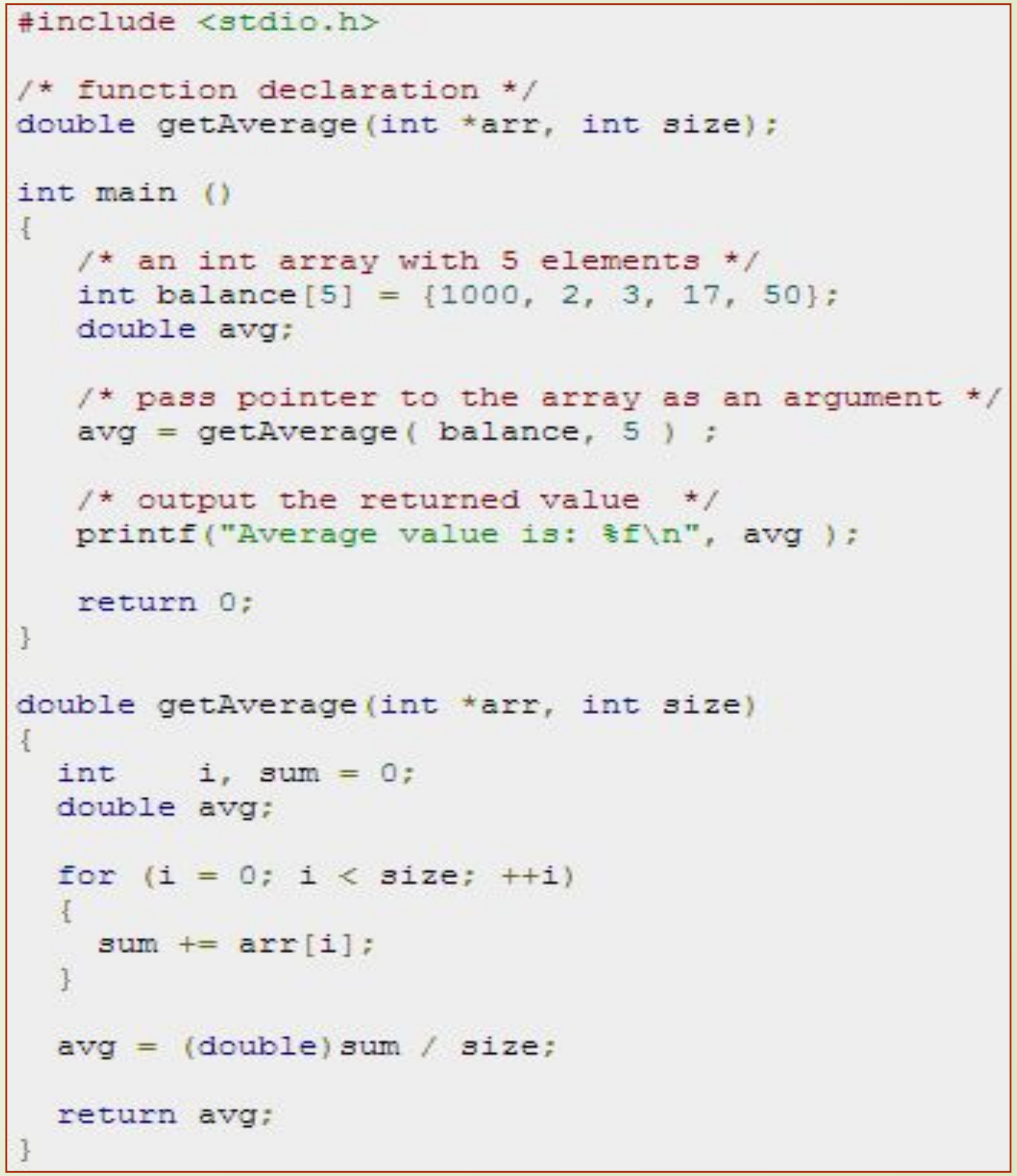
```c
#include <stdio.h>
#include <time.h>

void getSeconds(unsigned long *par);

int main ()
{
   unsigned long sec;


   getSeconds( &sec );

   /* print the actual value */
   printf("Number of seconds: %ld\n", sec );

   return 0;
}

void getSeconds(unsigned long *par)
{
   /* get the current number of seconds */
   *par = time( NULL );
   return;
}
```



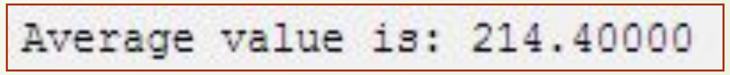
```
Number of seconds :1294450468
```

## Passing pointers to functions

The function, which can accept a pointer, can also accept an array as shown in the following example:



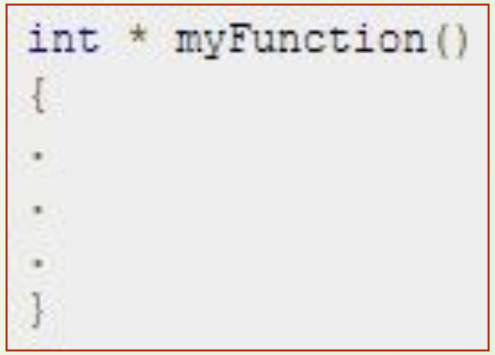
```c
#include <stdio.h>

/* function declaration */
double getAverage(int *arr, int size);

int main ()
{
   /* an int array with 5 elements */
   int balance[5] = {1000, 2, 3, 17, 50};
   double avg;

   /* pass pointer to the array as an argument */
   avg = getAverage( balance, 5 ) ;

   /* output the returned value  */
   printf("Average value is: %f\n", avg );

   return 0;
}

double getAverage(int *arr, int size)
{
  int    i, sum = 0;
  double avg;

  for (i = 0; i < size; ++i)
  {
    sum += arr[i];
  }

  avg = (double)sum / size;

  return avg;
}
```



```
Average value is: 214.40000
```

## Return pointer from functions

C allows us to return a pointer from a function. To do so, you would have to declare a function returning a pointer as in the following example:



```
int * myFunction()
{
.
.
.
}
```

Second point to remember is that, it is not good idea to return the address of a local variable to outside of the function so you would have to define the local variable as **static** variable.

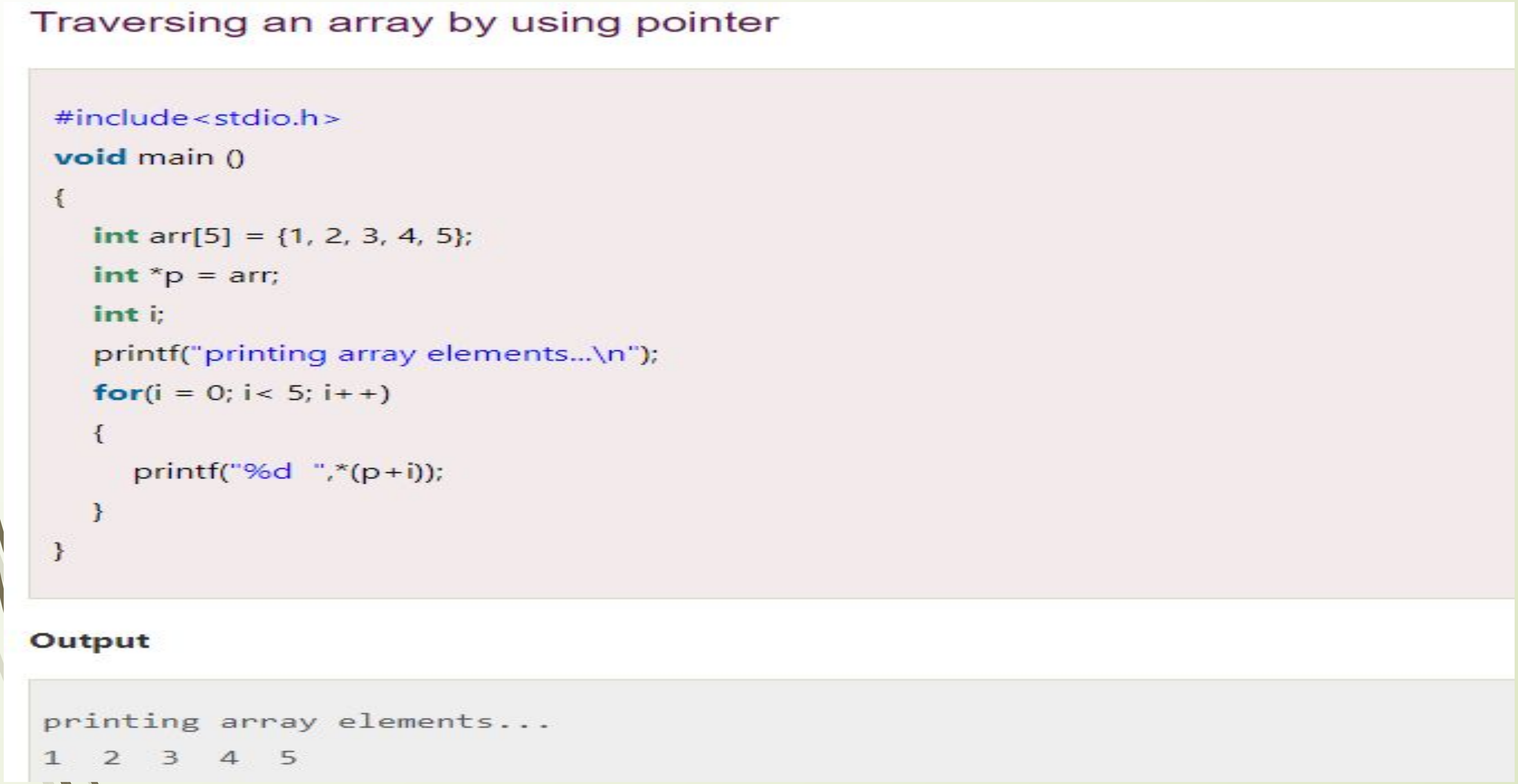
## Return pointer from functions

Consider the following function which will generate 10 random numbers and return them using an array name which represents a pointer i.e., address of first array element.

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

/* function to generate and retrun random numbers. */
int * getRandom( )
{
   static int  r[10];
   int i;

   /* set the seed */
   srand( (unsigned)time( NULL ) );
   for ( i = 0; i < 10; ++i)
   {
      r[i] = rand();
      printf("%d\n", r[i] );
   }

   return r;
}

/* main function to call above defined function */
int main ()
{
   /* a pointer to an int */
   int *p;
   int i;

   p = getRandom();
   for ( i = 0; i < 10; i++ )
   {
       printf("*(p + [%d]) : %d\n", i, *(p + i) );
   }

   return 0;
}
```

```
1523198053
1187214107
1108300978
430494959
1421301276
930971084
123250484
106932140
1604461820
149169022
*(p + [0]) : 1523198053
*(p + [1]) : 1187214107
*(p + [2]) : 1108300978
*(p + [3]) : 430494959
*(p + [4]) : 1421301276
*(p + [5]) : 930971084
*(p + [6]) : 123250484
*(p + [7]) : 106932140
*(p + [8]) : 1604461820
*(p + [9]) : 149169022
```

## Lab Task:
## Write a C program to Traversing an Array by using pointer



Traversing an array by using pointer

```
#include<stdio.h>
void main ()
{
    int arr[5] = {1, 2, 3, 4, 5};
    int *p = arr;
    int i;
    printf("printing array elements...\n");
    for(i = 0; i< 5; i++)
    {
        printf("%d  ",*(p+i));
    }
}
```

**Output**

```
printing array elements...
1   2   3   4   5
```

## Write a C program to swap two numbers without using the 3rd variable.

```
#include<stdio.h>
int main()
{
int a=10,b=20,*p1=&a,*p2=&b;
 printf("Before swap: *p1=%d *p2=%d",*p1,*p2);
*p1=*p1+*p2;
*p2=*p1-*p2;
*p1=*p1-*p2;
printf("\nAfter swap: *p1=%d *p2=%d",*p1,*p2);
 return 0;
}
```