Pointers

Akash Hegde

Seventh Sense Talent Solutions

Vivekananda Institute of Technology

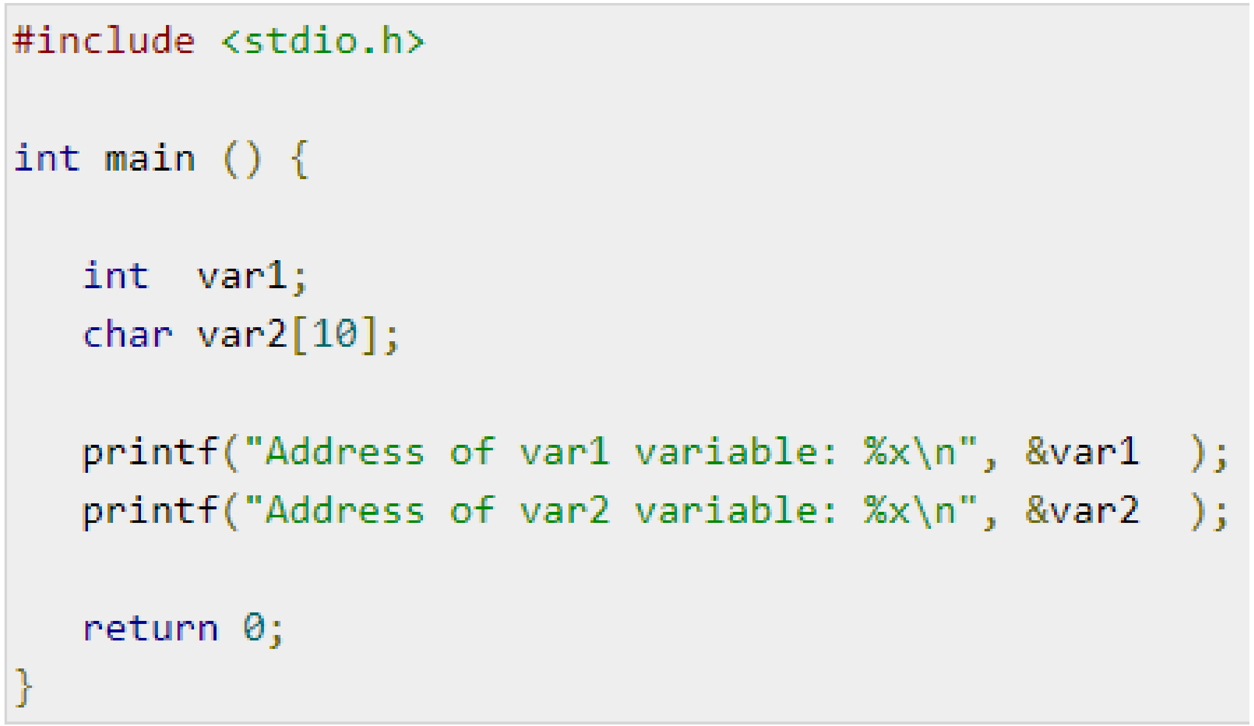
25 March 2021

Introduction to Pointers

- Some C programming tasks are performed more easily with pointers, and other tasks, such as dynamic memory allocation, cannot be performed without using pointers.
- Every variable is a memory location and every memory location has its address defined which can be accessed using ampersand (&) operator, which denotes an address in memory.

Introduction to Pointers

- Example (without pointer):



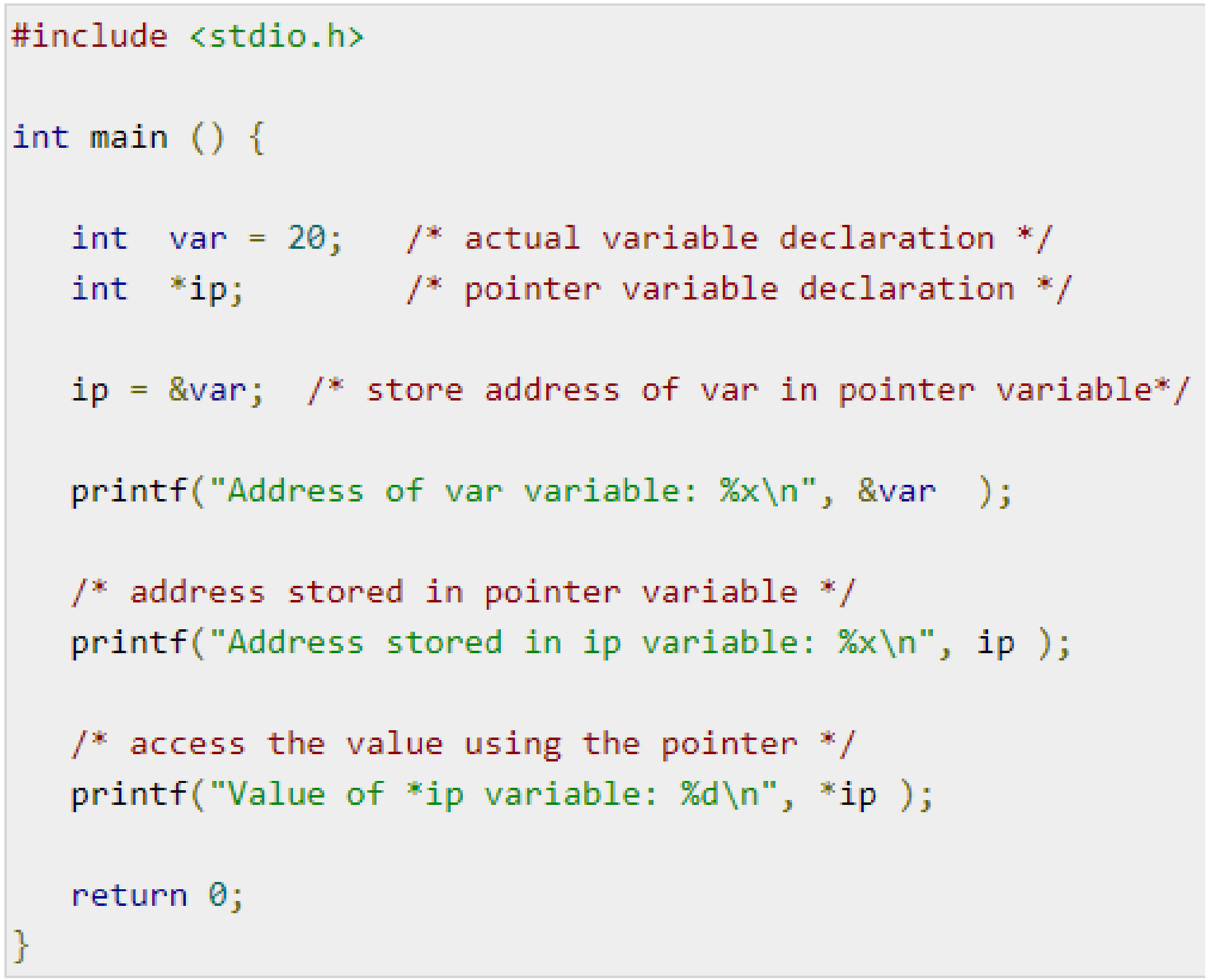
```c
#include <stdio.h>

int main () {

   int  var1;
   char var2[10];

   printf("Address of var1 variable: %x\n", &var1  );
   printf("Address of var2 variable: %x\n", &var2  );

   return 0;
}
```

Introduction to Pointers

- A **pointer** is a variable whose value is the address of another variable, i.e., direct address of the memory location.
- Like any variable or constant, you must declare a pointer before using it to store any variable address.
- Syntax:
  type *var_name;
- Here, **type** is the pointer's base type; it must be a valid C data type and **var-name** is the name of the pointer variable.

Introduction to Pointers

- Example (with pointer):



```c
#include <stdio.h>

int main () {

   int  var = 20;   /* actual variable declaration */
   int  *ip;        /* pointer variable declaration */

   ip = &var;  /* store address of var in pointer variable*/

   printf("Address of var variable: %x\n", &var  );

   /* address stored in pointer variable */
   printf("Address stored in ip variable: %x\n", ip );

   /* access the value using the pointer */
   printf("Value of *ip variable: %d\n", *ip );

   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 an 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.

NULL Pointers

- Example:

```
#include <stdio.h>

int main () {

   int  *ptr = NULL;

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

   return 0;
}
```

NULL Pointers

- In 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.

Pointer Arithmetic

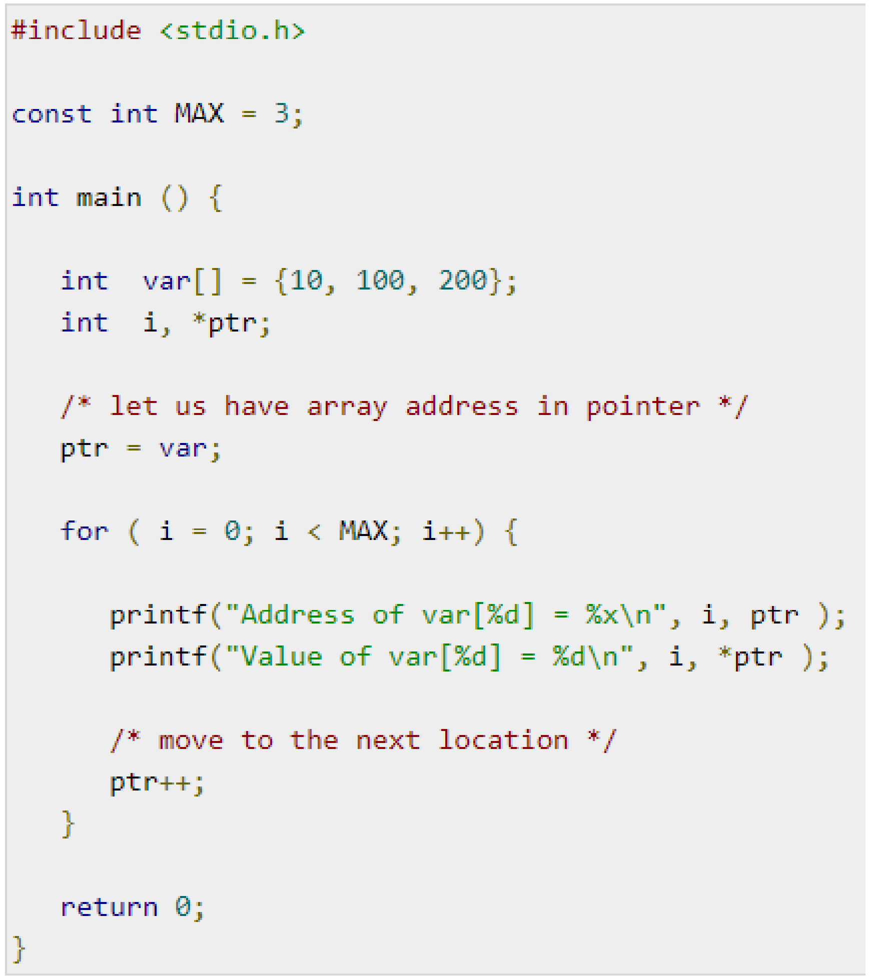
- A pointer in C is an address, which is a numeric value.
- Therefore, you can perform arithmetic operations on a pointer just as you can on a numeric value.
- There are four arithmetic operators that can be used on pointers: ++, --, +, and -
- Other than this, comparison operations can also be done with pointers.
- To understand pointer arithmetic, let us consider that **ptr** is an integer pointer which points to the address 1000.

Pointer Arithmetic

- Assuming 32-bit integers, let us perform the following arithmetic operation on the pointer: **`ptr++`**
- After the above operation, the **ptr** will point to the location 1004 because each time ptr is incremented, it will point to the next integer location which is 4 bytes next to the current location.
- This operation will move the pointer to the next memory location without impacting the actual value at the memory location.
- If **ptr** points to a character whose address is 1000, then the above operation will point to the location 1001 because the next character will be available at 1001.

Incrementing a Pointer

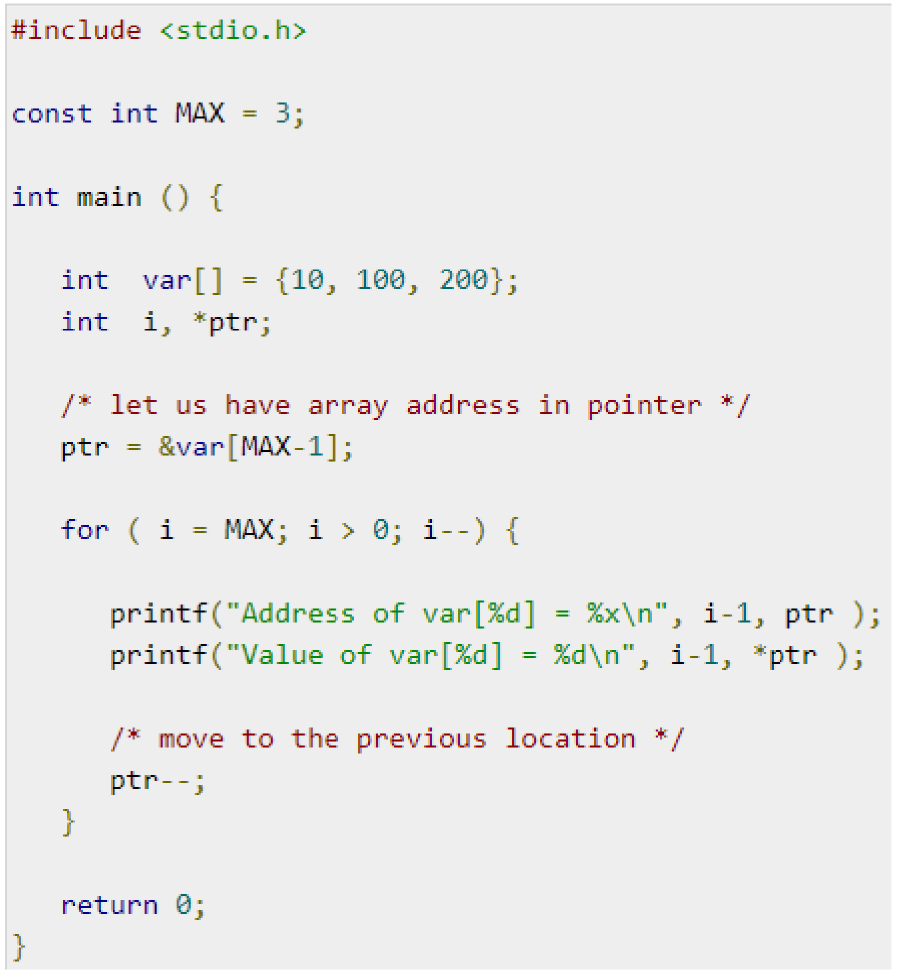
- Example to increment the variable pointer to access each succeeding element of the array:



```c
#include <stdio.h>

const int MAX = 3;

int main () {

   int  var[] = {10, 100, 200};
   int  i, *ptr;

   /* let us have array address in pointer */
   ptr = var;

   for ( i = 0; i < MAX; i++) {

      printf("Address of var[%d] = %x\n", i, ptr );
      printf("Value of var[%d] = %d\n", i, *ptr );

      /* move to the next location */
      ptr++;
   }

   return 0;
}
```

Decrementing a Pointer

- Example to decrement the variable pointer to access each preceding element of the array:

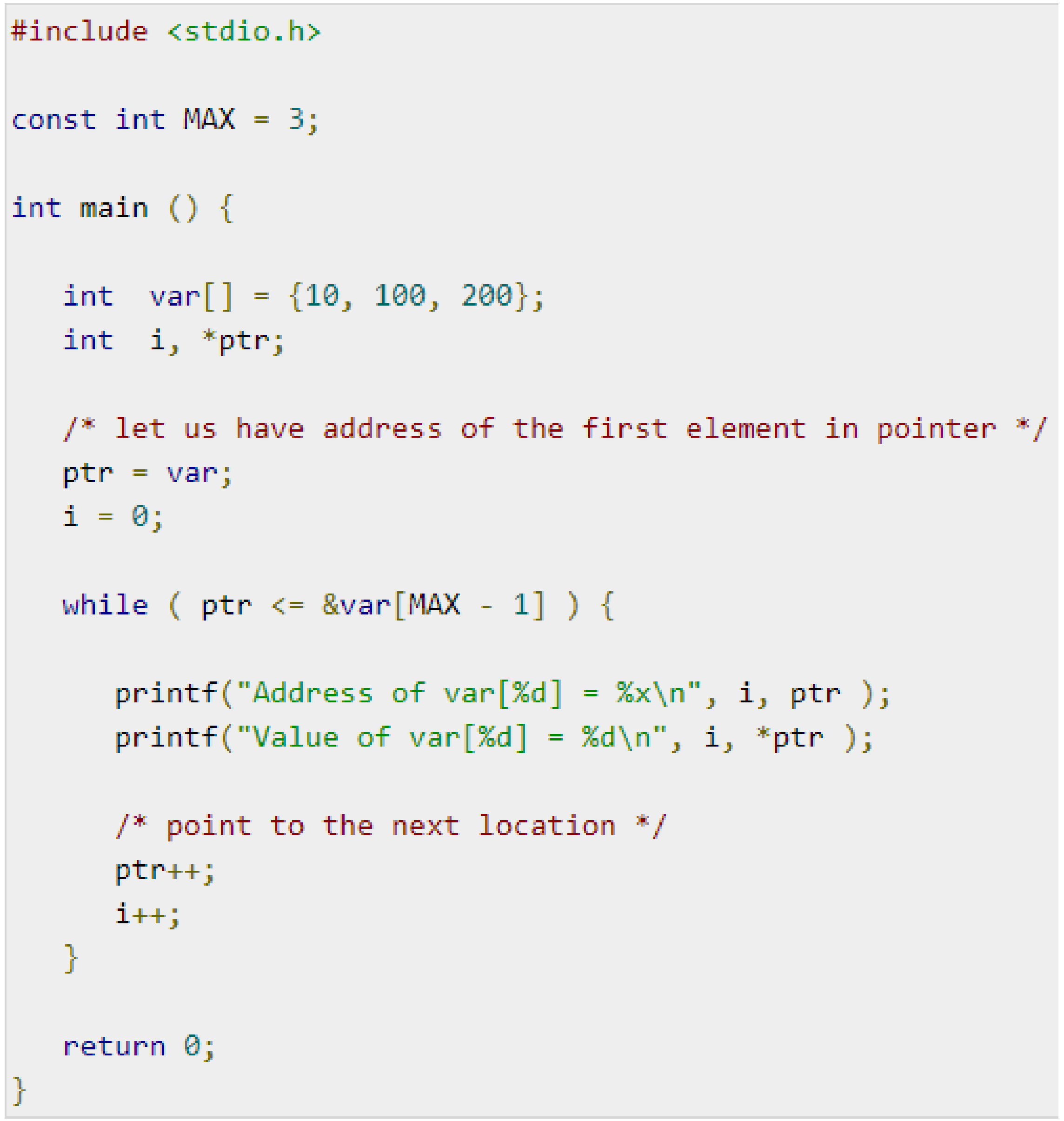


```c
#include <stdio.h>

const int MAX = 3;

int main () {

   int  var[] = {10, 100, 200};
   int  i, *ptr;

   /* let us have array address in pointer */
   ptr = &var[MAX-1];

   for ( i = MAX; i > 0; i--) {

      printf("Address of var[%d] = %x\n", i-1, ptr );
      printf("Value of var[%d] = %d\n", i-1, *ptr );

      /* move to the previous location */
      ptr--;
   }

   return 0;
}
```

Pointer Comparisons

- Pointers may be compared by using relational operators, such as ==, <, and >.
- If p1 and p2 point to variables that are related to each other, such as elements of the same array, then p1 and p2 can be meaningfully compared.
- Example to increment the variable pointer so long as the address to which it points is either less than or equal to the address of the last element of the array, which is &var[MAX - 1]:

Pointer Comparisons



```c
#include <stdio.h>

const int MAX = 3;

int main () {

   int  var[] = {10, 100, 200};
   int  i, *ptr;

   /* let us have address of the first element in pointer */
   ptr = var;
   i = 0;

   while ( ptr <= &var[MAX - 1] ) {

      printf("Address of var[%d] = %x\n", i, ptr );
      printf("Value of var[%d] = %d\n", i, *ptr );

      /* point to the next location */
      ptr++;
      i++;
   }

   return 0;
}
```

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[MAX];`
- It declares **`ptr`** as an array of MAX integer pointers. Thus, each element in ptr, holds a pointer to an int value.

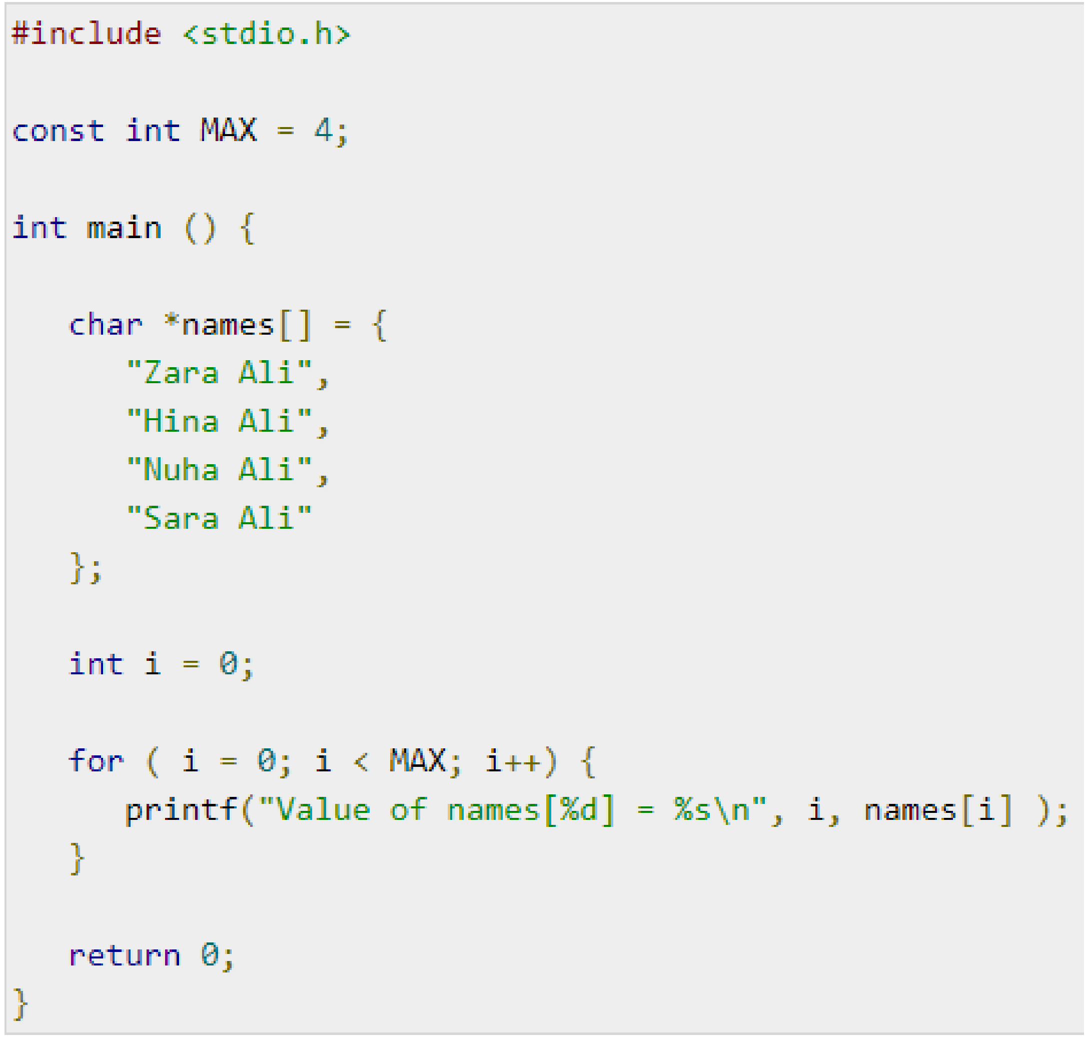
Array of Pointers

- Example to store integers in an array of pointers:

```
#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;
}
```

Array of Pointers

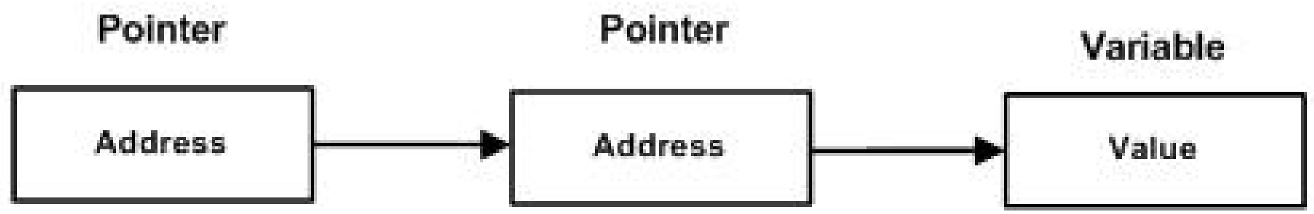
- Example to store list of strings in an array of pointers:



```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;
}
```

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.

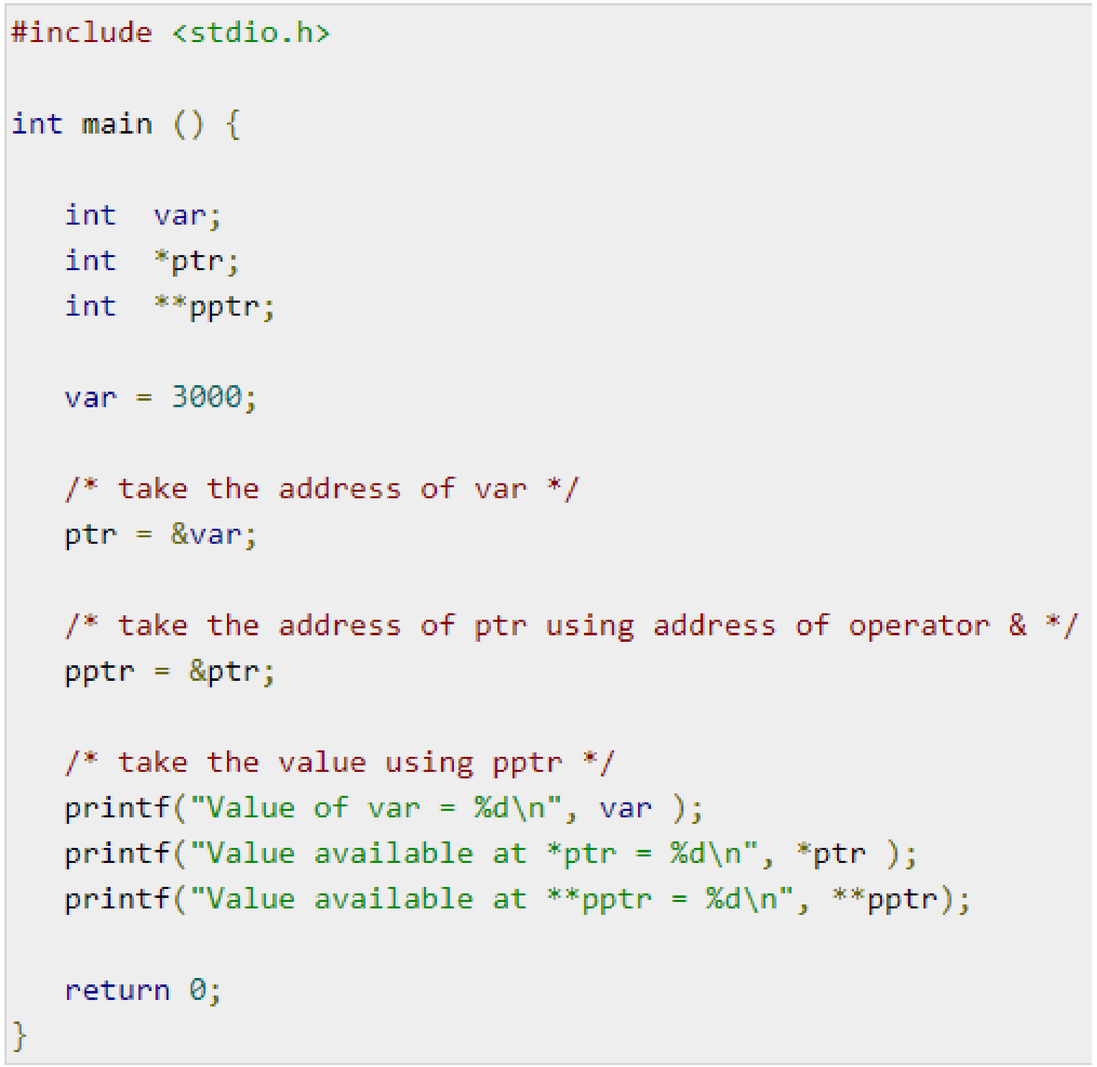


Pointer to Pointer

- 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, the following declaration declares a pointer to a pointer of type int –
  ```
  int **var;
  ```
- 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.

Pointer to Pointer

- Example:



```
#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;
}
```

Thank you!