4.6.ARRAY OF POINTERS

What is an Array of Pointers?

Just like an integer array holds a collection of integer variables, an **array of pointers** would hold variables of pointer type. It means each variable in an array of pointers is a pointer that points to another address.

The name of an array can be used as a pointer because it holds the address to the first element of the array. If we store the address of an array in another pointer, then it is possible to manipulate the array using pointer arithmetic.

Create an Array of Pointers

To create an array of pointers in C language, you need to declare an array of pointers in the same way as a pointer declaration. Use the data type then an asterisk sign followed by an identifier (array of pointers variable name) with a subscript ([]) containing the size of the array.

In an array of pointers, each element contains the pointer to a specific type.

Example of Creating an Array of Pointers

The following example demonstrates how you can create and use an array of pointers. Here, we are declaring three integer variables and to access and use them, we are creating an array of pointers. With the help of an array of pointers, we are printing the values of the variables.

```
#include <stdio.h>

int main() {
  // Declaring integers
  int var1 = 1;
  int var2 = 2;
  int var3 = 3;

  // Declaring an array of pointers to integers
  int *ptr[3];

  // Initializing each element of
  // array of pointers with the addresses of
  // integer variables
  ptr[0] = &var1;
  ptr[1] = &var2;
  ptr[2] = &var3;

  // Accessing values
  for (int i = 0; i < 3; i++) {
    printf("Value at ptr[%d] = %d\n", i, *ptr[i]);
  }

  return 0;
}
```

Output

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

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

There may be a situation when we want to maintain an array that can store pointers to an "int" or "char" or any other data type available.

(a) An Array of Pointers to Integers

Here 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.

Example

The following example uses three integers, which are 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. */
```

Output

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

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

b) <u>An Array of Pointers to Characters</u>

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

```
#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;
}
```

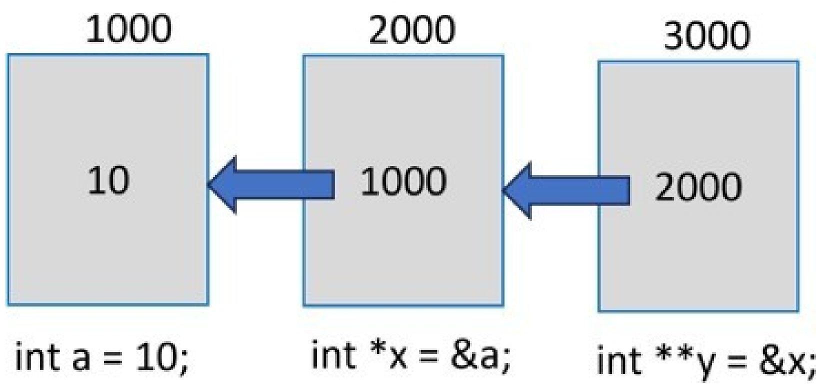
Output

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

```
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

We may have a pointer variable that stores the address of another pointer itself.



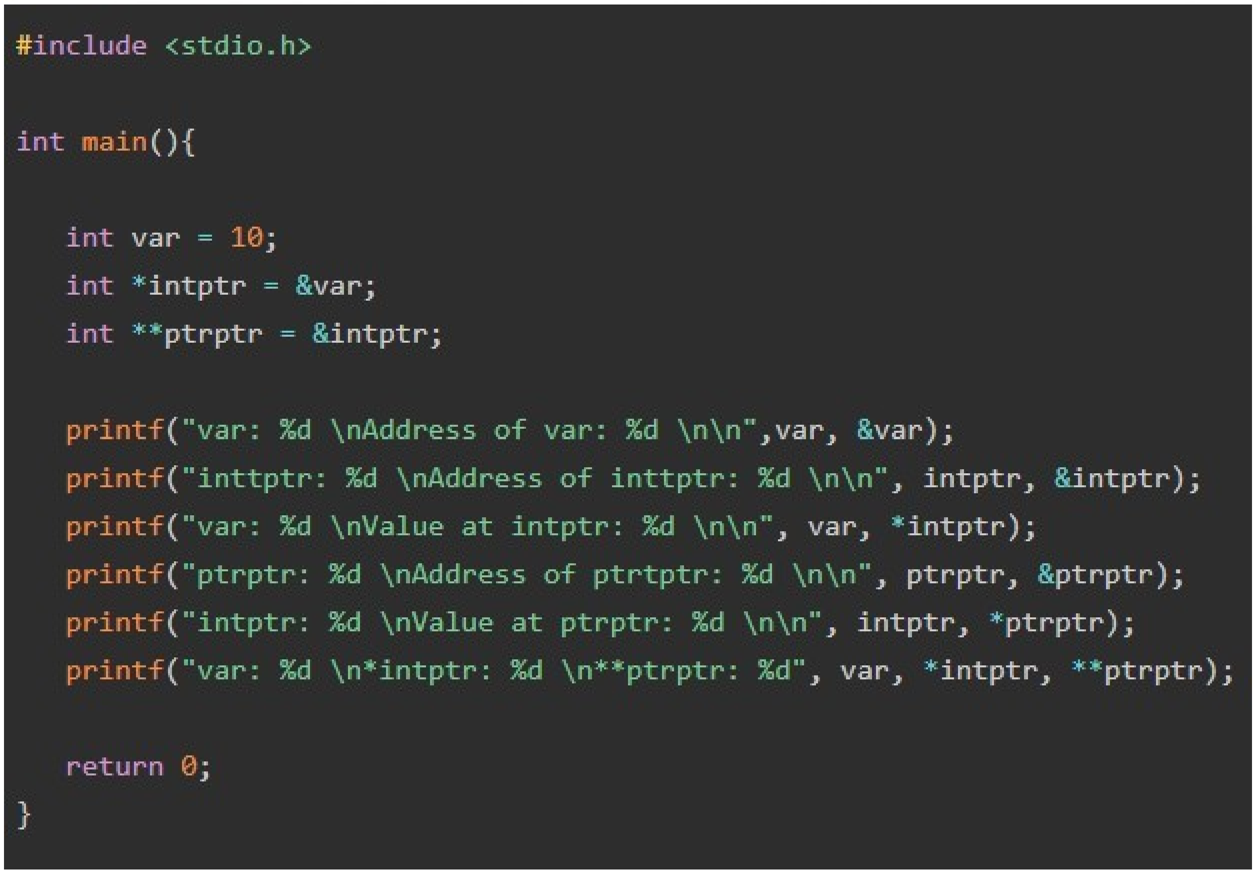
In the above figure, "a" is a normal "int" variable, whose pointer is "x". In turn, the variable stores the address of "x".

Note that "y" is declared as "int **" to indicate that it is a pointer to another pointer variable. Obviously, "y" will return the address of "x" and "*y" is the value in "x" (which is the address of "a").

To obtain the value of "a" from "y", we need to use the expression "**y". Usually, "y" will be called as the **pointer to a pointer**.

Example

Take a look at the following example −



```c
#include <stdio.h>

int main(){

   int var = 10;
   int *intptr = &var;
   int **ptrptr = &intptr;

   printf("var: %d \nAddress of var: %d \n\n",var, &var);
   printf("inttptr: %d \nAddress of inttptr: %d \n\n", intptr, &intptr);
   printf("var: %d \nValue at intptr: %d \n\n", var, *intptr);
   printf("ptrptr: %d \nAddress of ptrtptr: %d \n\n", ptrptr, &ptrptr);
   printf("intptr: %d \nValue at ptrptr: %d \n\n", intptr, *ptrptr);
   printf("var: %d \n*intptr: %d \n**ptrptr: %d", var, *intptr, **ptrptr);

   return 0;
}
```

Output

Run the code and check its output −

```
var: 10
Address of var: 951734452
```

```
inttptr: 951734452
Address of intptr: 951734456

var: 10
Value at intptr: 10

ptrptr: 951734456
Address of ptrtptr: 951734464
intptr: 951734452
Value at ptrptr: 951734452
var: 10
*intptr: 10
**ptrptr: 10
```

VOID POINTER

A void pointer is a pointer that has no associated data type with it. A void pointer can hold an address of any type and can be typecasted to any type.

Example of Void Pointer in C

```
// C Program to demonstrate that a void pointer can hold the address of any type-castable type
#include <stdio.h>
int main()
{
   int a = 10;
   char b = 'x';
   // void pointer holds address of int 'a'
   void* p = &a;
   // void pointer holds address of char 'b'
```

```
    p = &b;
}
```

Properties of Void Pointers

1. void pointers cannot be dereferenced.

Example

The following program doesn’t compile.

// C Program to demonstrate that a void pointer cannot be dereferenced

```
#include <stdio.h>
int main()
{
    int a = 10;
    void* ptr = &a;
    printf("%d", *ptr);
    return 0;
}
```

Output

The below program demonstrates the usage of a void pointer to store the address of an integer variable and the void pointer is typecasted to an integer pointer and then dereferenced to access the value. The following program compiles and runs fine.

// C program to dereference the void pointer to access the value

```
#include <stdio.h>
int main()
{
    int a = 10;
    void* ptr = &a;
    // The void pointer 'ptr' is cast to an integer pointer using '(int*)ptr' Then, the value is
dereferenced with `*(int*)ptr` to get the value at that memory location
```

```
    printf("%d", *(int*)ptr);

    return 0;

}
```

Output 10