Array of Pointers in C

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.

</> Open Compiler

```
#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.

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. */
   }

   for (i = 0; i < MAX; i++){
      printf("Value of var[%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
```

An Array of Pointers to Characters

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
```

An Array of Pointers to Structures

When you have a list of structures and want to manage it using a pointer. You can declare an array of structures to access and manipulate the list of structures.

Example

The below example demonstrates the use of an array of pointers to structures.

</> Open Compiler

```
#include <stdio.h>
#include <stdlib.h>
```

```c
#include <string.h>

// Declaring a structure
typedef struct {
  char title[50];
  float price;
} Book;

const int MAX = 3;
int main() {
  Book *book[MAX];

  // Initialize each book (pointer)
  for (int i = 0; i < MAX; i++) {
    book[i] = malloc(sizeof(Book));
    snprintf(book[i]->title, 50, "Book %d", i + 1);
    book[i]->price = 100 + i;
  }

  // Print details of each book
  for (int i = 0; i < MAX; i++) {
    printf("Title: %s, Price: %.2f\n", book[i]->title, book[i]->price);
  }

  // Free allocated memory
  for (int i = 0; i < MAX; i++) {
    free(book[i]);
  }

  return 0;
}
```

Output

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

```
Title: Book 1, Price: 100.00
Title: Book 2, Price: 101.00
Title: Book 3, Price: 102.00
```