Tutorial Link https://codequotient.com/tutorials/Pointer with arrays/5a006c21e63d6b7fd5dec2c0

TUTORIAL

Pointer with arrays

Chapter

As the array is stored contiguously in main memory pointer arithmetic helps in array manipulation. Suppose we declare an array age[5] in memory with base address 100. If each element takes 4 bytes then age[0] will be stored at locations 100 to 103. The array elements can be accessed by their index. If we use the name of the array only, it will return the base address of the array. So age is equivalent to &age[0]. We can use a pointer of type int to point to an array of int as shown below: -

```
int age[5] = {23, 43, 25, 34, 54};
int *ptr = age;      // int *ptr = &age[0] will do the same thing.
```

Now we can use the ptr and pointer arithmetic to traverse the array as shown in following program: -

```c
#include<stdio.h>
int main()
{
  int i;
```

```
  int age[5] = {1, 2, 3, 4, 5};
  int *p = age;
  for (i=0; i<5; i++)
  {
    printf("value at age[%d]=%d & value pointed by p=%d\n", i, age[i], *p);
    p++;
  }
  return 0;
}

```

The output of the above program will be: -

```
value at age[0]=1 & value pointed by p=1
value at age[1]=2 & value pointed by p=2
value at age[2]=3 & value pointed by p=3
value at age[3]=4 & value pointed by p=4
value at age[4]=5 & value pointed by p=5
```

In each iteration the pointer is incremented by 1 which automatically points to next element of the array. This combination is used mostly in programming to simplify the concepts and improve the readability of the program.

Also all the below statements will be equivalent due to array and pointers and print the elements of the array.

```
printf(“%d”, a[i]);
printf(“%d”, i[a]);
printf(“%d”, *(a+i));
printf(“%d”, *(i+a));
printf(“%d”, a[i]);
```

Because ‘a’ refers to the base address of the array and it converts to pointer arithmetic and all statements will add the value of i to base address and do pointer arithmetic and print the value.

also

```
printf(“%d”, *a); // it will print the value of a[0] only
a++;    // compile time error, as we can not change the base
address of array.
```

These are used in following program: -

```c
#include<stdio.h>
int main()
{
  int i=0;
  int age[5] = {1, 2, 3, 4, 5};
  int *p = age;
  printf("value at age[%d]=%d \n", i, age[i]);
  printf("value at age[%d]=%d \n", i, i[age]);
  printf("value at age[%d]=%d \n", i, *(age+i));
  printf("value at age[%d]=%d \n", i, *(i+age));

  printf("Value at age[0] is %d \n", *age);
  printf("Base Address of  array age is by age=%p and
by p=%p\n", age,p);
  return 0;
}

```

The output of this program is: -

```
value at age[0]=1
value at age[0]=1
value at age[0]=1
value at age[0]=1
Value at age[0] is 1
Base Address of array age is by age=0xbf857264 and by
p=0xbf857264
```

Pointers and array also create some declarations hard to grasp, like the two below have totally different meanings: -

```
int (*pointer1)[5];
int *pointer2[5];
```

Here, pointer1 is a pointer to an array of 5 integers, So we can declare an array of size 5 and assign its base address to pointer1 as below: -

```
int age[5] = {0, 1, 2, 3, 4};
pointer1 = age;       // or pointer1 = &age[0];
```

Whereas pointer2 is an array of 5 elements and each element is a pointer to an integer. In this case pointer2 is an array itself of pointers pointing to integer variables i.e.

Index	pointer2[0]	pointer2[1]	pointer2[2]	pointer2[3]	pointer2[4]
value	Address of an integer variable	Address of an integer variable	Address of an integer variable	Address of an integer variable	Address of an integer variable

Each of the elements of pointer2 can point to a integer variable.

Video Explanation

<iframe width="560" height="315" src="https://www.youtube.com/embed/DJB6PdGmlbk" title="YouTube video player" frameborder="0" allow="accelerometer; autoplay; clipboard-write; encrypted-media; gyroscope; picture-in-picture" allowfullscreen></iframe>