Pointer to an Array in C

It is most likely that you would not understand this section until you are through with the chapter 'Pointers'.

Assuming you have some understanding of pointers in C, let us start: An array name is a constant pointer to the first element of the array. Therefore, in the declaration −

```
double balance[50];
```

balance is a pointer to &balance[0], which is the address of the first element of the array balance. Thus, the following program fragment assigns **p** as the address of the first element of **balance** −

```
double *p;
double balance[10];

p = balance;
```

It is legal to use array names as constant pointers, and vice versa. Therefore, *(balance + 4) is a legitimate way of accessing the data at balance[4].

Once you store the address of the first element in 'p', you can access the array elements using *p, *(p+1), *(p+2) and so on. Given below is the example to show all the concepts discussed above −

Live Demo

```
#include <stdio.h>

int main () {

   /* an array with 5 elements */
   double balance[5] = {1000.0, 2.0, 3.4, 17.0, 50.0};
   double *p;
   int i;

   p = balance;

   /* output each array element's value */
   printf( "Array values using pointer\n");
```

```
   for ( i = 0; i < 5; i++ ) {
      printf("*(p + %d) : %f\n",  i, *(p + i) );
   }

   printf( "Array values using balance as address\n");

   for ( i = 0; i < 5; i++ ) {
      printf("*(balance + %d) : %f\n",  i, *(balance + i) );
   }

   return 0;
}
```

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

```
Array values using pointer
*(p + 0) : 1000.000000
*(p + 1) : 2.000000
*(p + 2) : 3.400000
*(p + 3) : 17.000000
*(p + 4) : 50.000000
Array values using balance as address
*(balance + 0) : 1000.000000
*(balance + 1) : 2.000000
*(balance + 2) : 3.400000
*(balance + 3) : 17.000000
*(balance + 4) : 50.000000
```

In the above example, p is a pointer to double, which means it can store the address of a variable of double type. Once we have the address in p, ***p** will give us the value available at the address stored in p, as we have shown in the above example.