Course
ELEC-A7100
Course materials
Your points
Microsoft Teams
Code Vault

This course has already ended.

« 1 Pointers Course materials 3 Arrays »
ELEC-A7100 / 3. Pointers / 2 Address arithmetics

Third round tasks as a `ZIP file.`

This section focuses on one of the most important concepts of the C language: **Pointers.** Pointers allow processing of **arrays**, **strings** and various other C-programming variables. After reading through this section, you will understand pointers and operations allowed on pointers, how to deal with array data using pointers, as well as know how to write simple programs in which pointers are used. In future sections, pointers are used in practice constantly, so in this course you will be practicing how to use them in abundance.

Address arithmetics¶

Plus and minus arithmetic operators can be used also on pointers. These cause the pointer **to be moved either forward or backward by the given number of objects of the size of referenced data unit**. Therefore, the adjustment in the address depends on the size of the data type behind the pointer.

Here is an example that demonstrates address arithmetics in different formats:

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      int array[50];  // reserve space for 50 integers
6      int *intPtr = array;  // assign the start of the array to a pointer
7      int i = 50;
8      while (i > 0) {
9          *intPtr = i * 2;  // write i*2 to the location indicated by the pointer
10         intPtr++;  // move the pointer to the next element
11         i--;
12     }
13
14     intPtr = array;  // move the pointer back to the beginning of the array
15
16     for (i = 0; i < 50; i++) {
17         int value = *(intPtr + i);  // retrieve the i:th element in the array
18         printf("%d ", value);
19     }
20 }
```

The function starts by allocating space for 50 integers (arrays are described later on this page), and setting a pointer **'intPtr'** to point at the beginning of this space (reference to array is a pointer to its first element). In the while loop, the function sets the location referred by 'intPtr' based on integer variable **i**, multiplied by two (line 9). intPtr++ on line 10 causes the pointer to move to the next integer in the array. We can repeat this 50 times, because we allocated space for 50 integers.

After the array is set, **'intPtr'** is reset back to the beginning of the array (line 14), and a different form of loop is applied. Here we set the integer value based on the i-th object in the array (line 17): adding 'i' to a pointer walks **'i'** steps forward in memory. It is possible to use dereference operator together with arithmetic calculations on pointers, as shown here, but the use of parenthesis is important in this case because of the precedence rules. As a result, this function will output even numbers from 100 to 2 in decreasing order, separated by space.

In order for the address arithmetics to work correctly, it is important that the pointer data type is correct. Compiler will remind you about this, if incompatible pointers are used together.

Pointers can be used in comparisons like other variables, but it should be noticed that in such case addresses are compared, not the values behind the pointers. For example the following code will tell that the pointers are different, but the values behind pointers are same.

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      int a = 5;
6      int b = 5;
7      int *pa = &a;
8      int *pb = &b;
9
10     if (pa == pb)
11         printf("Pointers are same\n");
12     else
13         printf("Pointers are different\n");
14
15     if (*pa == *pb)
16         printf("Values are same\n");
17 }
```

« 1 Pointers Course materials 3 Arrays »