**Pointer arithmetics.**

To conduct arithmetical operations on pointers is a little different than to conduct them on regular integer types. To begin with, only addition and subtraction operations are allowed; the others make no sense in the world of pointers. But both addition and subtraction have a slightly different behaviour with pointers, according to the size of the data type to which they point.

When fundamental data types were introduced, we saw that types have different sizes. For example: char always has a size of 1 byte, short is generally larger than that, and int and long are even larger; the exact size of these being dependent on the system (namely on compiler and architecture of the system). For example, let's imagine that in a given system, char takes 1 byte, short takes 2 bytes, and long takes 4.

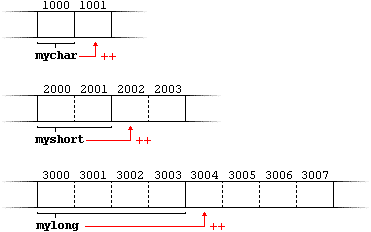
Suppose now that we define three pointers in this compiler:

|  |  |  |
| --- | --- | --- |
| 1 2 3 | char \*mychar;  short \*myshort;  long \*mylong; |  |

and that we know that they point to the memory locations 1000, 2000, and 3000, respectively. Therefore, if we write:

|  |  |  |
| --- | --- | --- |
| 1 2 3 | ++mychar;  ++myshort;  ++mylong; |  |

*mychar*, as one would expect, would contain the value 1001. But not so obviously, *myshort* would contain the value 2002, and *mylong* would contain 3004, even though they have each been incremented only once. The reason is that, when adding one to a pointer, the pointer is made to point to the following element of the same type, and, therefore, the size in bytes of the type it points to is added to the pointer.



This is applicable both when adding and subtracting any number to a pointer. It would happen exactly the same if we wrote:

|  |  |  |
| --- | --- | --- |
| 1 2 3 | mychar = mychar + 1;  myshort = myshort + 1;  mylong = mylong + 1; |  |

Regarding the increment (++) and decrement (--) operators, they both can be used as either prefix or suffix of an expression, with a slight difference in behaviour: as a prefix, the increment happens before the expression is evaluated, and as a suffix, the increment happens after the expression is evaluated. This also applies to expressions incrementing and decrementing pointers, which can become part of more complicated expressions that also include dereference operators (\*). Remembering operator precedence rules, we can recall that postfix operators, such as increment and decrement, have higher precedence than prefix operators, such as the dereference operator (\*). Therefore, the following expression:

|  |  |  |
| --- | --- | --- |
|  | \*p++ |  |

is equivalent to \*(p++). And what it does is to increase the value of p (so it now points to the next element), but because ++ is used as postfix, the whole expression is evaluated as the value pointed originally by the pointer (the address it pointed to before being incremented).

Essentially, these are the four possible combinations of the dereference operator with both the prefix and suffix versions of the increment operator (the same being applicable also to the decrement operator):

|  |  |  |
| --- | --- | --- |
| 1 2 3 4 | \*p++ // same as \*(p++): increment pointer, and dereference unincremented address  \*++p // same as \*(++p): increment pointer, and dereference incremented address  ++\*p // same as ++(\*p): dereference pointer, and increment the value it points to  (\*p)++ // dereference pointer, and post-increment the value it points to |  |

A typical -but not so simple- statement involving these operators is:

|  |  |  |
| --- | --- | --- |
|  | \*p++ = \*q++; |  |

Because ++ has a higher precedence than \*, both p and q are incremented, but because both increment operators (++) are used as postfix and not prefix, the value assigned to \*p is \*q before both p and q are incremented. And then both are incremented. It would be roughly equivalent to:

|  |  |  |
| --- | --- | --- |
| 1 2 3 | \*p = \*q;  ++p;  ++q; |  |

Like always, parentheses reduce confusion by adding legibility to expressions.