Address Arithmetic

When a pointer points to a contiguous list of data elements, such as the data stored on the heap by calling `stbi_load()`, we can apply the operators `+` and `-` to the pointer. This is called **pointer or address arithmetic**. Pointer arithmetic is similar to mixed type arithmetic with integers and floating-point numbers. If you add an integer and a floating-point number, the result is a floating-point number. Similarly:

- **Adding an integer** to a pointer gives us a **new address value.**
- **Subtracting one pointer from another** produces an integer.

Pointer addition considers the **size of the base type**; it doesn't just change the address by x number of bytes. Consider this code:

```
vector<int> v{1, 2, 3, 4, 5};
auto *p = &v[1];
cout << "p->" << p << ", " << *p << endl;
cout << "(p+1)->" << (p+1) << ", " << *(p+1) << endl;
```

When run, (click the previous link) you'll see something like this:

```
p->0x559a1c997eb4, 2
(p+1)->0x559a1c997eb8, 3
```

The pointer `p` contains the address `0x559a1c997eb4` (although it may be a different address when you run it), and it points to the second element in the `vector<int> v`. The address `(p + 1)` is `0x559a1c997eb8`. Note that for each unit that is added to a pointer value, the internal numeric value must be increased by the size of the **base type of the pointer**. In this case, that is `4` bytes, since the `sizeof(int)` is `4` on this platform.

Pointer Difference

Subtracting one pointer from another returns a **signed number** (of type `ptr_diff`) which represents the **number of elements** (**not** the bytes) between the two pointers. This is called **pointer difference**, and we'll use it more when we start looking at arrays.