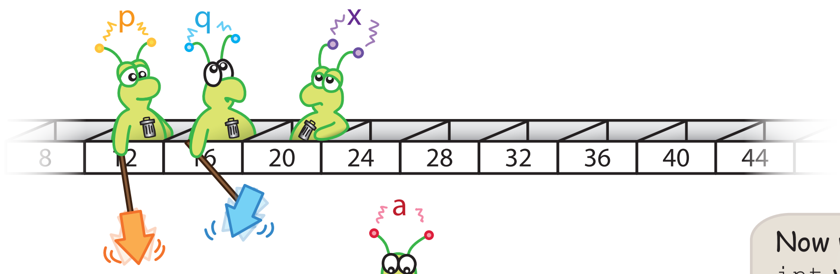
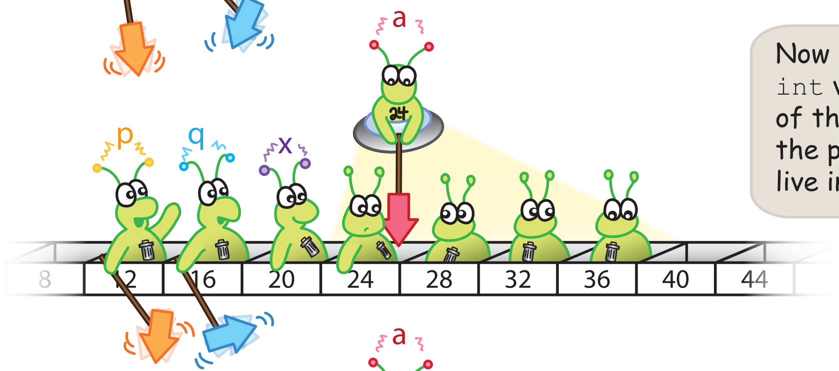
Pointers and Arrays

Imagine memory as long block of boxes that store data. Each box is labeled with an **address**. A **pointer** is simply a variable that holds a particular address. An **array** is a group of contiguous boxes that can be accessed by their index values. Array and pointer variables are mostly the same; we're going to highlight one of the ways they are different.



```
int *p, *q, x;
```

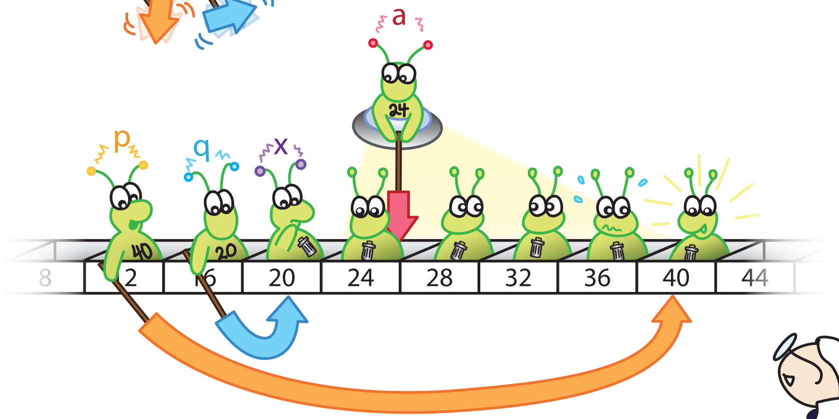
Here, we declare p and q as pointers that will hold the addresses of int variables, and x as an ordinary int variable.



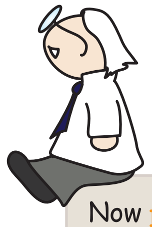
Now we define an array that can store 4 int values. Now, a points to the first index of this array. However, notice that unlike the pointer variables p and q, a does NOT live in memory.

```
int a[4];
```

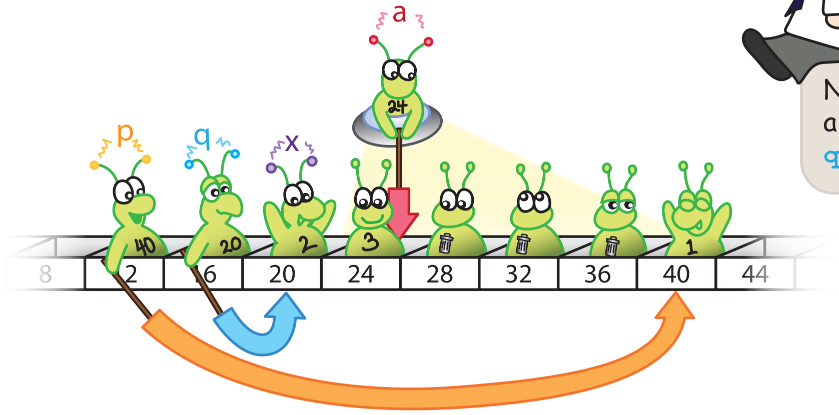
In the illustrations above, none of these variables have been assigned values yet, so they contain "garbage" -- whatever had been stored into these blocks of memory beforehand. We change that with the code below.



```
p = (int*) malloc(sizeof(int));
q = &x;
```



Now p contains, or **points** to, the address of a dynamically allocated memory space that can store one int value, and q points to the address of the variable **x**.



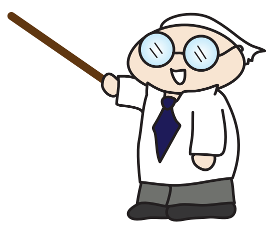
```
*p = 1;
*q = 2;
*a = 3;
```

When we **dereference** these pointers, we simply look inside the addresses that they point to. In this way, we can access the data stored there and even change those values. Here, we store the values of 1, 2, and 3 into the boxes that the addresses p, q, and a point to, respectively.

```
printf("*p:%u, p:%u, &p:%u\n", *p, p, &p);
printf("*q:%u, q:%u, &q:%u\n", *q, q, &q);
printf("*a:%u, a:%u, &a:%u\n", *a, a, &a);
```

Printed output

```
*p:1, p:40, &p:12
*q:2, q:20, &q:16
*a:3, a:24, &a:24
```



One last thing before you go.... We said that the variable of an array doesn't live in memory. But, the system prints the address of the first index of the array as the address of **a**.