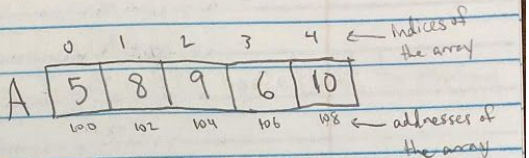
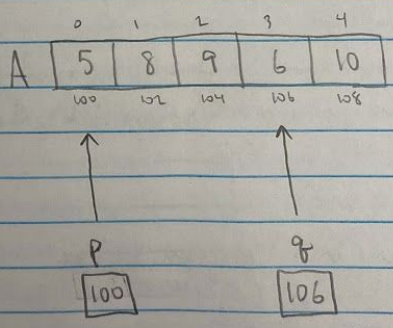
Pointer Math

```
int main()
{
```

```
int A[5] = {5,8,9,6,10};
```




```
/*
The array is of type 'int'

The size of an int is 2 bytes

The value '5' in the array is taking up space at the memory location
'100'. It is taking space at '100' and less than '101' (100 & 101), hence
why it takes up 2 bytes of space.

*/

int *p, *q; // p and q are two pointers

p = &A[0];

q = &A[3];
```




```
/*
As the syntax might suggest, p finds the address of the value stored
at point 0 of the array. Also, q finds the address of the value stored
at point 3 of the array.
*/
```

/*

The initialization of p can also be written as:

p = A;

OR

p = &A;

These all have the same meaning.

*/

Pointer Increment:

p++;

/*

'p++' tells the program to move on to the next elements stored in the given array.

It will increment by 2, since that is how the memory allocations are separated in the given array.

*/

Here in our program, we have:

int *p;
p++; //increments by 2 bytes

float *p;
p++; //increments by 4 bytes

(this incrementation varies depending on the data type being used)

Pointer Decrementi:

q--; //moves backe

Still increments by 2 bytes, so q will move from memory space [106] to memory space [104]

Constant Addition:

p = p+2; //moves 'p' element 2 spaces in the forward direction of the array

So,

If p is currently at 5, (100)
p+2 moves it to where 9 is (104)

Constant Subtraction:

q = q-2; //moves 'q' element 2 space in the backward direction

So,

If q is currently at 6 (106)
q-2 moves it to where 8 currently is (102)

Subtraction of 2 pointers:

I = q-p; //p and q are pointing at different directions

/*

q-p = 106-100 = 6

6/2 bytes = [3]

*/

We CANNOT:

- add two pointers (p+q)
- multiply two pointers (p*q)
- multiply constants to a pointer (2*p)

} none of these have any meaning

Let's observe the following line of code:

```
printf("%d", *++p)

/*
   The first operation of the given expression is ① '++' and the
   second operation is ② *. It goes from right to left.
*/
```

So if p starts at value '5' or memory location '100', it will move to value '8' or memory location '102'.

Let's observe another line of code:

```
printf("%d", ++*p)

/* First operation is ① *
   Second operation is ② ++
*/
```

So first we take the data of p, where the value '5' is taken. Then, we use the ++ operation that ultimately turns the given value into '6'

Let's look at another operation:

printf("%d", *p++)

This operation means that we should continue reading from right to left. So the first operation we deal with is '++' on the pointer. However, this expression is a post-increment expression, so it will actually not be done right away. It will instead be done <u>after</u> the expression. So the next operation that will be done is '*', which is done on 'p' first. This means it will take the value '5'. Now, it will increment. It will move 'p' to the next integer. So it will move from value 5, memory location 100, to value 8, memory location 102

++*p → take data and increment data / ++(*p) → same meaning

*++p → move pointer to next element and read data

*p++ → read data, then move p to next / (*p)++ → different meaning

↑
increments the data

Pointer Math