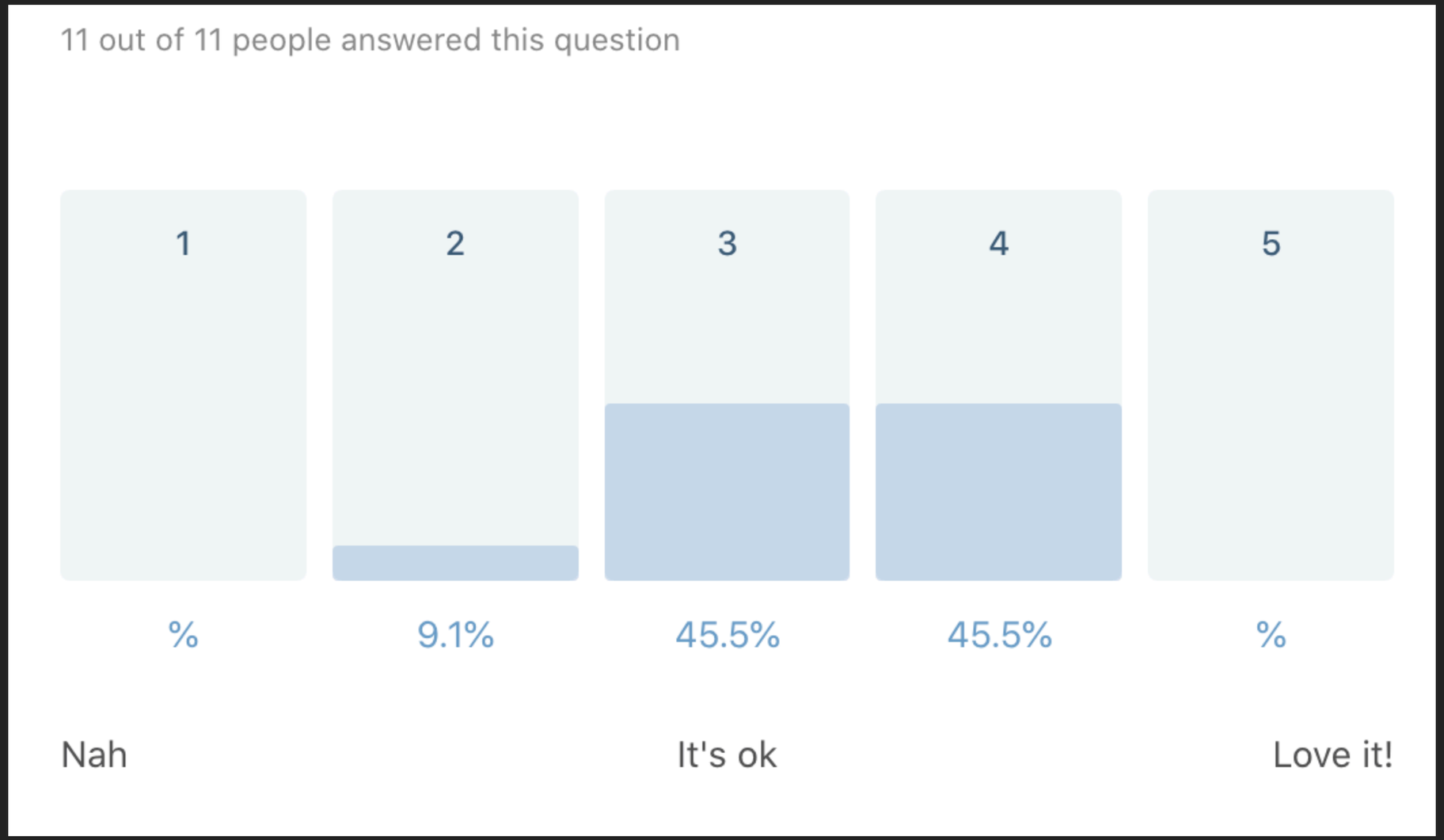
SEPTEMBER 30TH 2020

ELEMENTARY PROGRAMMING

SOME COVID BEST PRACTICES BEFORE WE START

- If you fill ill, go home
- Keep your distance to others
- Wash or sanitise your hands
- Disinfect table and chair
- Respect guidelines and restrictions

FEEDBACK CHECK



NEW FEEDBACK

- I would really like for you to take a survey at the end of the session
- Feedback is important, please take the time to do it
- Pretty please <3
- Type this in your browser http://bit.ly/elemprog7

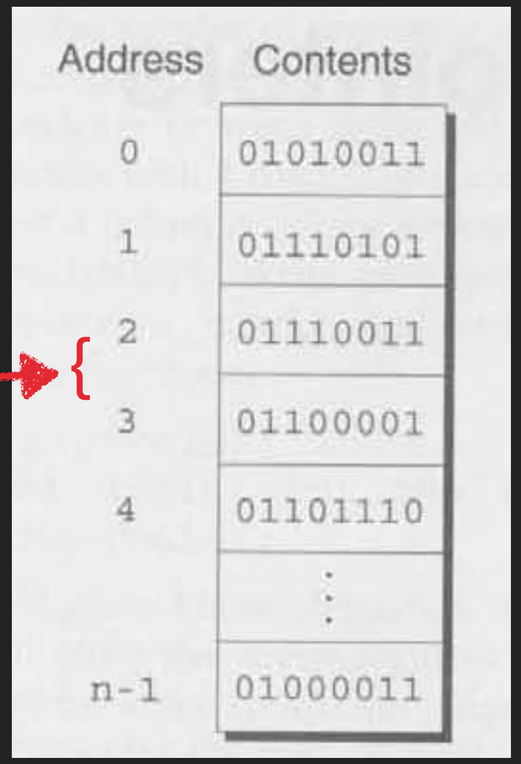
POINTERS

- In most modern computers, the main memory is divided into bytes (each byte store 8 bits of information)
- Each byte as a unique address in memory
- A variable occupies one or more bytes in memory

POINTERS

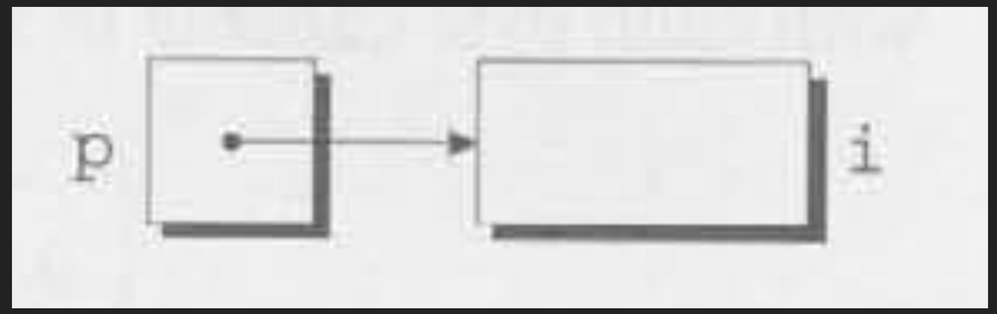
- This is an example of memory:

The address of the first byte of a variable is said to be the address of the variable. For example if a variable occupies address 3 and 4, then 3 is the address of the variable



POINTERS

- We can store the address of the variable in special places called pointer variables
- We declare pointer variables this way: `type *p;`



POINTERS

- So for example, all the following are valid pointers:
  - `int *p;`
  - `double *q;`
  - `char *r;`

POINTERS

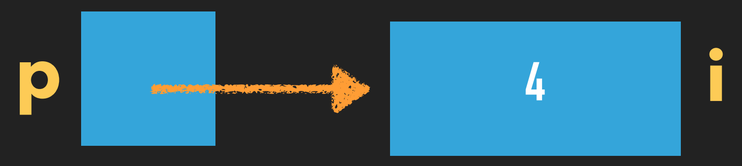
- With & we access the address of a variable:

```
int i = 4;
int *p;
p = &i;
```

POINTERS

- With * we access the value pointed by a pointer:

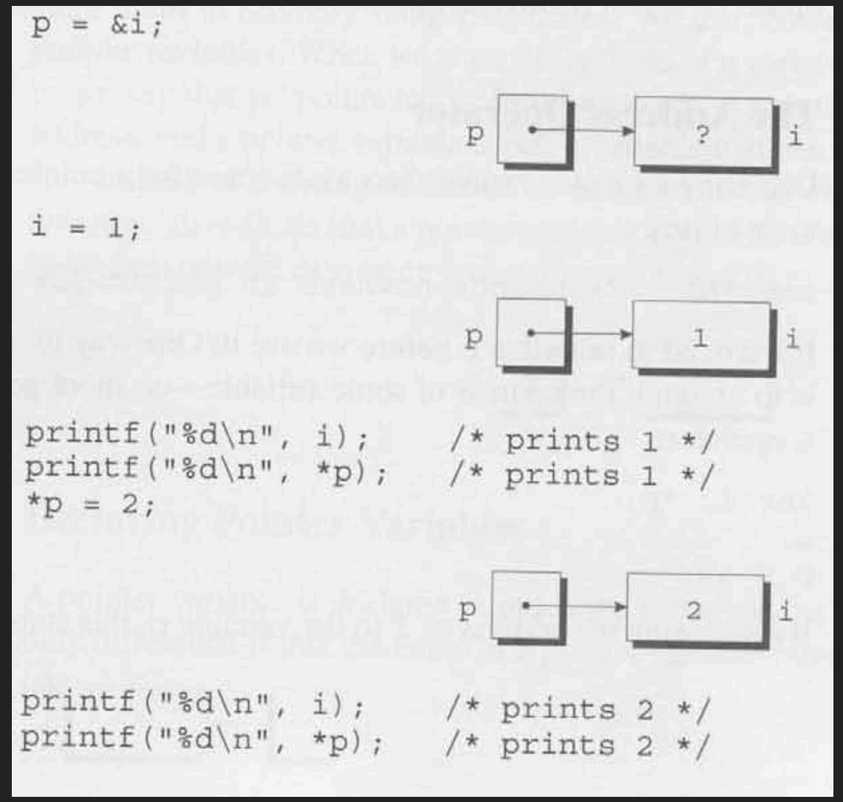
```
int i = 4;
int *p;
p = &i;
printf("%d",*p);
```



4 will be printed

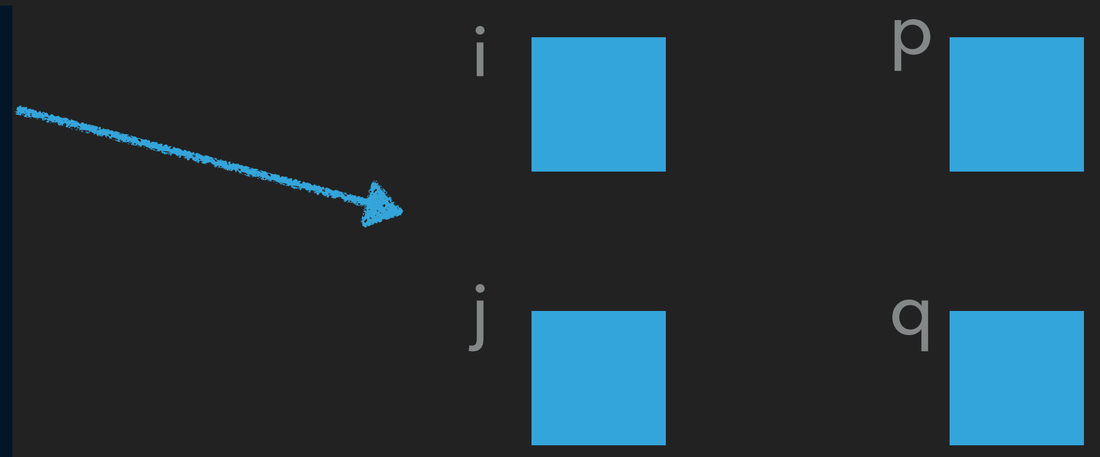
POINTERS

- A pointer point to the same address so changing the value of a pointer change the value of the variable



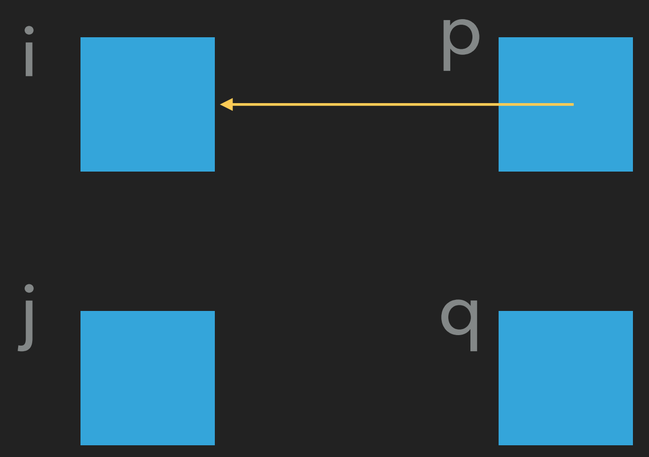
POINTERS CAN PLAY WITH YOUR BRAIN #1

```
int i, j, *p, *q;
p = &i;
q = p;
*p = 1;
*q = 2;
```



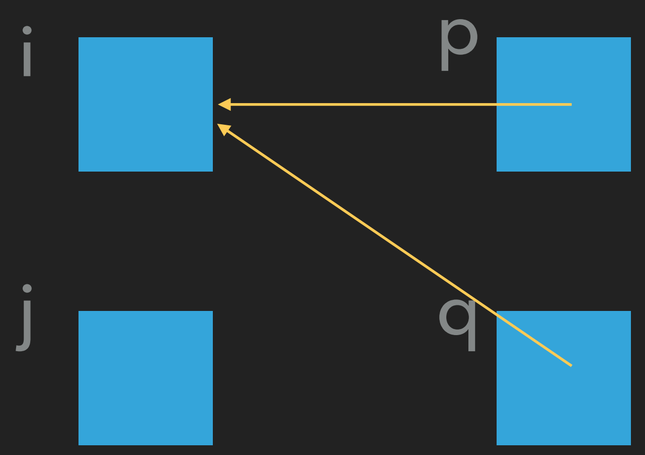
POINTERS CAN PLAY WITH YOUR BRAIN #1

```
int i, j, *p, *q;
p = &i;
q = p;
*p = 1;
*q = 2;
```



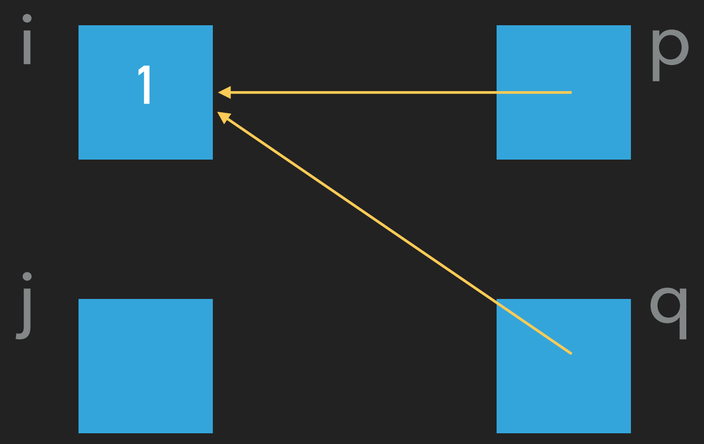
POINTERS CAN PLAY WITH YOUR BRAIN #1

```
int i, j, *p, *q;
p = &i;
q = p;
*p = 1;
*q = 2;
```



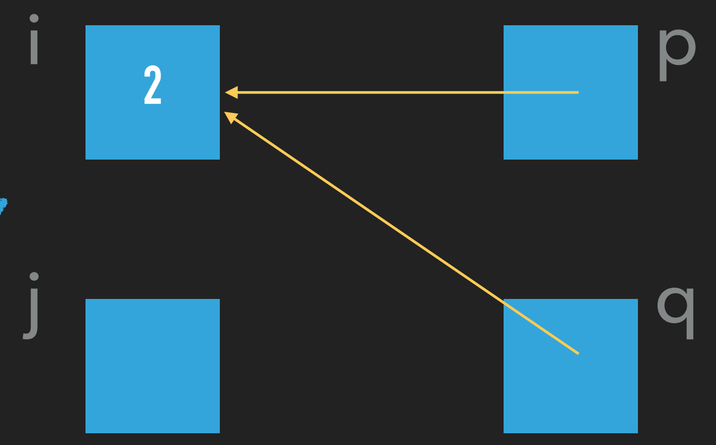
POINTERS CAN PLAY WITH YOUR BRAIN #1

```
int i, j, *p, *q;
p = &i;
q = p;
*p = 1;
*q = 2;
```



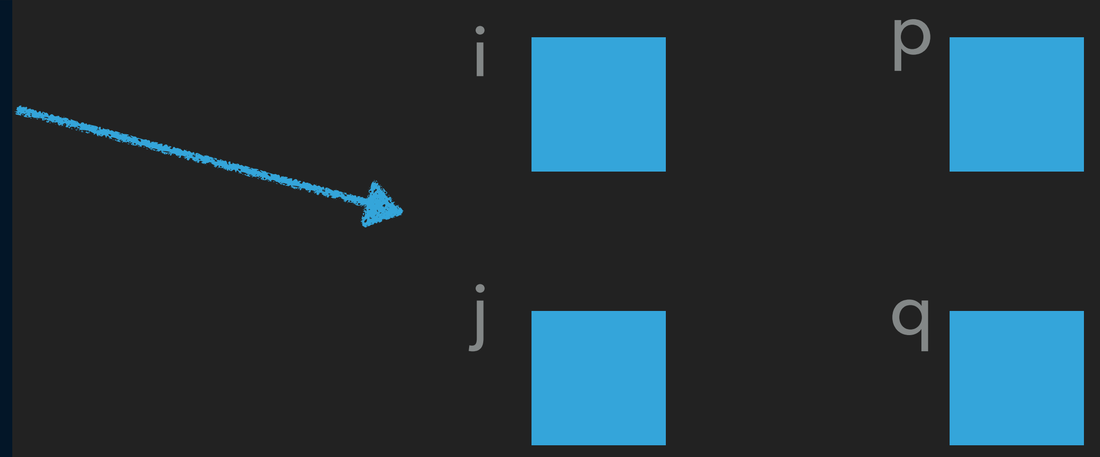
POINTERS CAN PLAY WITH YOUR BRAIN #1

```
int i, j, *p, *q;
p = &i;
q = p;
*p = 1;
*q = 2;
```



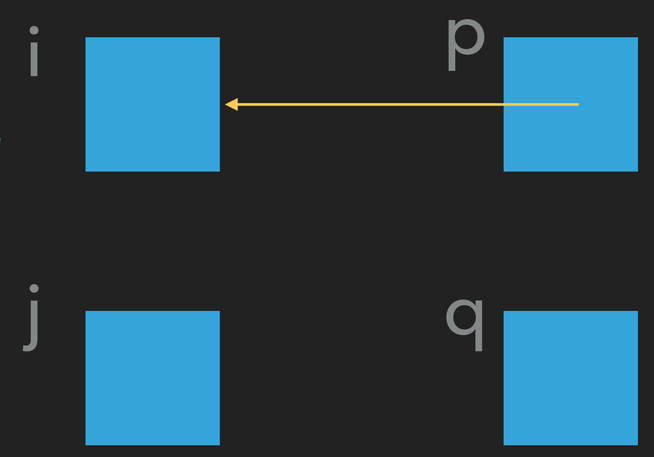
POINTERS CAN PLAY WITH YOUR BRAIN #2

```
int i, j, *p, *q;
p = &i;
q = &j;
i = 1;
*q = *p;
```



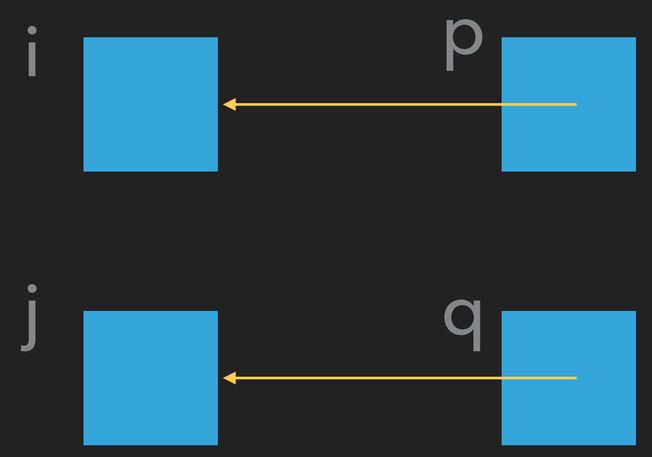
POINTERS CAN PLAY WITH YOUR BRAIN #2

```
int i, j, *p, *q;
p = &i;
q = &j;
i = 1;
*q = *p;
```



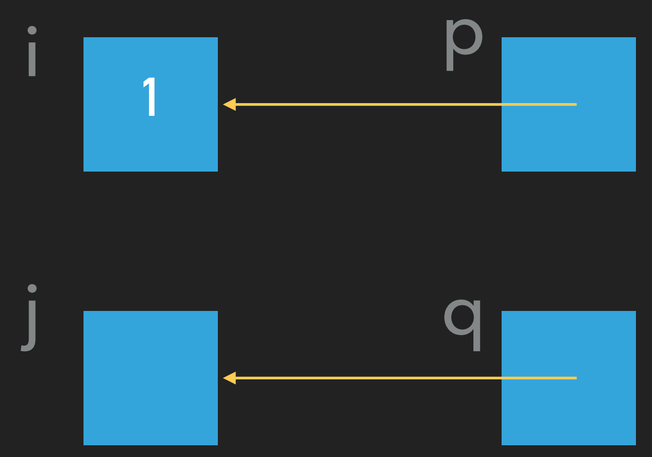
POINTERS CAN PLAY WITH YOUR BRAIN #2

```
int i, j, *p, *q;
p = &i;
q = &j;
i = 1;
*q = *p;
```



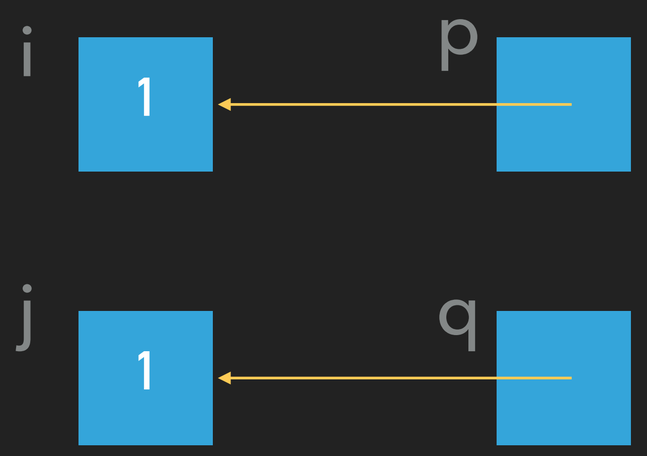
POINTERS CAN PLAY WITH YOUR BRAIN #2

```
int i, j, *p, *q;
p = &i;
q = &j;
i = 1;
*q = *p;
```



POINTERS CAN PLAY WITH YOUR BRAIN #2

```
int i, j, *p, *q;
p = &i;
q = &j;
i = 1;
*q = *p;
```



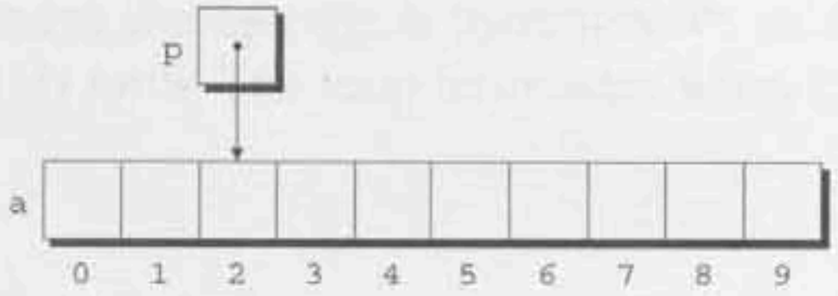
YOU CAN USE POINTERS IN THE ARGUMENTS OF A FUNCTION

```
void decompose(double x, long *int_part, double *frac_part) {
  *int_part = (long) x;
  *frac_part = x - *int_part;
}
```

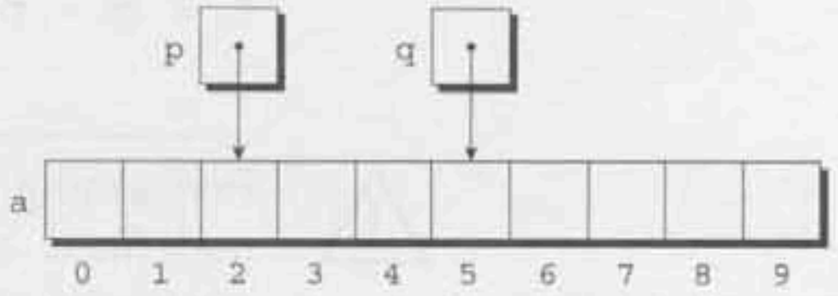
```
  double x = 5.5;
  long int_part;
  double frac_part;
  decompose(x, &int_part, &frac_part);
  printf("int %ld\nsdouble %g\n", int_part, frac_part);
```

POINTERS AND ARRAYS – ADD

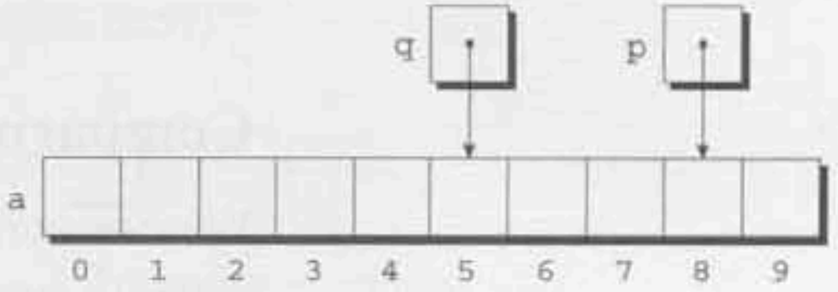
```
p = &a[2];
```




```
q = p + 3;
```

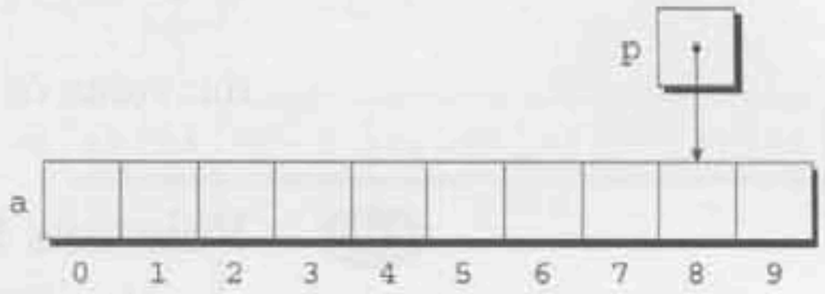



```
p += 6;
```

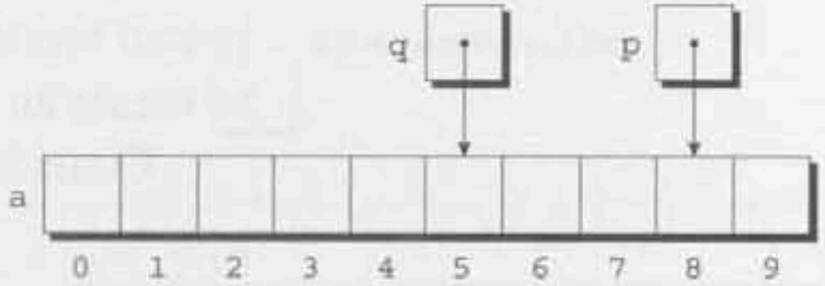


POINTERS AND ARRAYS – SUBTRACT INTEGERS

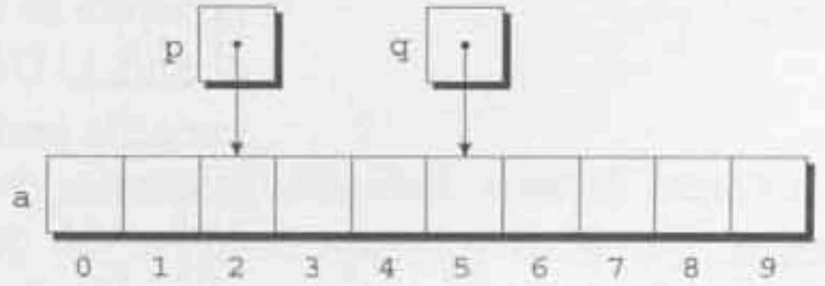
```
p = &a[8];
```




```
q = p - 3;
```




```
p -= 6;
```



POINTERS AND ARRAYS – ARRAY AS POINTER

- Given we have an array defined as `int a[10]`, then `a` is a pointer to the first element and we can do something like this:

```
#include <stdio.h>
#define N 10
int main(void) {
  int a[N], *p;
  printf("Enter %d numbers: ", N);
  for(p = a; p < a + N; p++) {
    scanf("%d", p);
  }
  printf("In reverse order");
  for(p = a + N - 1; p ≥ a; p--) {
    printf(" %d", *p);
  }
  printf("\n");
}
```

POINTERS AND ARRAYS – ARRAY AS POINTER

- And that is the reason why when we declare an array as argument of a function we also need to pass the side. That's actually a point to the first element of the array.

```
int find_largest(int a[], int n) {
  int max;
  max = a[0];
  for(int i = 1; i < n; i++) {
    if(a[i] > max) {
      max = a[i];
    }
  }
  return max;
}
```

STRINGS AS POINTERS

- `"this"` is a string literal and it's an array of chars
- We can use string literals whenever there is a `char *c` as well
- Whenever we do `char *c = "abc";` c points to the first element of "abc" so it's not an assignment of the array
- This is how it's in memory:

T	H	I	S	\0

STRINGS AS POINTERS

- `"this"` is a string literal and it's an array of chars
- We can use string literals whenever there is a `char *c` as well
- Whenever we do `char *c = "abc";` c points to the first element of "abc" so it's not an assignment of the array
- This is how it's in memory:

T	H	I	S	\0

HOW CAN WE INITIALIZE A STRING

- `char date[8]="June 14"`
- `char date[8] = {'J’, ‘u', 'n', ‘e', ' ', ‘1’,’4','\0'}`
- `char date[9] = {'J’, ‘u', 'n', ‘e', ' ', ‘1’,’4’,’\0', ‘\0'}`
- `char date[]="June 14"`
- `char *date="June 14"`

IS IT DIFFERENT?

- Yes:
  - when you use the pointer, the pointer can point to different string literal during the lifetime of the program
  - when you use an array variable instead you can’t change where you are pointing (but you can change the content of the array)

HOW CAN WE READ A STRING?

- Your can use `scanf` as usual but you need to remember the `scanf` stops when there are spaces
- You can use `gets` but it's not safe because it doesn't have any check on maximum allowed chars
- You can write one by yourself:

```
int read_line(char str[], int n) {
    int ch, i = 0;
    while ((ch = getchar()) ≠ '\n') {
      if(i < n) {
        str[i] = ch;
        i = i + 1;
      }
    }
    str[i] = '\0'; // we close the string with null char
    return i;
}
```

PLEASE READ 13.5 AND 13.5

- Question about this two chapters could be present during the exam:
  - How does `strncpy` work?
  - How does `strcat` work?
  - Are there strings idioms?
- We will see now some examples in the code

NEW FEEDBACK

- I would really like for you to take a survey at the end of the session
- Feedback is important, please take the time to do it
- Pretty please <3
- Type this in your browser http://bit.ly/elemprog7

SOME COVID BEST PRACTICES BEFORE WE LEAVE

- Disinfect table and chair
- Maintain your distance to others
- Wash or sanitise your hands
- Respect guidelines and restrictions outside