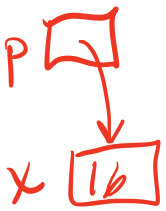
Last time:

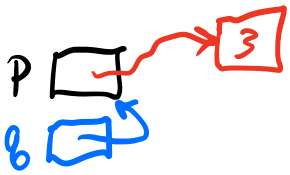
```
Void increment(int *p)
{
   *p = *p+1;
}

int x=16;
increment(&x);
```



What if you want a pointer to be changed?

- you need to pass the address of the pointer, of course;

```
int *p= NULL;
initialize(&p, 3); // this should cause p to
                   // point to a memory location
                   // that holds 3.
```

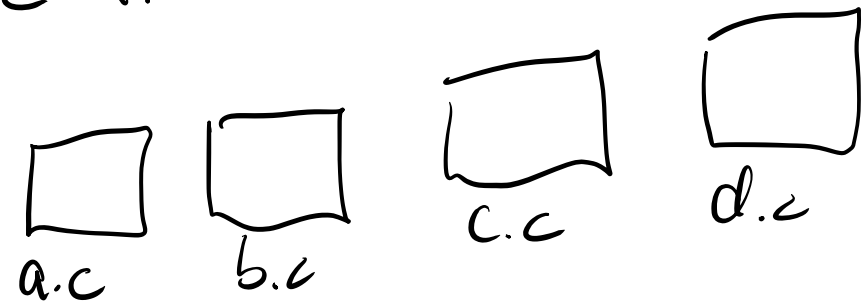


How to write initialize?

```
void initialize(int **q, int y)
{
  *q = malloc(sizeof(int));
  **q = 3;
}
```

Multi-File C programs

Simplest form: Just a bunch of .c files.



One of the files (and only one) must have the main() function.

Compiling them all at once:

```
gcc a.c b.c c.c d.c
```

↖ generate an executable named a.out (a.exe on Windows)

```
gcc -o myprog a.c b.c c.c d.c
```

the executable is named myprog (myprog.exe on Windows)

Compiling one file at a time:

```
gcc -c b.c
```

↖ just compile the file, don't try to link to form an executable.

- generates a file named b.o

↑ stands for "object" file which contains machine code (binary)

```
gcc -o yourprog a.o b.o c.c d.o
```

↑ will set compiled

- linked together to form the executable yourprog.

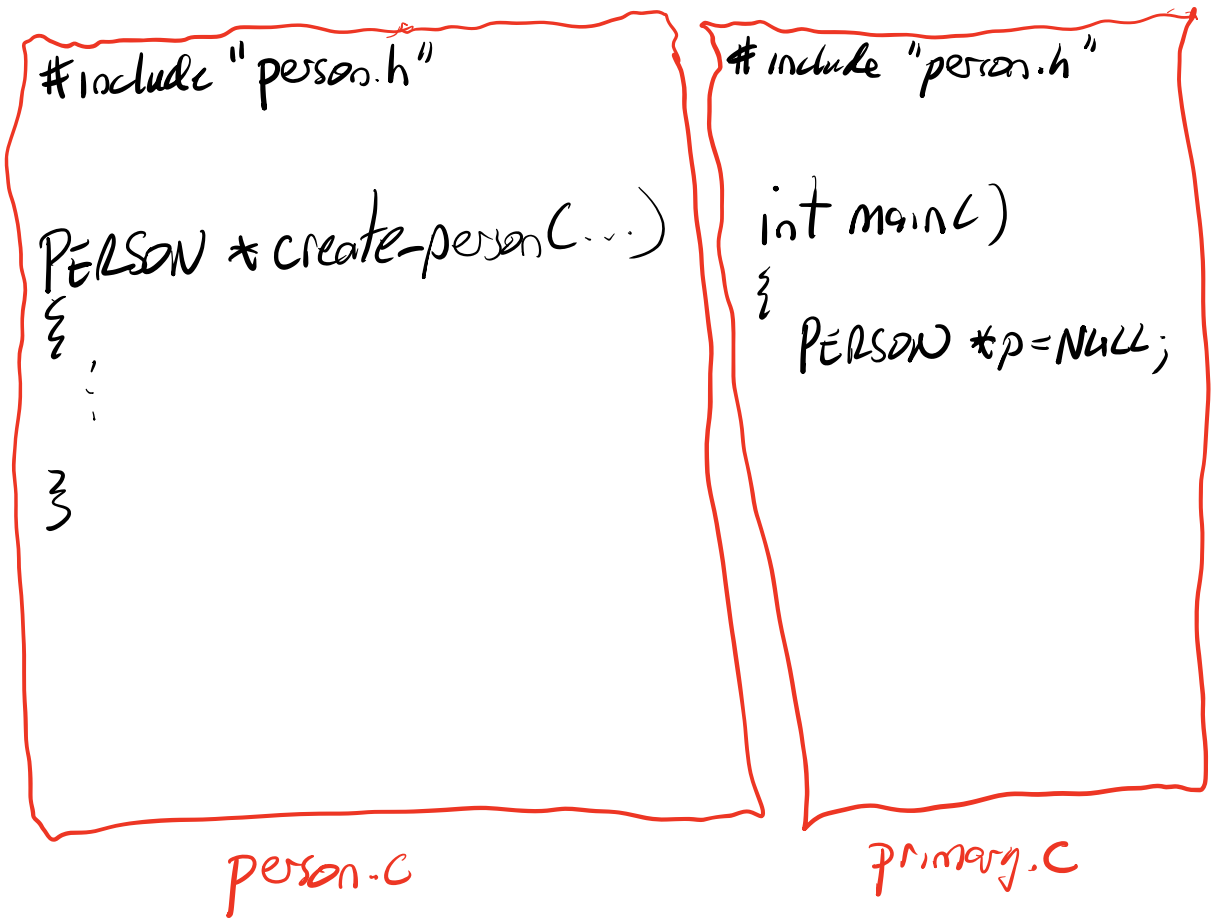
What if you need to share type declarations, global variables, and functions between the files?

- Use header files
- .h extension

For type declarations:

```
typedef struct {
    char *name;
    int age;
} PERSON;
```

person.h



To call a function written in another file:

- the function must be declared using a "prototype" (or "signature") in a .h file.

```
int compute(int x);
```

foo.h

```
int compute(int x)
{
  :
}
```

foo.c

```
#include "foo.h"

void f()
{
  int y = compute(27);
  :
}
```

bar.c

How about global variables?

```
extern int myAge;
```

hello.h

```
#include "hello.h"
int myAge;

int f()
{
  myAge = 25;
}
```

hello.c

```
#include "hello.h"

int g()
{
  int m = myAge;
}
```

bye.c

Note: Also put shared "#define ..." in .h files

Numbers in binary

- base 2
- digits are 0 + 1

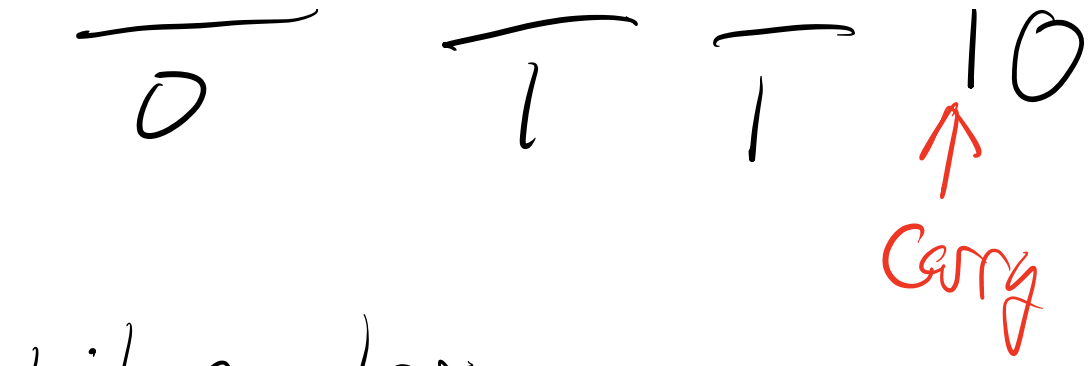
16's 8's 4's 2's 1's

10110

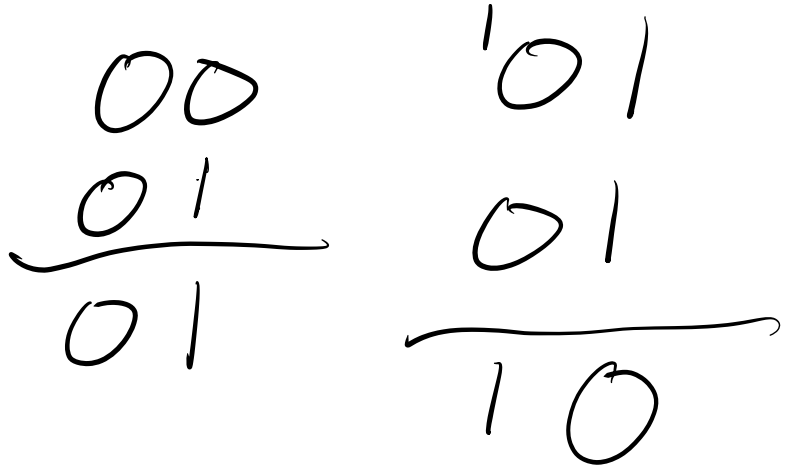
Adding binary numbers:

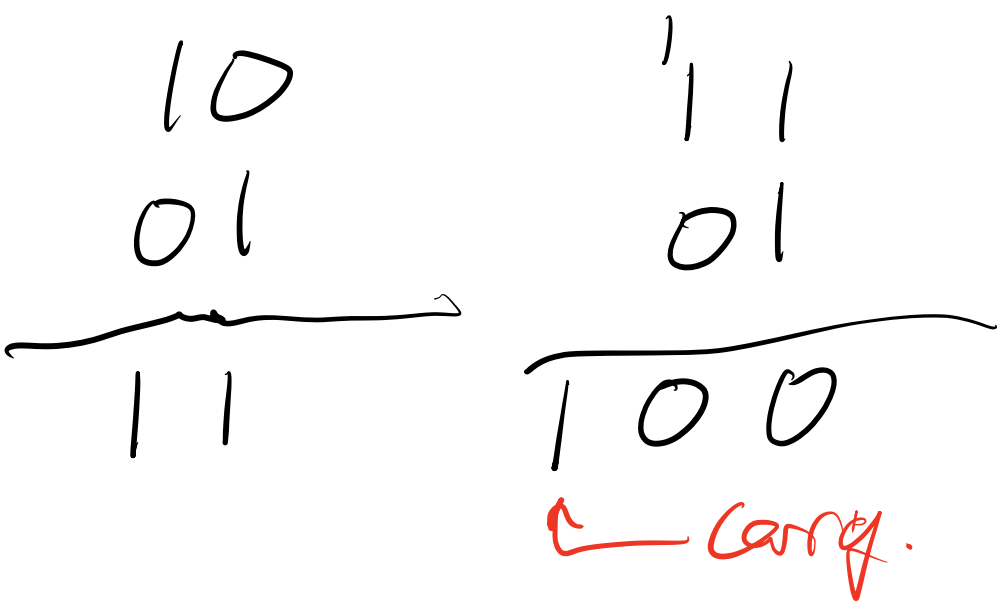
1 bit numbers:

	0	1	0	1
+	0	0	1	1



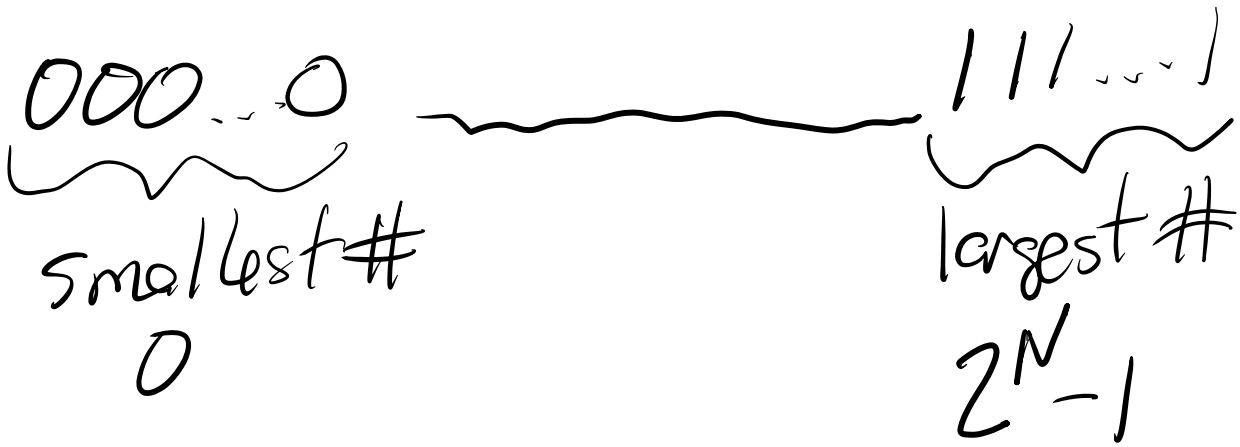
2 bit numbers:





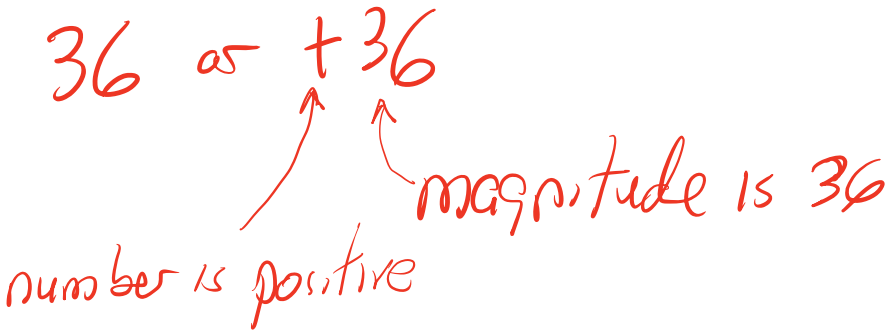
N bit number:

- assume unsigned



How about negative numbers?

- Humans represent signed numbers using "sign + magnitude".



What is the algorithm for adding signed numbers represented as sign + magnitude?

– it's complicated!

```
  23
+ 14
----
  37
```
} just adding

```
  23
+-14
```
} subtraction

```
 -23
+ 14
```
} subtraction and flip sign

-23
$+-14$

} add and, flip the sign.

Computers don't use sign + magnitude!