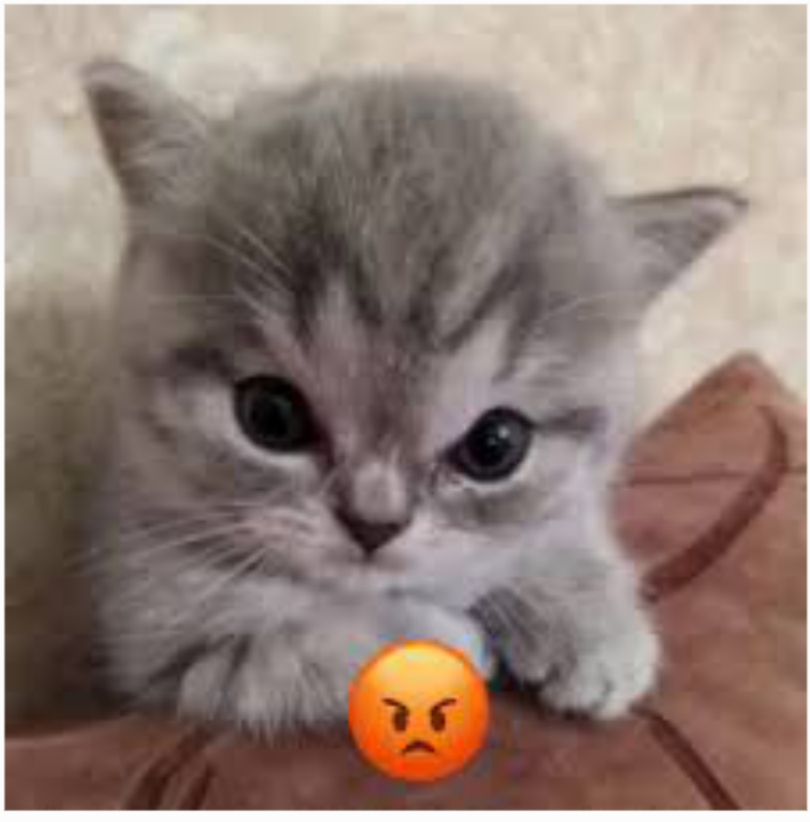
10. Memory API



Types of memory

Stack

```
void func() {
  int x; // declares an integer on the stack
  ...
}
```

- allocation / deallocations : managed implicitly by the compiler
- called automatic memory

Heap

```
void func() {
  int *x = (int *) malloc(sizeof(int));
  ...
}
```

- for long-lived memory
- allocations / deallocations : explicitly handled by programmer
  => heavy responsibility
- malloc()
  - returns the address which is then stored on the stack for sue by the program

Common Errors

Forgetting to Allocate Memory

: Memory needs to be allocated before you call them

```
char *src = "hello";
char *dst;          // oops! unallocated
strcpy(dst, src); // segfault and die
```

```
char *src = "hello";
char *dst = (char *) malloc(strlen(src) + 1);
strcpy(dst, src); // work properly
```

Not allocating enough memory -> buffer overflow

Forgetting to initialize Allocated Memory

could be harmful : former data leak

Forgetting to free memory -> run out of memory

When you are done with a chunk of memory, you should make sure to free it.

- Using a garbage-collected language doesn't help here :
  if you still have a reference to some chunk of memory, no garbage collector will ever free it, and thus memory leaks remain a problem even in more modern languages

When the process dies, the OS will clean up all of its allocated pages and thus no memory leak will take place.

Freeing Memory before you are done with it

: double free -> result : undefined

Freeing memory repeatedly

Calling free() incorrectly

mmap()

- create an anonymous memory region which is not associated with any particular file but rather with swap space.
- Can then also be treated like a heap and managed as such.