Free

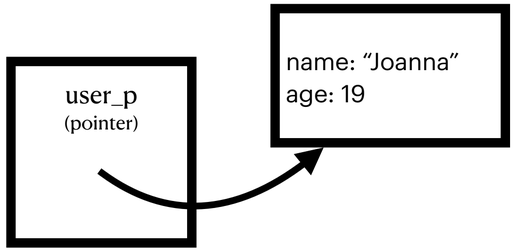
An equal and opposite reaction

- Since the program doesn't manage heap memory for us, we need to do that ourselves.
- When a program terminates, the operating system reclaims all its memory.
- However, if we were to write larger programs that ran for longer periods of time, our memory usage will steadily grow with the amount of times we called **`malloc`**, causing performance issues.
- So, for every **`malloc`**, there must be a **`free`**.
- But... when do we call **`free`**?

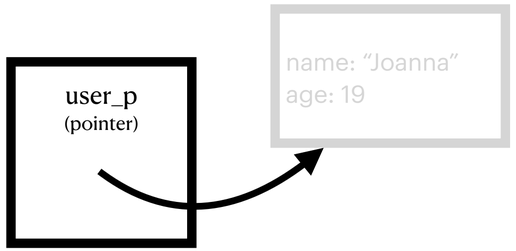
Free

Problem

We allocate some memory for a struct

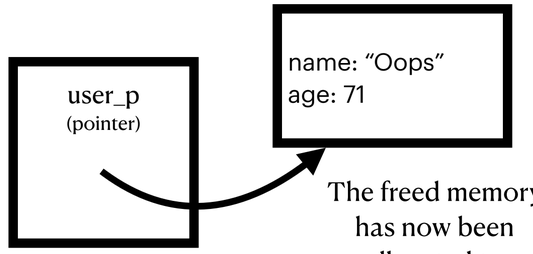


We free the piece of memory



... A bunch of mallocs later

We try to access the same piece of memory, but the data we retrieve isn’t what we expect.



The freed memory has now been allocated to something else

- **Issue:** You must never access freed memory.
  - When we return memory back to the system via **`free`**, it is now free to allocate that piece of memory to something else.
- An access-after-free error occurs when we try to *dereference* a pointer pointing to a freed address.
  - You can still change the address stored inside the pointer itself (reassign it)
- **Solution:** Only free memory you know you’ll never need to use again.