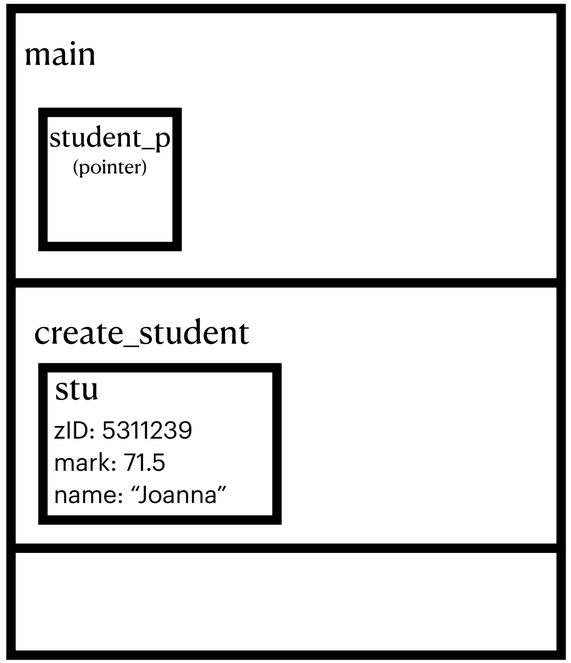
COMP1511 Week 8

malloc and linked lists

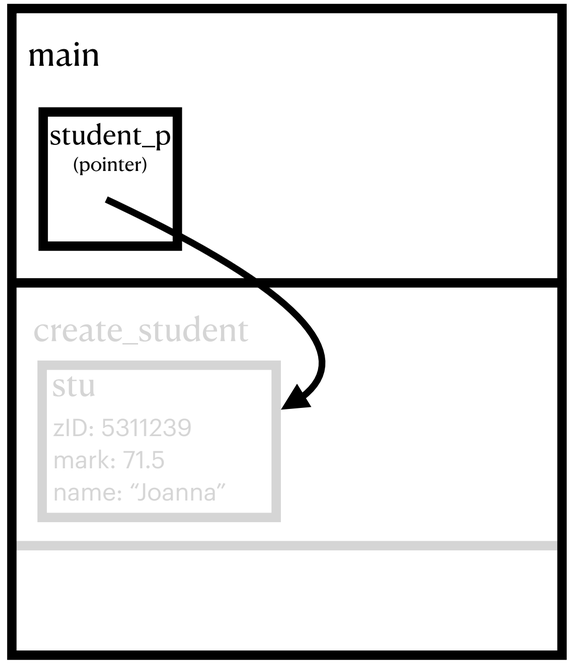
Joanna Lin

Returning an Address

Problem



`create_student` creates a `struct student` called `stu` and initialises its fields.



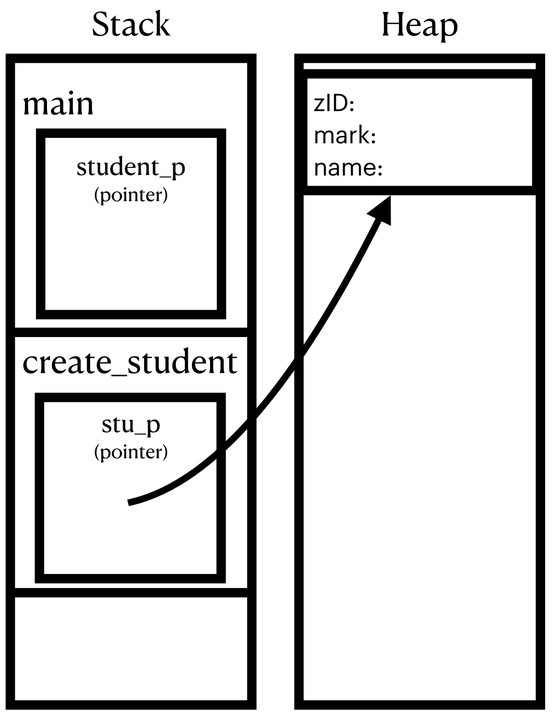
when we exit `create_student`, its memory is deallocated and the address of `stu` is stored in `student_p`. now `student_p` is left pointing at memory that is unsafe to access.

The Heap

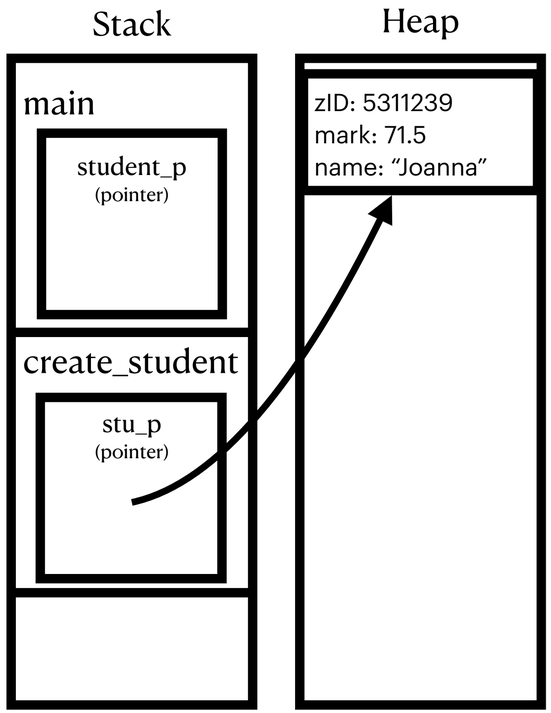
Self-Memory Management

- So far, all memory we've work with have been allocated on a part of computer memory called the **stack**.
  - As soon as a block of memory goes 'out of scope', it gets deallocated (destroyed) by the program.
  - This is good because we don't have to manage memory ourselves — the program does it for us and so memory usage stays efficient. However, it's not very helpful when we want to return an address!
- Instead, we take advantage of a separate block of memory called the **heap**.
  - The program will not manage the heap for us. This means that we can validly return addresses of memory we allocate on the heap from functions.
  - This means we must free (deallocate) memory on the heap when we no longer need that block of memory, otherwise we'll create **memory leaks** (more on this next week). This means that the block of memory can't be used by other processes even though we'll never use what is stored there ever again.
  - We use the function `malloc` to allocate memory, and `free` to deallocate memory.

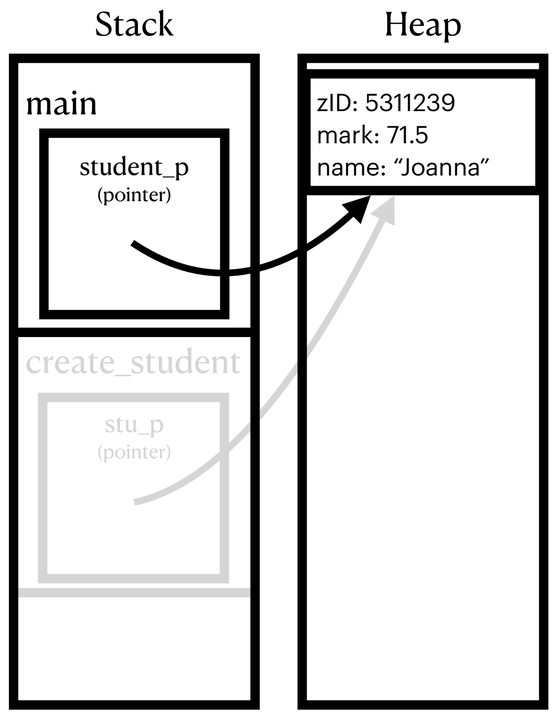
Memory Model



`malloc` allocates a block of memory on the heap for a **`struct student`**



The struct fields are initialised by dereferencing the **`stu_p`** pointer



`create_student` returns the address to the heap-allocated struct, which is copied into the **`student_p`** variable. **`student_p`** points to the heap-allocated struct.