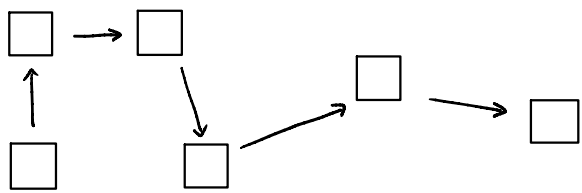
Recap: Linked lists

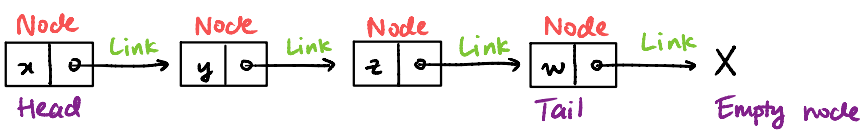
A linked list is a type of data structure that addresses the primary limitation of arrays: they can hold a variable amount of elements. To achieve this, rather than allocating a contiguous chunk of memory, we allocate tiny chunks at a time for each element, and use pointers to link them together.

Array: contiguous, but fixed in size once created*



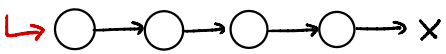
Linked list; scattered through memory but not fixed in size; can grow/shrink

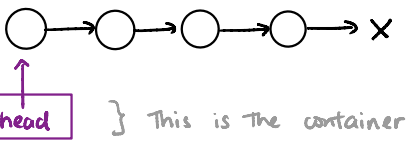
Anatomy of a linked list is as follows:



A lot of what we will do in this course is extending the notion of a linked list to more exotic data structures.

Ex. Two ways of representing a linked list:

①  (normal list)

②  (con-list: containerised list)

This is the container

Pros of ②? I can add things to the container!
e.g. the tail, length, ...
(at the expense of bookkeeping overhead.)

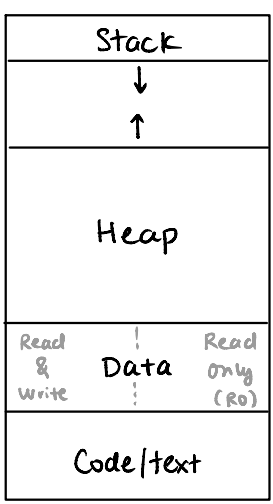
Recap: C memory semantics.

In C, there are 2 forms of memory allocation:

1. **Stack allocations**: When you declare simple variables, structs or arrays without using malloc, i.e. via a line like "`int numbers[10];`", the memory allocated is placed in a location called the stack. This is local to the current function, which is why you cannot return an array allocated as previously and have it persist after returning.

2. **Heap allocations**: When you use malloc or calloc, the memory allocated is placed in a memory location known as the heap. This is a global memory region to your program, so things stored in the heap survive function returns. There is also typically more heap available than stack. The drawback is that you have to make sure you free any heap memory you allocate, as it is not automatically reclained like stack memory.

More broadly, a simplified view of a C program's memory might be:



Heap and stack are as described earlie
Though one thing to note is that the stack is split into frames, while the heap is shared with all programs (more or less).

Data segment; global variables, static variables (e.g. string literals such as "hello COMP2521")
Note that it has 2 halves: a read only and a read & write section. String literals go in RO!

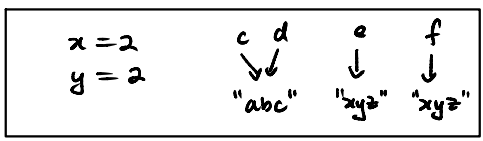
Executable code for programs; all read only.

Ex.

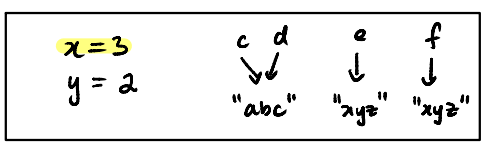
```
int x, y;
char *c, *d, *e, *f;

x = y = 2;
c = d = "abc";
e = "xyz"; f = "xyz";

x++;
*c = 'A';
```

] string literals ⇒ stored in data RO segment.

After assignments:



After x++:



What happens at the final line and why? No! c is a read-only ptr. (Remember they are string lits.)