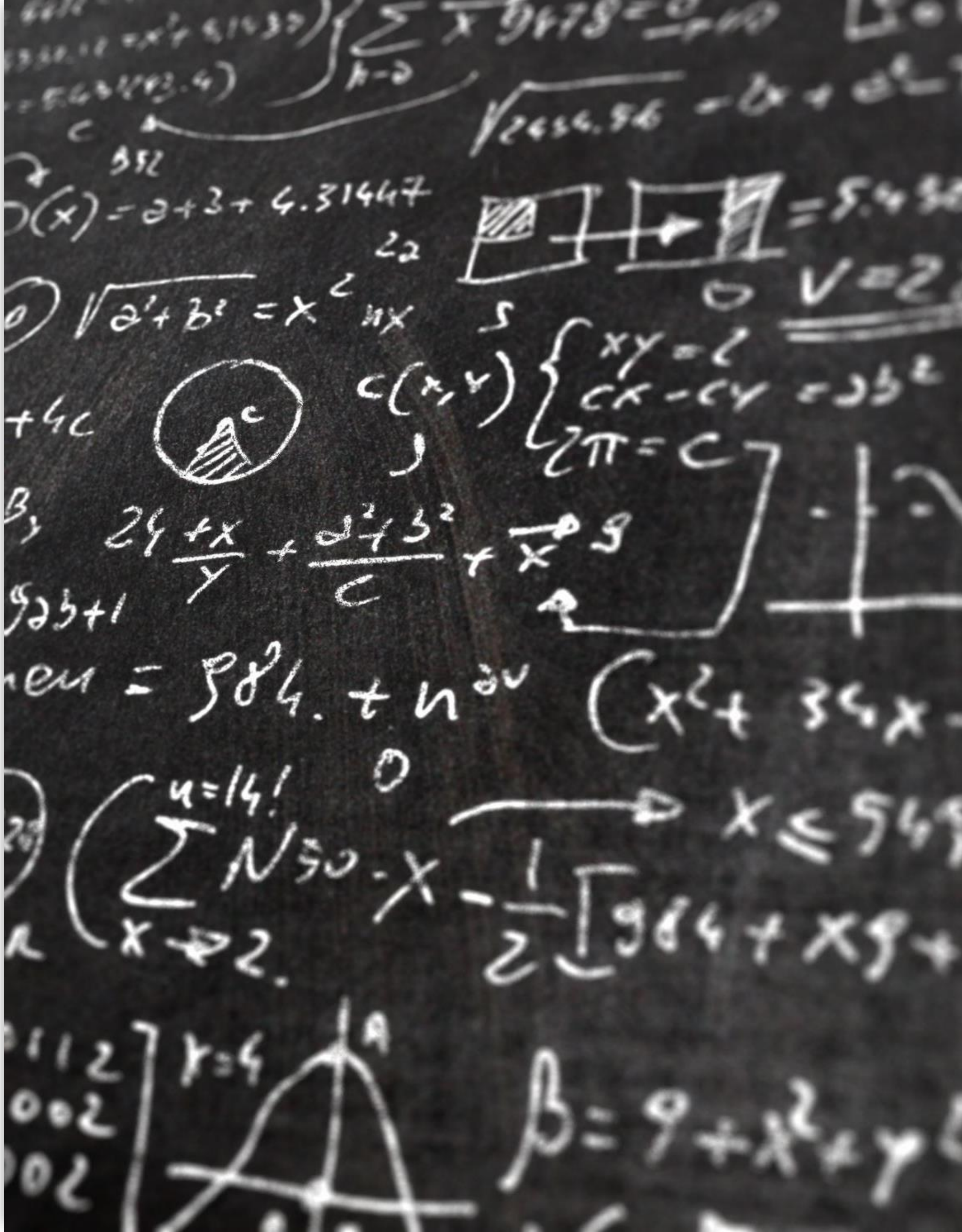
School of Computing & Communications | Lancaster University

SCC.111 Software Development – Lecture 18: Dynamic data structures

Adrian Friday, Nigel Davies, Hansi Hettiarachchi, Saad Ezzini

This lecture

- What’s wrong with ‘basic’ variables
- Why we use dynamic data structures
- Using pointers and dynamic memory to build a data structure
- A worked example



We’ve covered several variable types

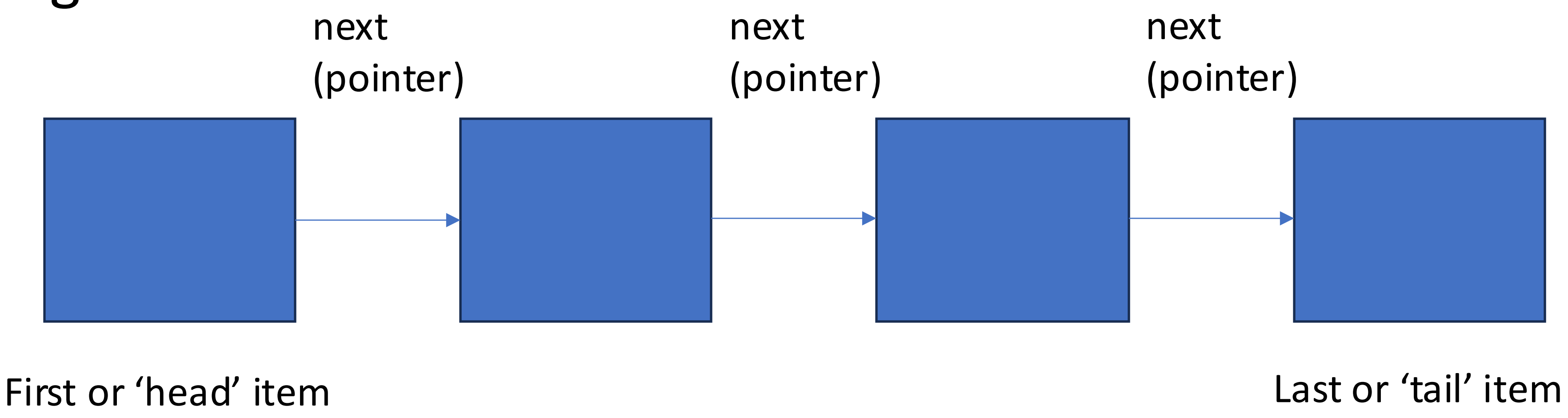
- Basic types (int, char, float, double, etc.)
- Arrays of basic types (fixed length sequences)
- Compound types (structs)
- And started with allocating dynamic memory for these at runtime (malloc, free)

But consider this:

- What happens if the data we want to process is of unknown size?
  - We’d like our application to work despite flexible sized data!
- Or we need a more powerful ways of organising the data to make it quicker to search or sort?
  - Simple arrays lend themselves to linear organised data (e.g. lists), ideally of known size…
  - What about implementing more advanced data structures?

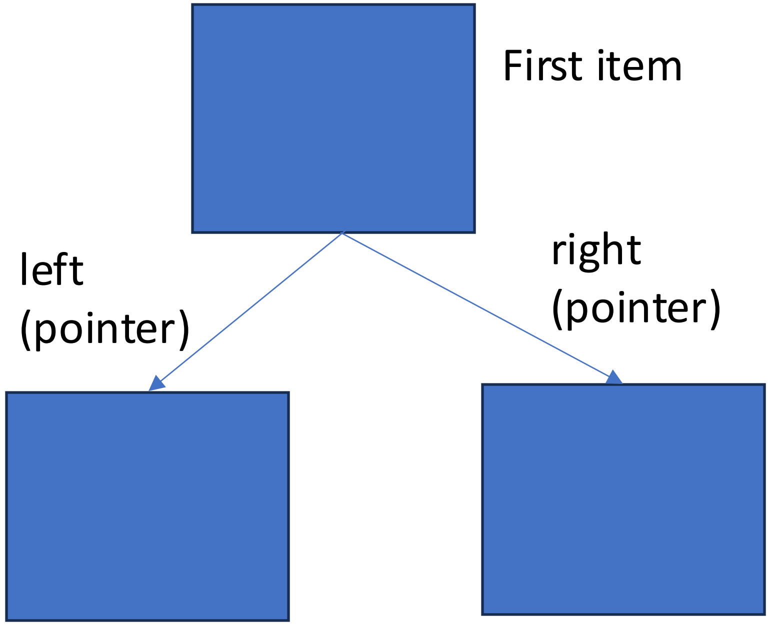
Fortunately, compound variables and dynamic memory… *also links to ADTs in SCC.121!*

- Use dynamic memory to allocate elements in data structures
- Use pointers between dynamic instances to organise our data structure
- … e.g. a list:



Fortunately, compound variables and dynamic memory… *also links to ADTs in SCC.121!*

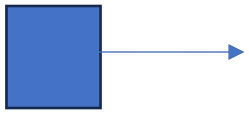
- Use dynamic memory to allocate elements in data structures
- Use pointers between dynamic instances to organise our data structure
- … e.g. or a tree:



Building a dynamic ‘singly linked’ list

- We can start with ‘no items’
- And build up our data structure one item at a time.

- … e.g. a list:

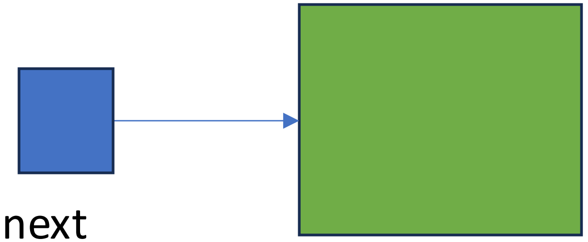


next
(pointer)
to **NULL**
(nothing)

Chaining nodes/ growing the list

- E.g. Allocate a new node (using malloc)
- ‘Chain it’ so our first item points to the new item

- … e.g. a list:



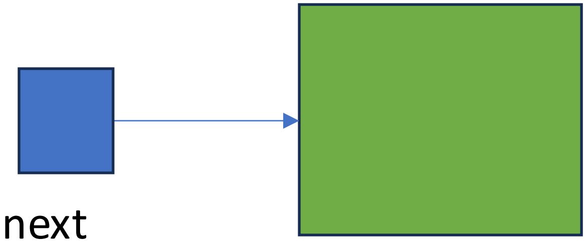
next (pointer) to **NULL** (nothing)

next (pointer) to **new node**

Chaining nodes/ growing the list

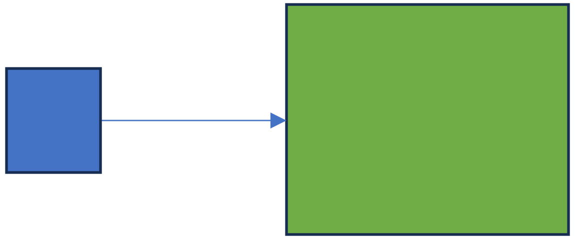
- E.g. Allocate a new node (using malloc)
- ‘Chain it’ so our first item points to the new item

- … e.g. a list:



next (pointer) to **NULL** (nothing)

next (pointer) to **new node**

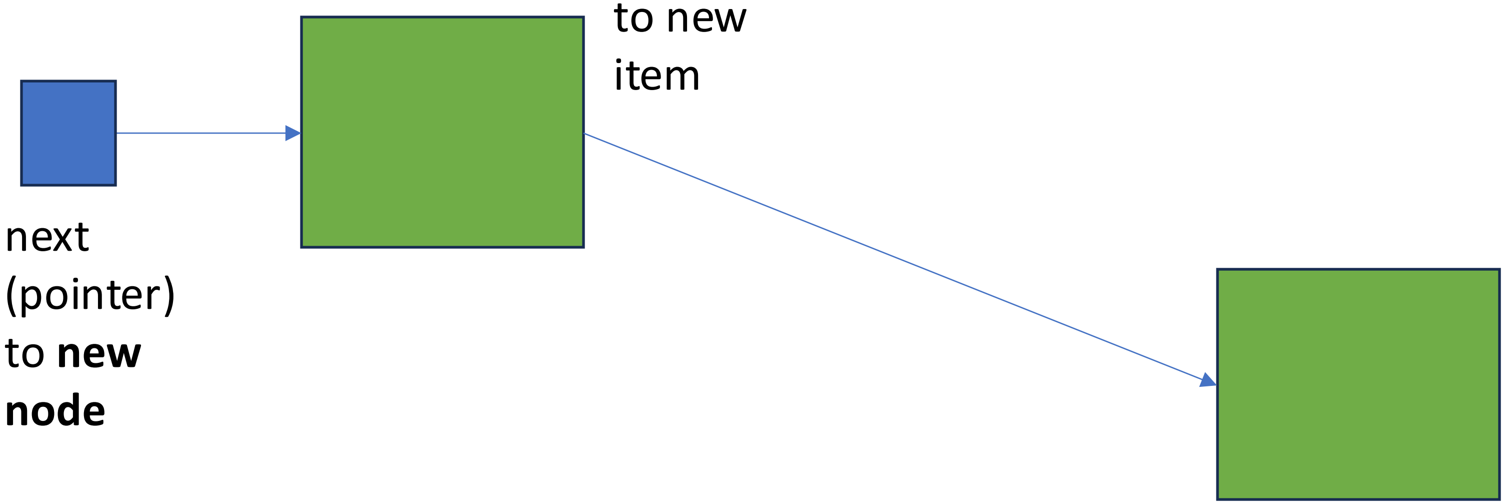


next (pointer) to **NULL** (nothing)

next (pointer) to **new node**

Chaining nodes/ growing the list

- E.g. Allocate a new node (using malloc)
- ‘Chain it’ so our first item points to the new item
- … e.g. a list:



next (pointer) to new item

next (pointer) to **new node**

next (pointer) to **NULL** (nothing)

Generally, we…

1. Allocate space for a ‘node’ (a struct) of the appropriate type
2. Find where to add our item (start, end, insertion point)
3. Adjust the pointers to stay consistent with the type of data structure we’re working with

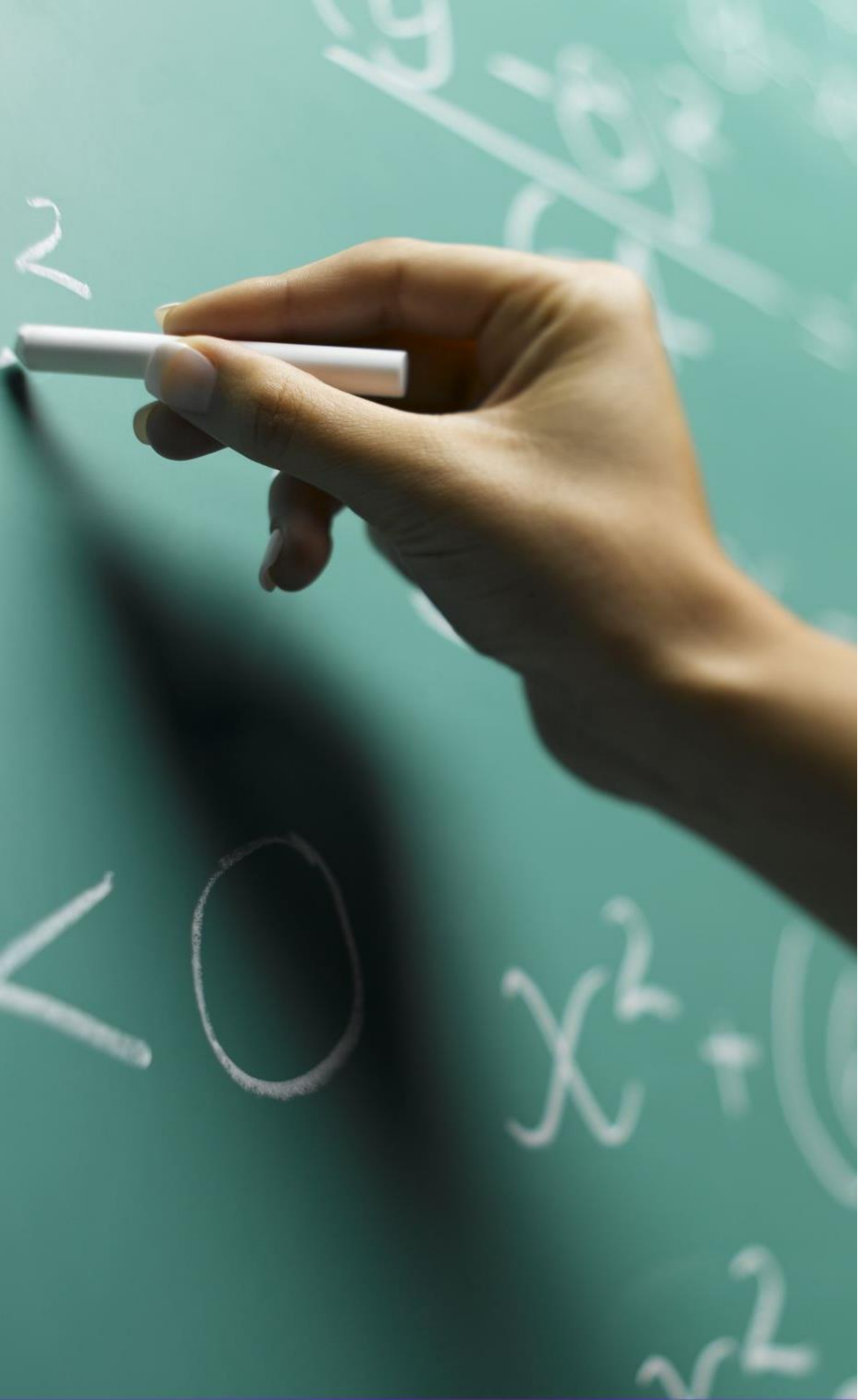
- *Working with all these pointers will take some practice to get right! (it took us lots of practice too!)*

Let’s work through a simple example

- We want to create a list of student names and colleges that we keep sorted in alphabetical order.
- We don’t know how big the list will be and we don’t want to keep resorting or moving data so we don’t want to be using an array.

The data structure we need for this

- A linked list:
  - Three operations: **insert** (add to the list), **find** (return a pointer to the item in the list) and **delete** (remove from the list)

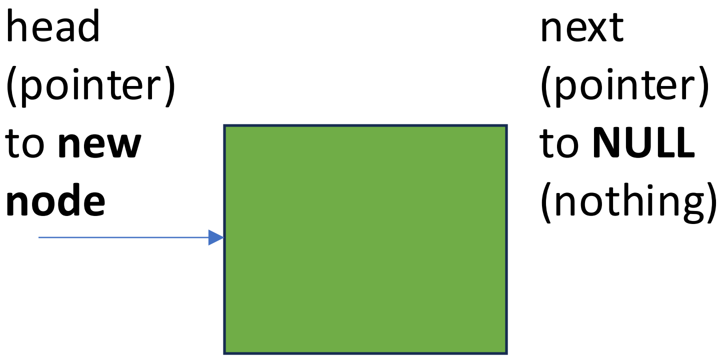


For this we need to develop 3 things:

- A good understanding of pointers
- Compound types (structs)
- Dynamic memory (malloc)

Compound variables and dynamic memory…

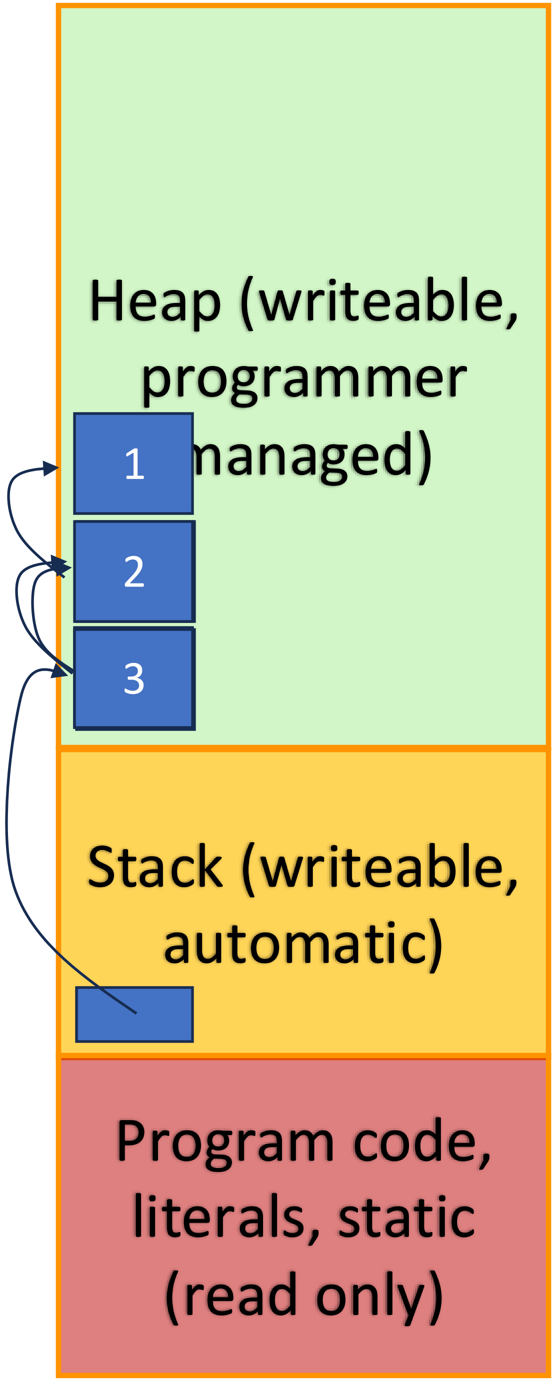
1. Declare a type for our node (struct)
2. A variable representing the pointer to the data structure
3. For each node, allocate a new node (using malloc)
4. ‘Chain it’ so our first item points to the new item


head (pointer) to **new node**
next (pointer) to **NULL** (nothing)

First or ‘head’ item

In memory…

- As the data structure grows, we allocate more ‘blocks’ of dynamic memory
- We chain these together to form our data structure
- We need to be careful to manage our pointers and hand memory back to the memory allocator(!)


Heap (writeable, programmer managed)
1
2
3
Stack (writeable, automatic)
Program code, literals, static (read only)

Let’s look at a code example

Declare our ‘node’ struct

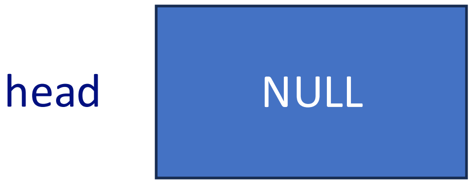
```
/* Define listItem node type */

struct listItem {
  char name[20];
  char college[20];
  struct listItem *next;
};
```

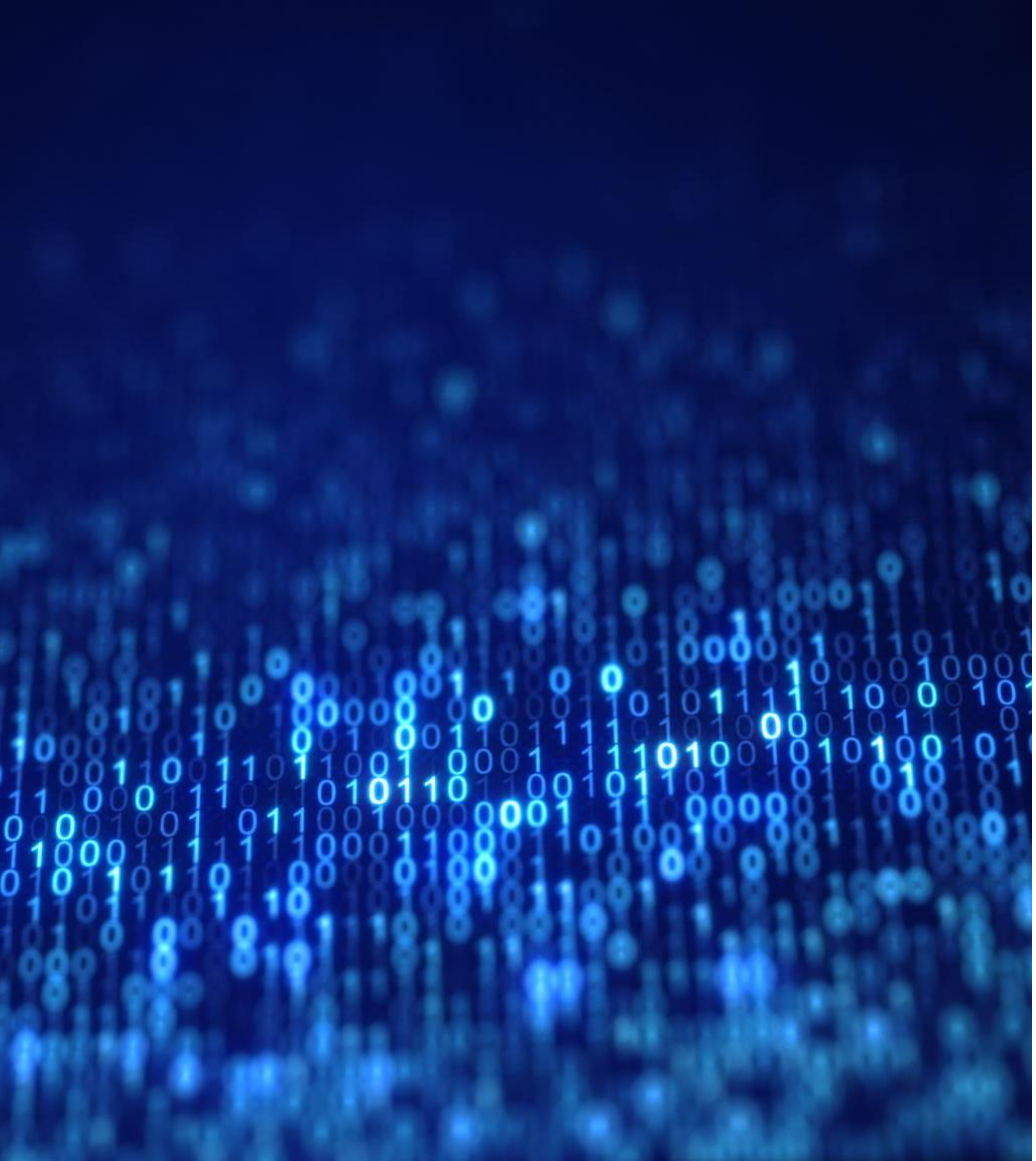
Note self-referential pointer struct listItem *next;

Declare our variable (head pointer)

```
int main()
{
  struct listItem *head = NULL;
}
```



A pointer set to point to nothing (or 0 / NULL)



Summary

- Presented an example of a dynamic data structure (a linked list)
- How structures can contain pointers to the same or other structures
- How malloc/ free are used to create space for items
- Practice pointer manipulation