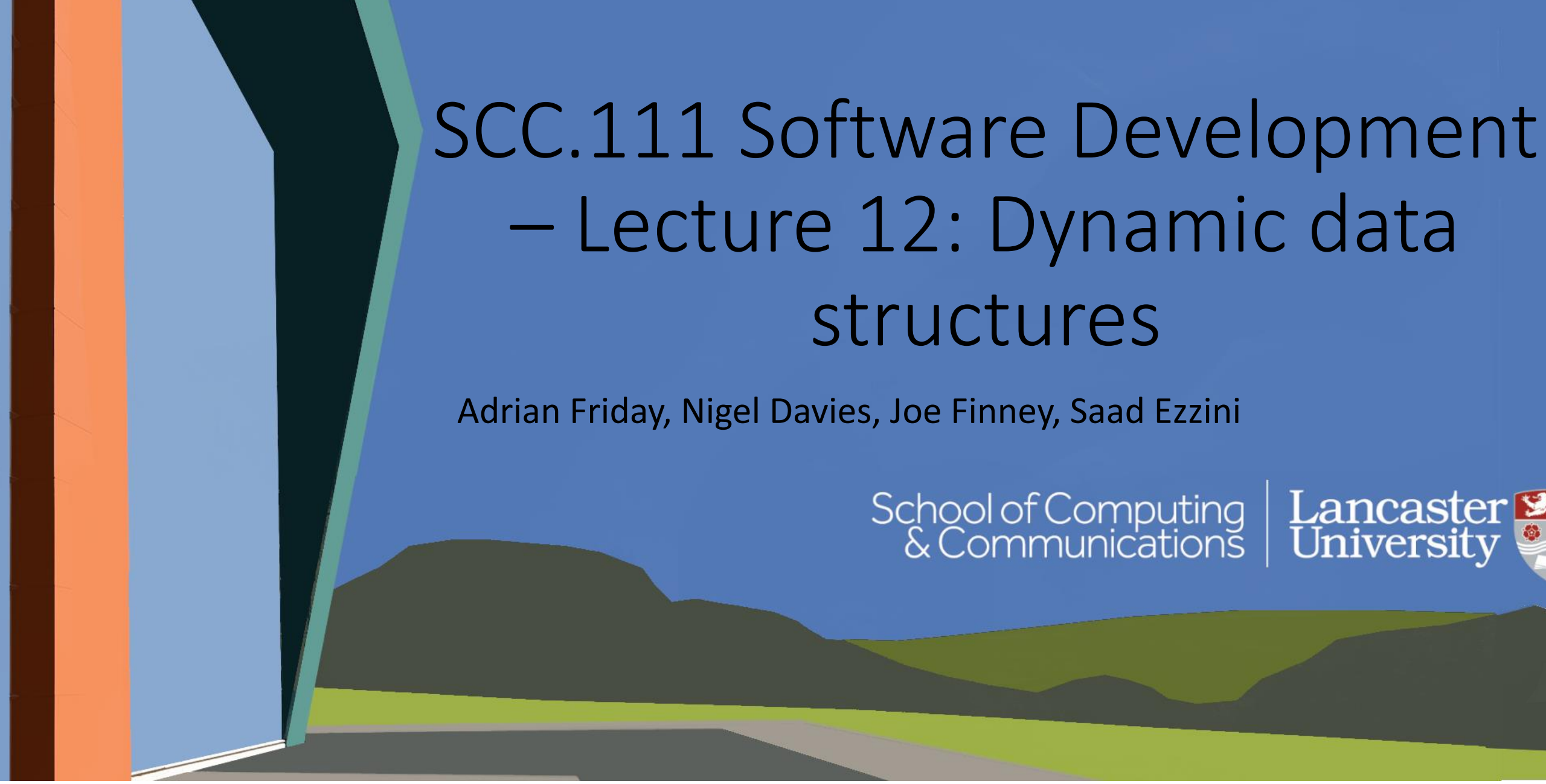


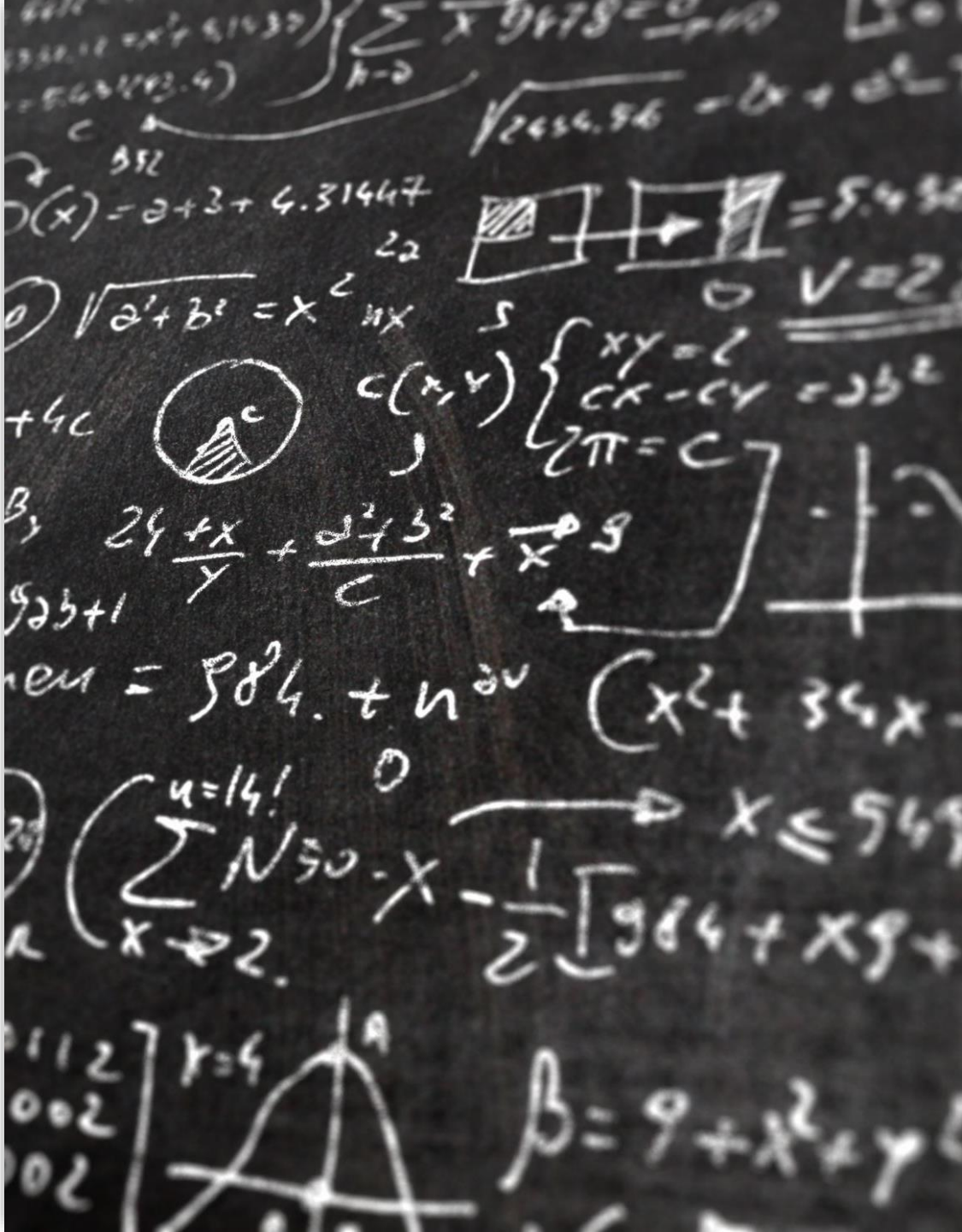
SCC.111 Software Development – Lecture 12: Dynamic data structures

Adrian Friday, Nigel Davies, Joe Finney, Saad Ezzini

School of Computing & Communications | Lancaster University

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)

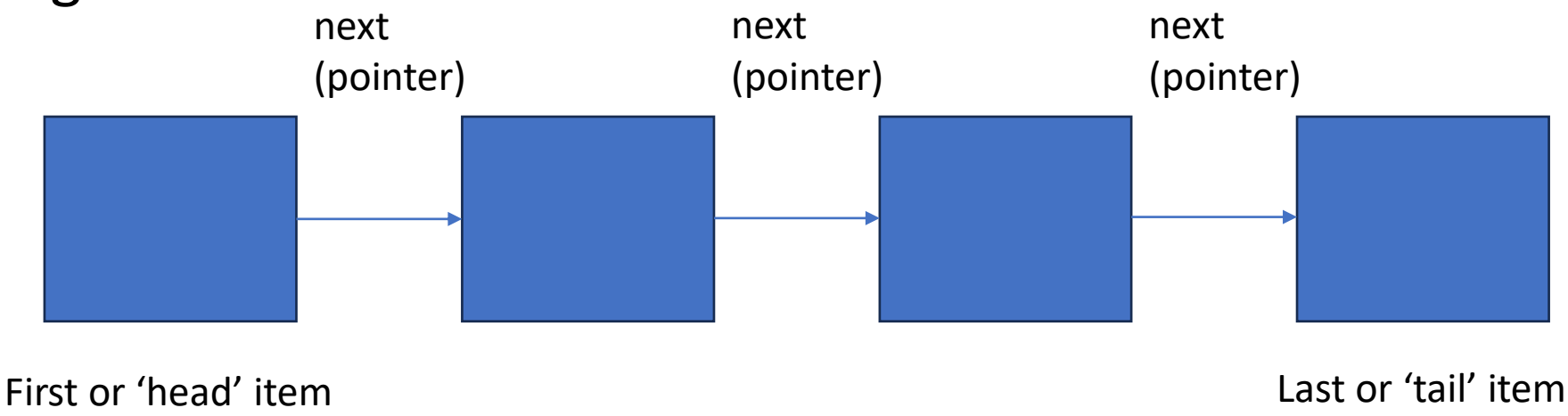
But consider this:

- 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:



Building a dynamic ‘singly linked’ list

- We can start with ‘no items’
- And build up our data structure one item at a time (*enqueue an item*)!

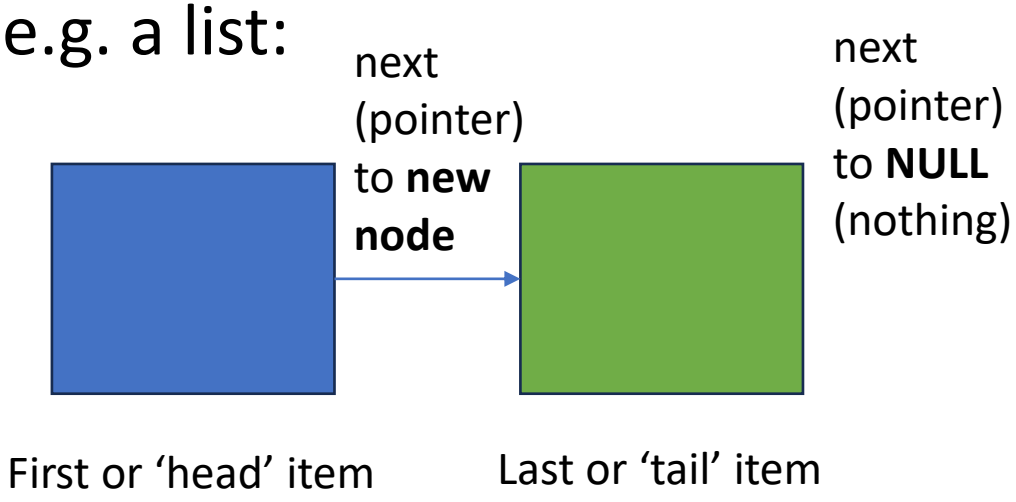
- … e.g. a list:

First item is both ‘head’ and ‘tail’

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:



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!)*

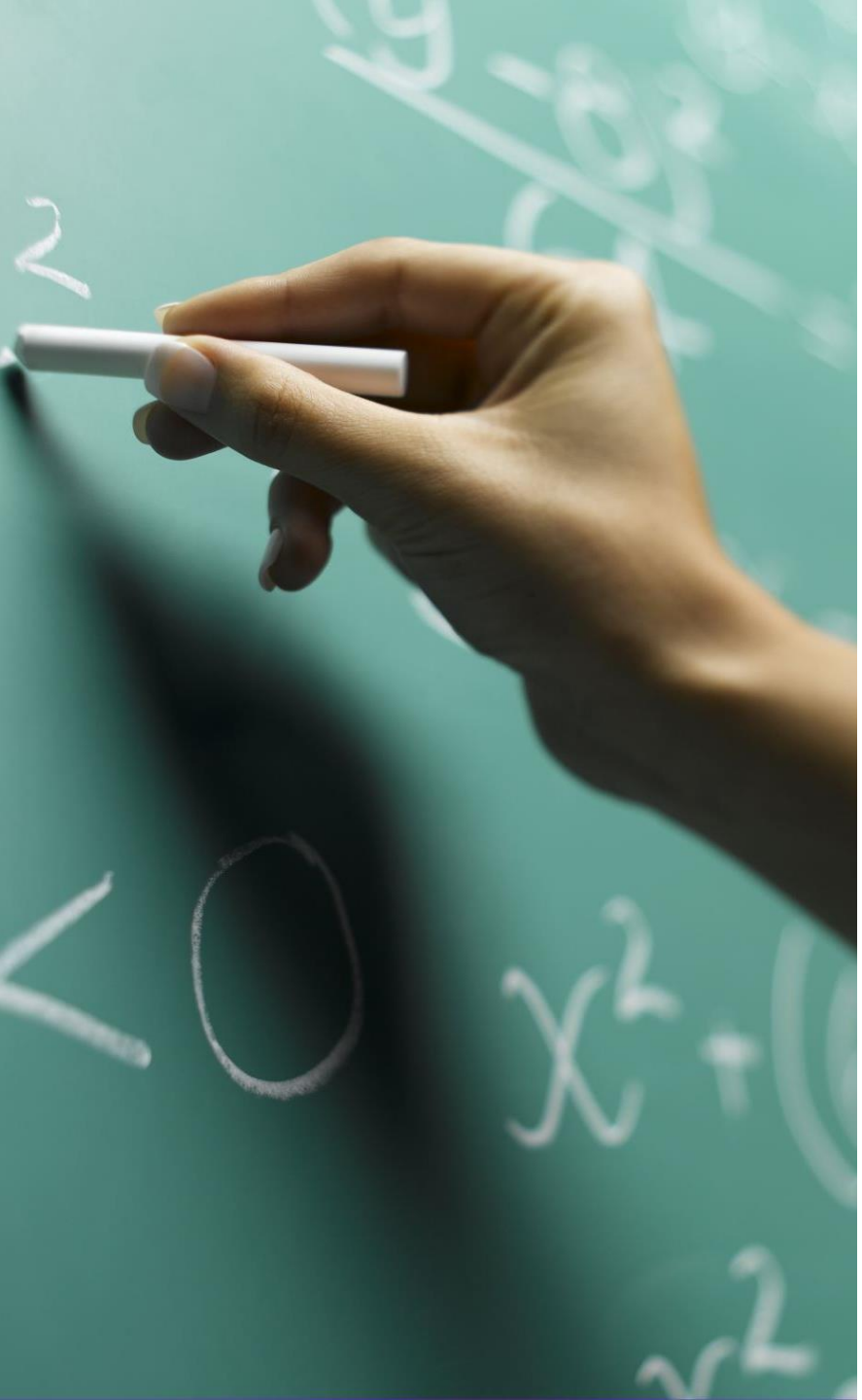
Let’s work through a simple example

- We’ll create a simple dynamic collection, from first principles…

- Operations on a collection:
  - **add** items in a collection
  - **remove** items from a collection

  - We might also want to **find** an item in a collection, but this is for another day…

- i.e. A dynamic 'array' that grows and shrinks as needed…

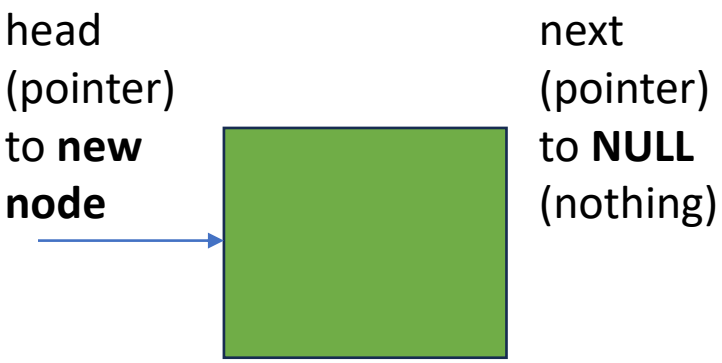


For this we need to develop 3 things:

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

Compound variables and dynamic memory…

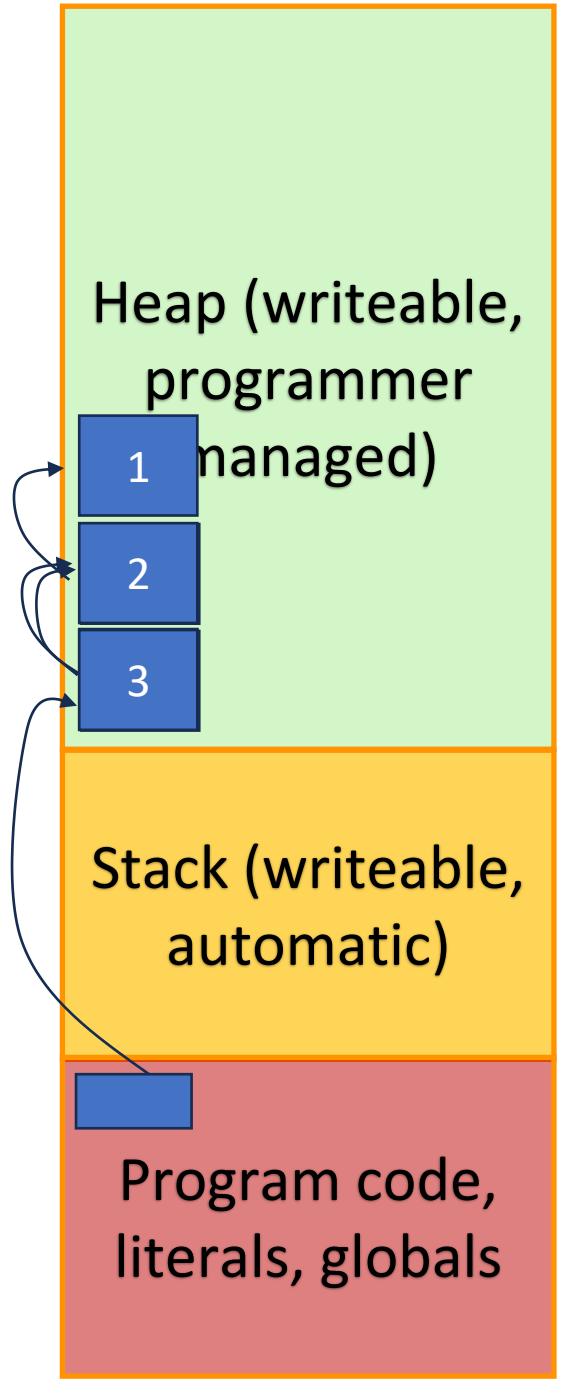
1. Declare a type for our node (struct)
2. A variable representing the pointer to the data structure
3. For each node, allocate a new node (using malloc)
4. ‘Chain it’ so our first item points to the new item



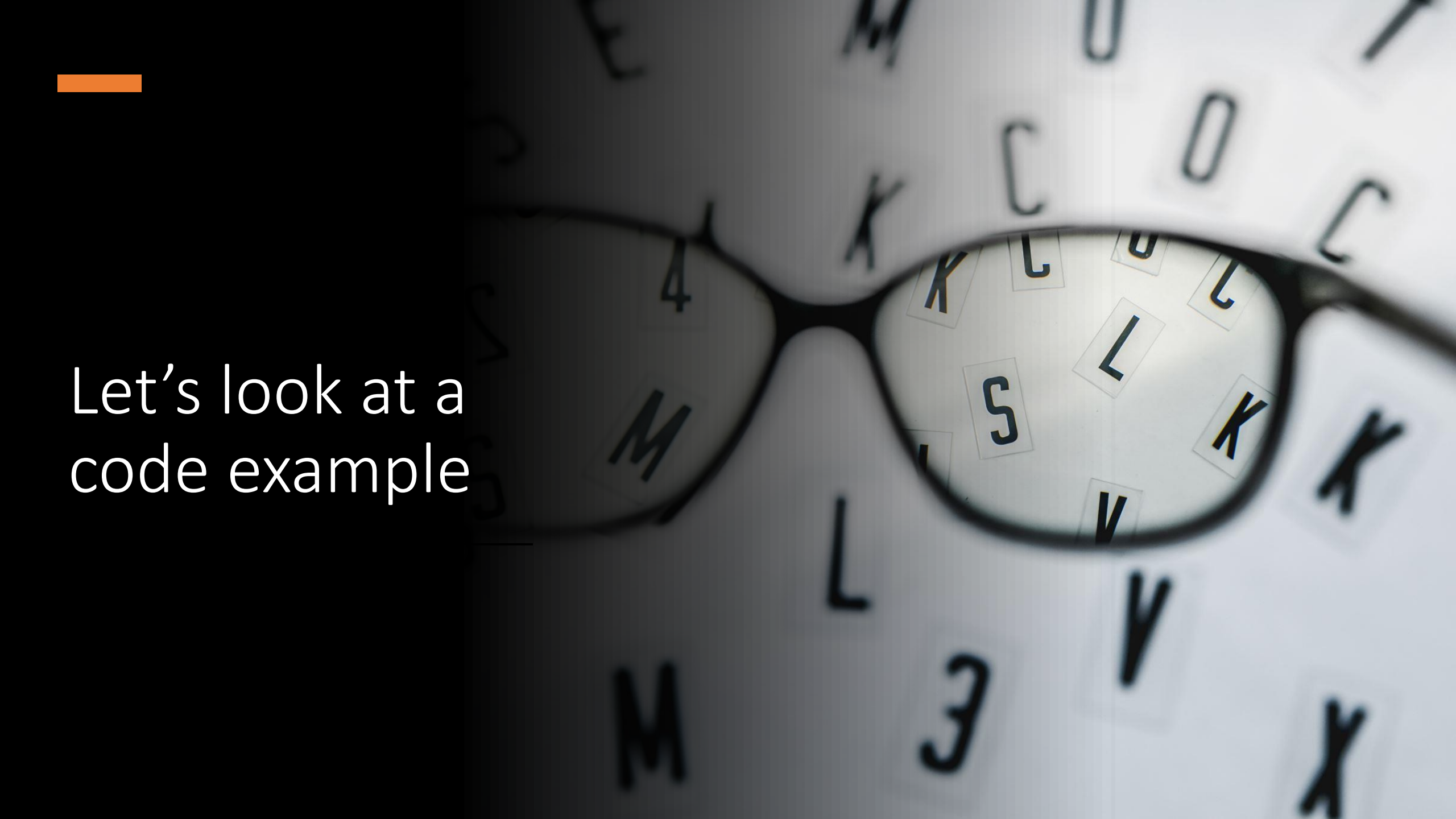
First or ‘head’ item

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(!)



Let’s look at a code example



Declare our ‘node’ struct

```
struct node
{
    int age;
    struct node *next;
};
```

Note self-referential pointer struct node *next;

Declare our variable (head pointer)

```
struct node *collection = NULL;
```

To add a node:

```
void add_node(struct node *n)
{
    n->next = collection;
    collection = n;
}
```

To remove a node:

```
void remove_node(struct node *n)
{
    struct node *prev;
    struct node *current;

    if (collection == NULL)
        return;

    if (collection == n)
        collection = n->next;

    prev = collection;
    current = collection->next;

    while(current != NULL)
    {
        if (current == n)
        {
            prev->next = current->next;
            free(n);
        }

        prev = current;
        current = current->next;
    }
}
```

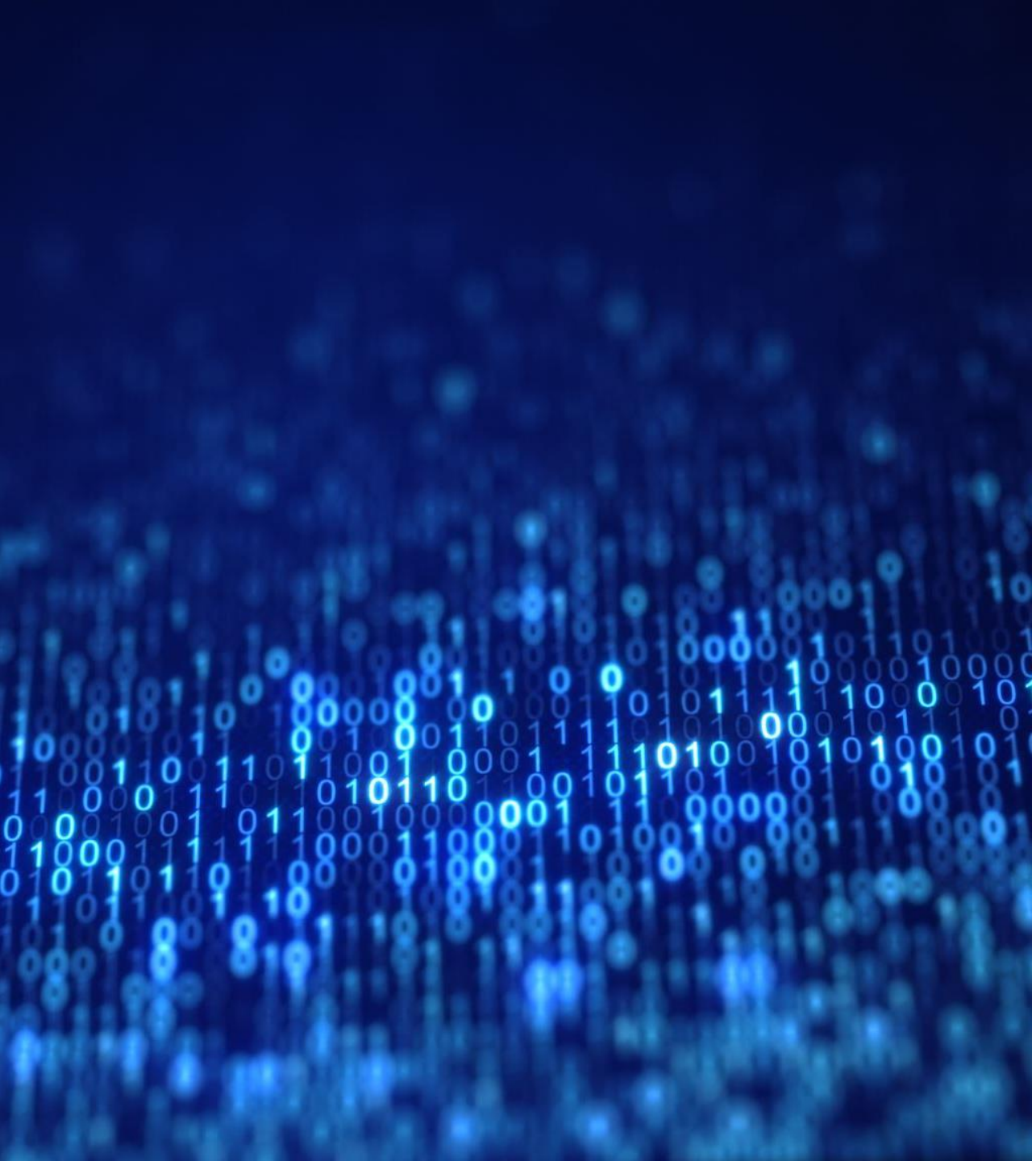
In more detail…

```
struct node *create_new_node()
{
    struct node *n = (struct node *) malloc(sizeof(struct node));
    n->next = NULL;
    n->age = 0;
    return n;
}
```

Allocate new node on the heap and get pointer

How large is a struct node in memory?

What is the stuff in brackets before malloc all about?



Summary

- Presented an example of a dynamic data structure (a collection)
- How structures can contain pointers to the same or other structures
- How malloc / free are used to create space for items
- Practice pointer manipulation
- Start to understand how ADTs support computer programs…