Linked Lists

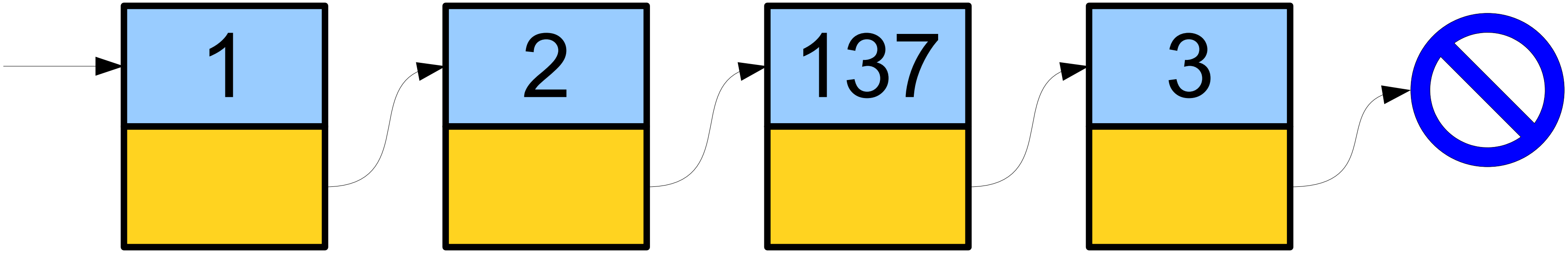
Part One

Outline for Today

- ***Linked Lists, Conceptually***
  - A different way to represent a sequence.
- ***Linked Lists, In Code***
  - Some cool new C++ tricks.

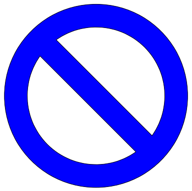
Linked Lists at a Glance

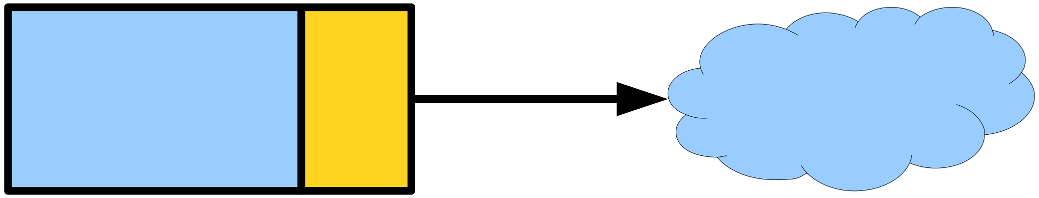
- A ***linked list*** is a data structure for storing a sequence of elements.
- Each element is stored separately from the rest.
- The elements are then chained together into a sequence.
- The end of the list is marked with some special indicator.



A Linked List is Either…

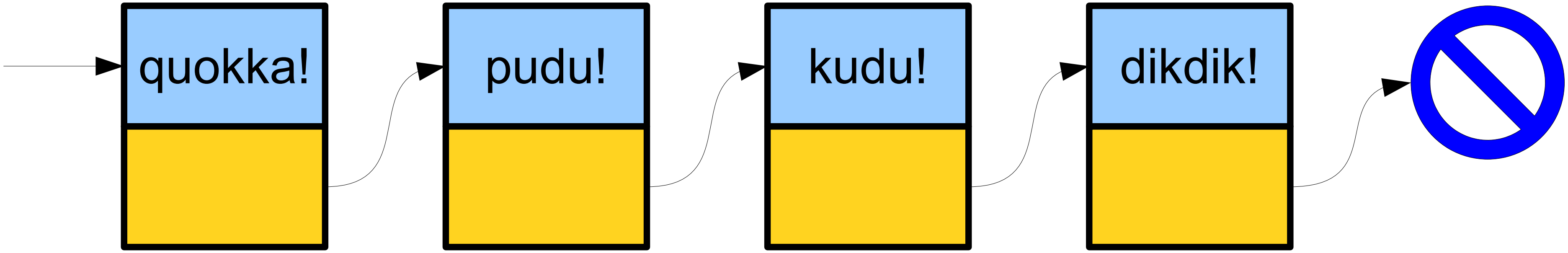
…an empty list,
or…





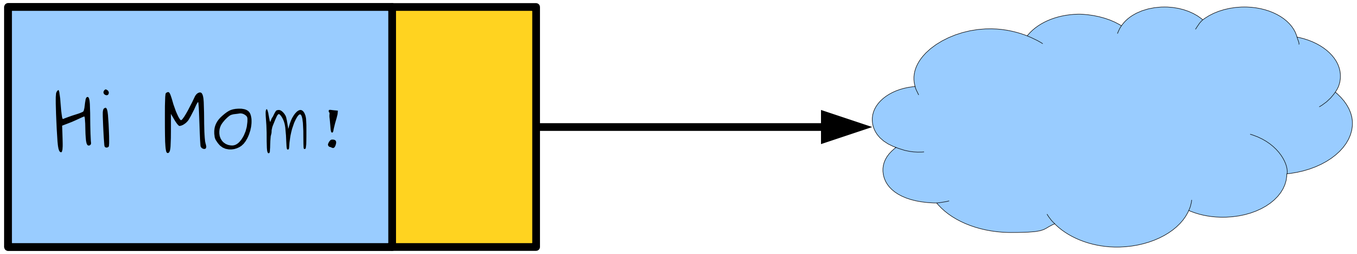
a single cell…

… that points at
another linked list.



Representing Linked Lists

```
struct Cell {
    string value;
    Cell* next;
};
```



a single cell…

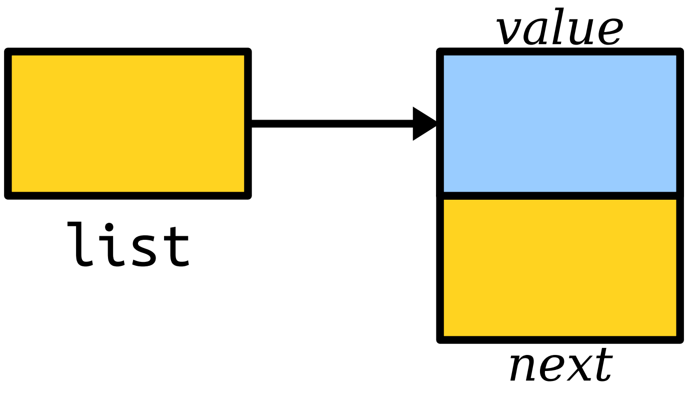
… that points at another linked list.

```
struct Cell {
    string value;
    Cell* next;
};

Cell* list = new Cell;
```

We just want a single cell, not an array of cells. To get the space we need, we'll just say `new Cell`.

Notice that `list` is still a `Cell*`, a pointer to a cell. It still says "look over there for your `Cell`" rather than "I'm a `Cell`!"



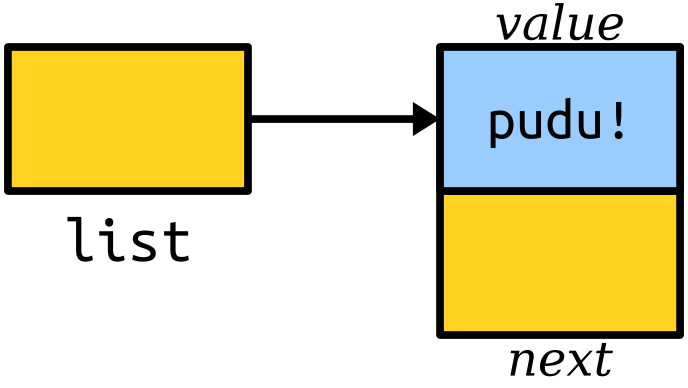
Yes, it's confusing that C++ uses the same types to mean "look over there for an array of `Cells`" and "look over there for a single `Cell`."

```
struct Cell {
    string value;
    Cell* next;
};

Cell* list = new Cell;
list->value = "pudu!";
```

Because list is a pointer to a `Cell`, we use the arrow operator `->` instead of the dot operator.

Think of `list->value` as saying “start at `list`, follow an arrow, then pick the `value` field.”

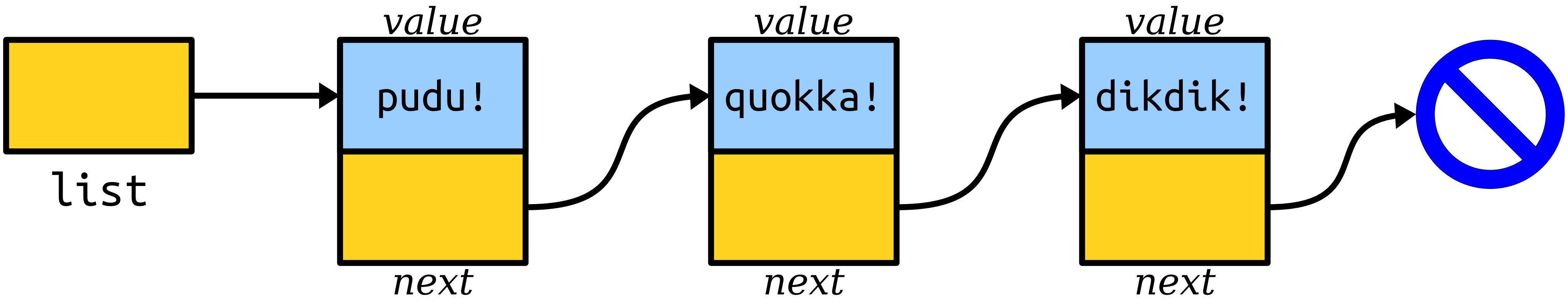


```
struct Cell {
    string value;
    Cell* next;
};

Cell* list = new Cell;
list->value = "pudu!";
list->next = new Cell;
list->next->value = "quokka!";
list->next->next = new Cell;
list->next->next->value = "dikdik!";
list->next->next->next = nullptr;
```

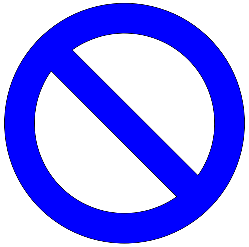
C++ uses the **nullptr** keyword to mean “a pointer that doesn’t point at anything.”

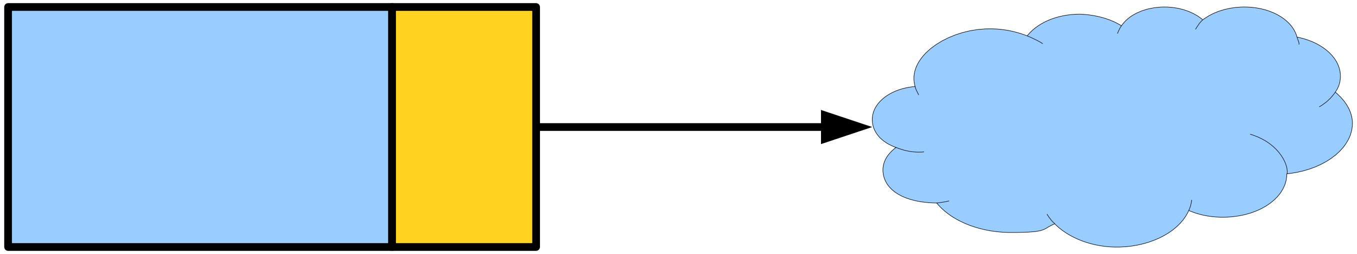
(Older code uses NULL instead of **nullptr**; that’s also okay, but we recommend **nullptr**.)



A Linked List is Either…

…an empty list,
represented by
`nullptr`, or…





a single linked list
cell that points…

… at another linked
list.

Measuring a Linked List

Printing a Linked List

Time-Out for Announcements!

Looking Ahead: Partners

- Assignment 6 (the one out this week) must be completed individually.
- Assignments 7 and 8 may be done either individually or with a partner.
- Your partner must be in the same section as you.
- If you know someone you want to work with but are not in their section, ping Jonathan by Wednesday so we can make the swap.

Tone Matrix Contest

- We’re holding a Tone Matrix contest, analogous to the Recursive Drawing contest we ran earlier in the quarter.
- Interested in entering?
  - Record a video using your Tone Matrix program. Be creative!
  - Post a link to the video on the EdStem thread set up for the contest.
- Deadline to submit is next ***Monday, March 3*** at ***1:00PM***.
- We’ll award a small number of prizes to popular entries. This is 100% optional and has no bearing on your course grade.

```
lecture = lecture->next;
```

Building a Linked List

(without hardcoding it)

Cleaning Up a Linked List

Endearing C++ Quirks

- If you allocate memory using the `new[]` operator (e.g. `new int[137]`), you have to free it using the `delete[]` operator.

```
delete[] ptr;
```

- If you allocate memory using the `new` operator (e.g. `new Cell`), you have to free it using the `delete` operator.

```
delete ptr;
```

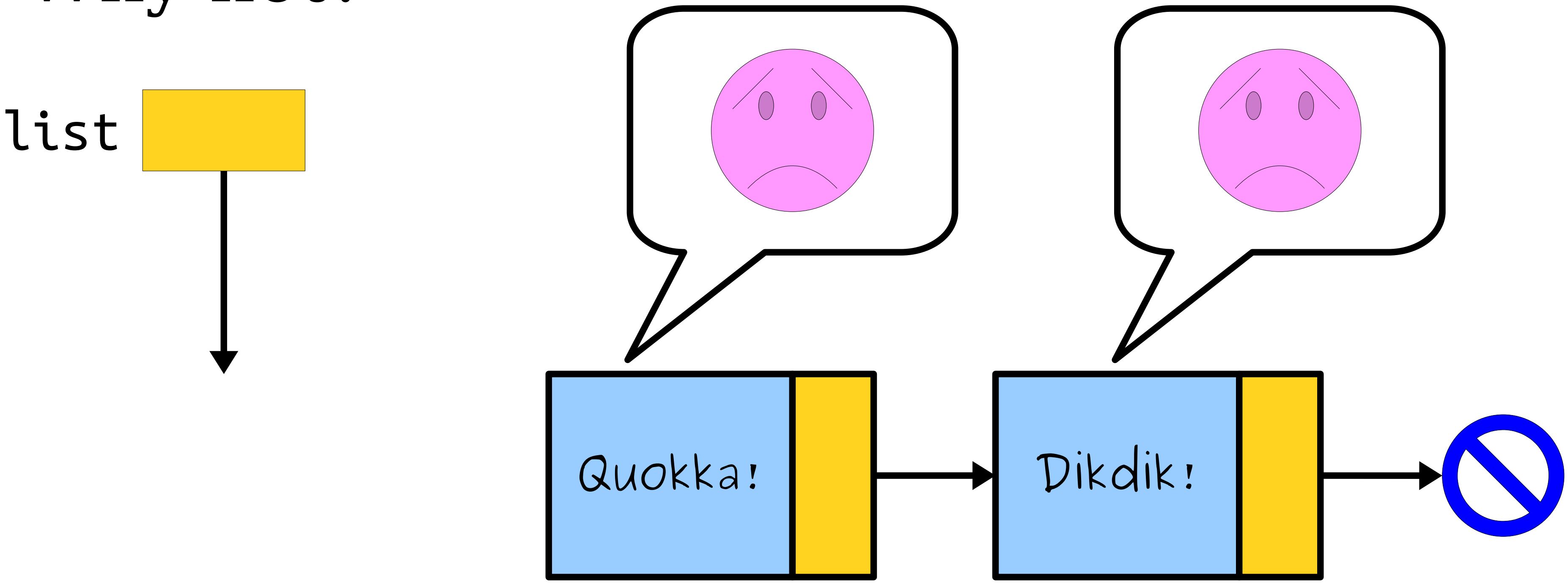
- ***Make sure to use the proper deletion operation***. Mixing these up is like walking off the end of an array or using an uninitialized pointer; it *might* work, or it might instantly crash your program, etc.

Cleaning Up Memory

- To free a linked list, we can't just do this:

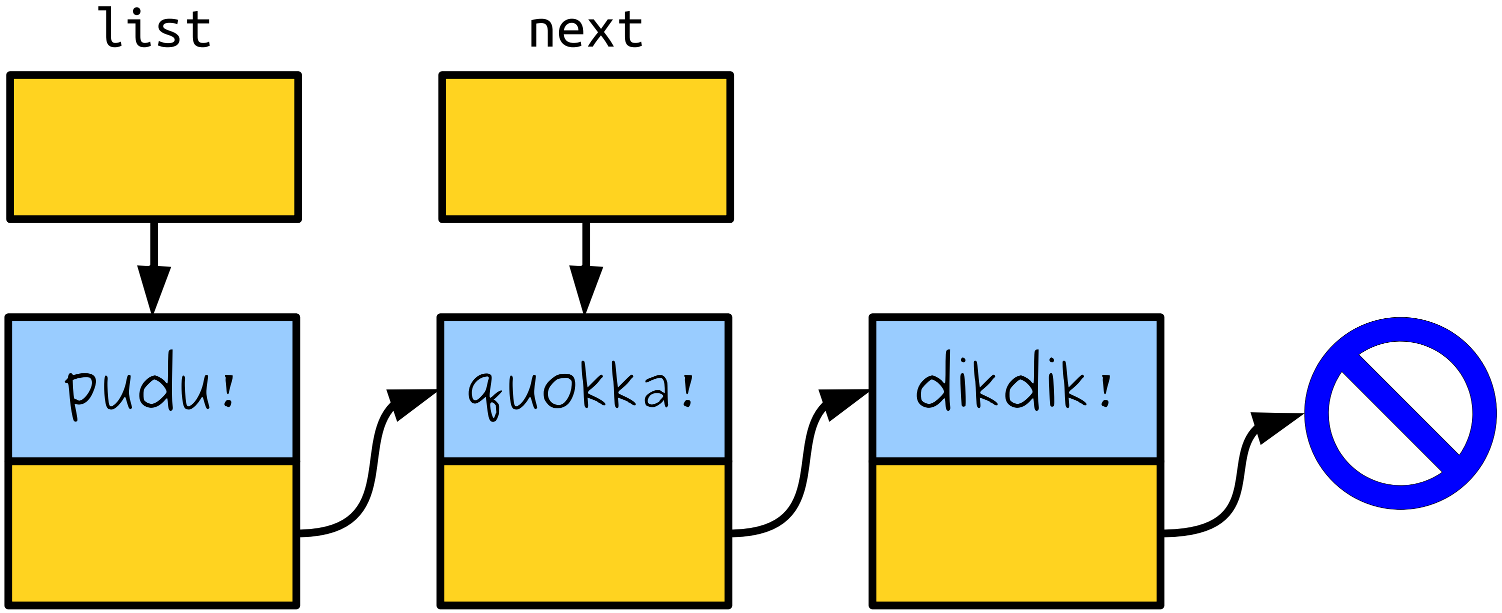
```
delete list;
```

- Why not?



Pointers Into Lists

- When processing linked lists iteratively, it's common to introduce pointers that point to cells in multiple spots in the list.
- This is particularly useful if we're destroying or rewiring existing lists.



Your Action Items

- ***Read Chapter 12.1 – 12.3.***
  - There’s lots of useful information in there about how to work with linked lists.
- ***Keep Working on Assignment 6***
  - If you’re following our suggested timetable, you’ll have finished the Enumerations Warmup and Linear Probing Warmup by now. Aim to complete Implementing Linear Probing by Wednesday if you can.
  - As always, come talk to us if you have any questions!

Next Time

- ***Pointers by Reference***
  - Getting a helping hand.
- ***Tail Pointers***
  - Harnessing multiple pointers into a list.