COMP1511 - Programming Fundamentals

Term 2, 2019 - Lecture 16

What did we learn yesterday?

Linked Lists

- Some Revision
- Insertion into Lists
- Finding certain things in lists using loops

What are covering today?

Battle Royale Game

- Linked List Insertion is complete
- We'll continue Linked List Removal
- Seeing the game being played
- Freeing memory

Assignment 2, Pokedex

- Specification overview
- Information about assessment

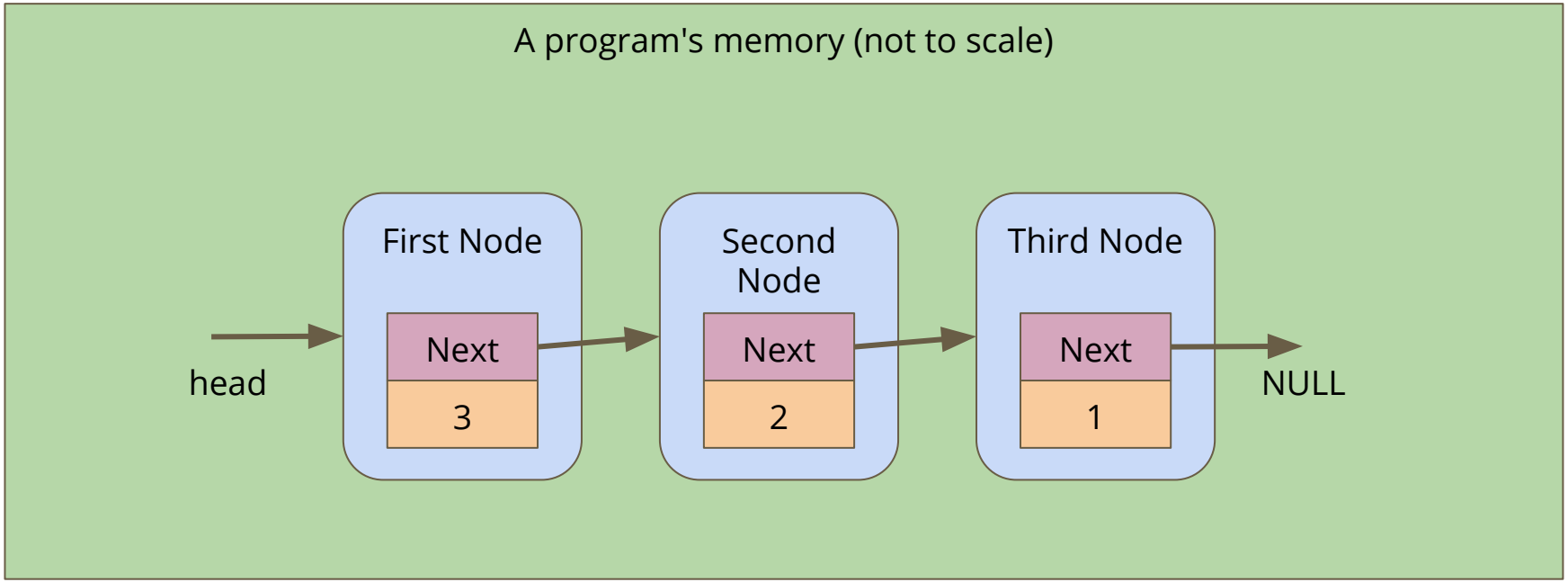
Removing a node

If we want to remove a specific node

- We need to look through the list and see if a node matches the one we want to remove
- To remove, we'll use **next** pointers to connect the list around the node
- Then, we'll free the node itself that we don't need anymore

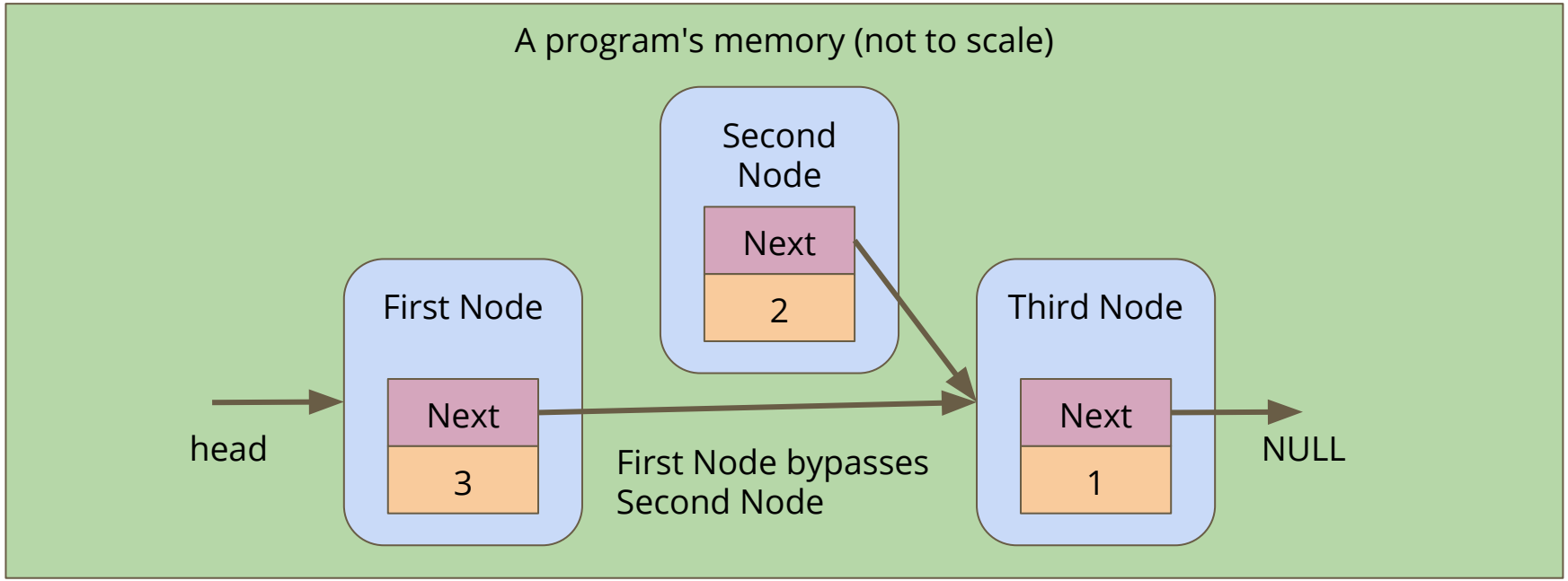
Removing a node

If we want to remove the Second Node



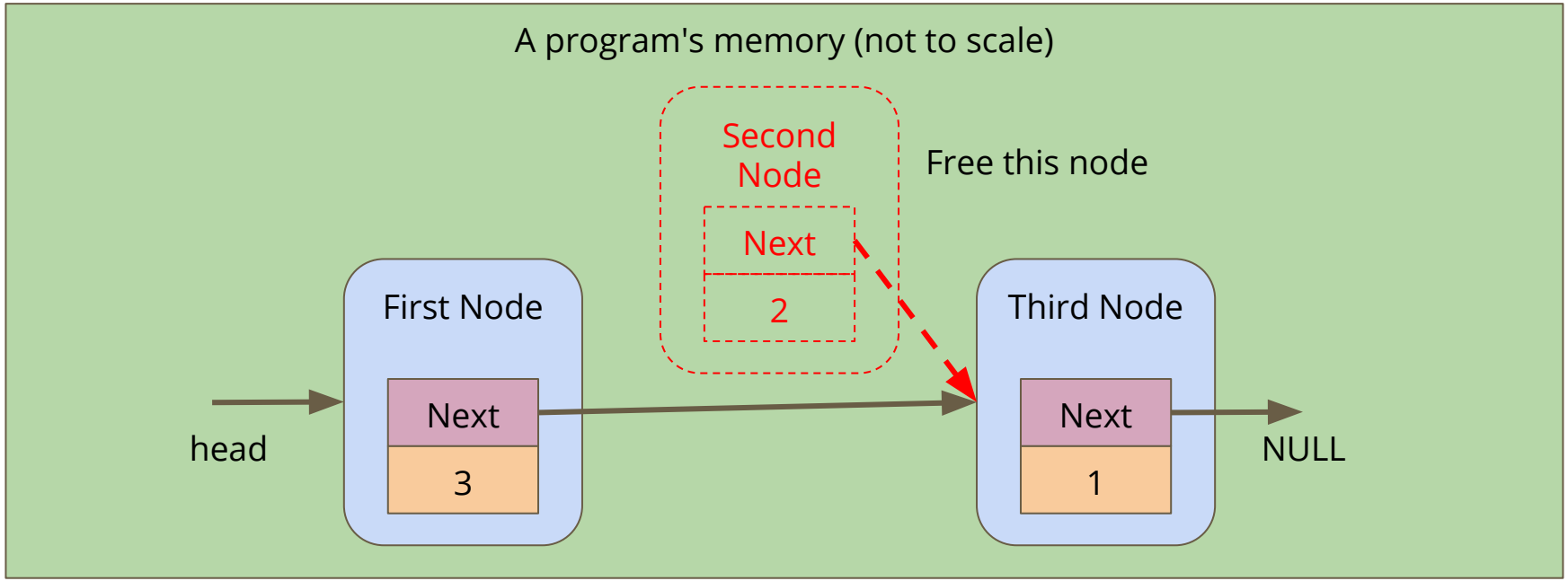
Skipping the node

Alter the First Node's **next** to bypass the node we're removing



Freeing the node

Free the memory from the now bypassed node



Finding the node

Loop until you find the right match

```
struct node *removeNode(char name[], struct node* head) {
    struct node *previous = NULL;
    struct node *n = head;
    // Keep looping until we find the matching name
    while (n != NULL && strcmp(name, n->name) != 0) {
        previous = n;
        n = n->next;
    }
    if (n != NULL) {
        // if n isn't NULL, we found the right node
```

Removing the node

Having found the node, remove it from the list

```
if (n != NULL) {
        // if n isn't NULL, we found the right node
        if (previous == NULL) {
            // it's the first node
            head = n->next;
        } else {
            previous->next = n->next;
        }
        free(n);
    }
    return head;
}
```

The Battle Royale Game

In a Battle Royale, people are removed from the game one at a time until only one person is left. They are the winner

- We can create a list of players
- We can make sure it's in a nice alphabetical order
- We can remove a single player from the list
- Now we need to remove players one at a time
- When there's only one left, they are the winner!

Game code

Once our list is created, we can loop through the game

- We print out the player list (we might want to modify that function!)
- Our user will tell us who was knocked out

```
// A game loop that runs until only one player is left
while (printPlayers(head) > 1) {
    printf("Who just got knocked out?\n");
    char koName[MAX_NAME_LENGTH];
    fgets(koName, MAX_NAME_LENGTH, stdin);
    koName[strlen(koName) - 1] = '\0';
    head = removeNode(koName, head);
    printf("----------\n");
}
printf("The winner is: %s\n", head->name);
```

Cleaning Up

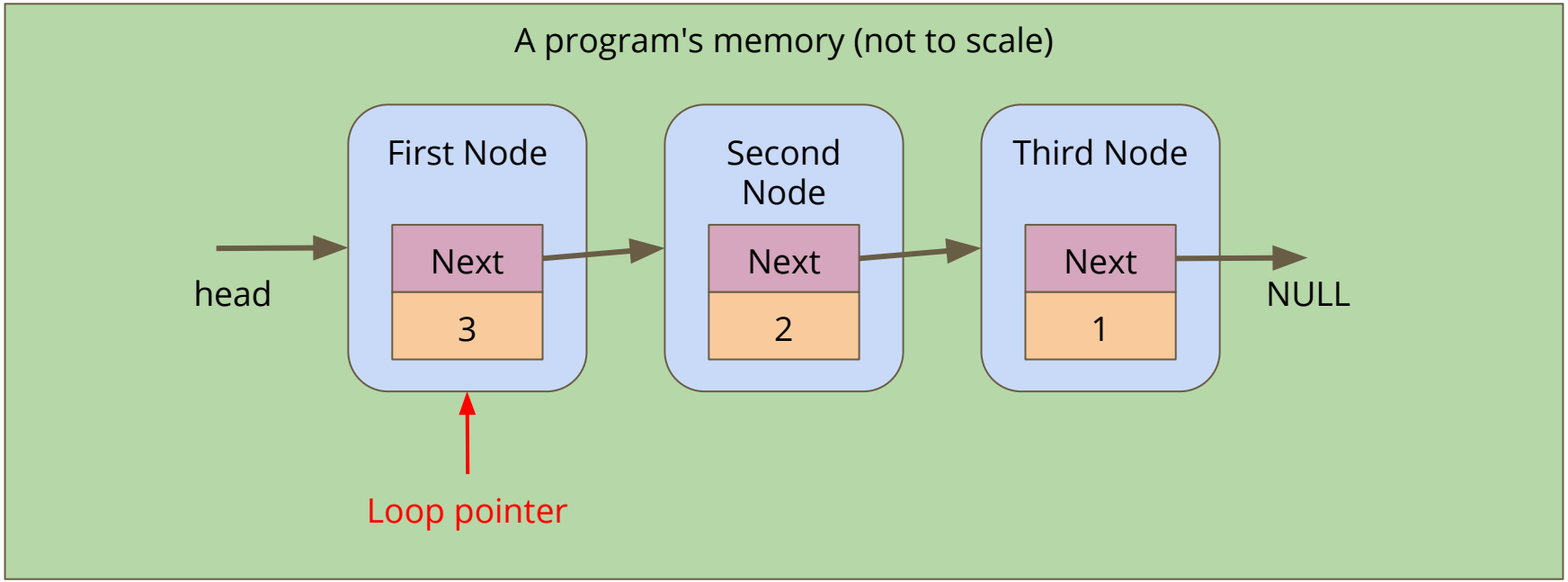
Remember, All memory allocated (malloc) needs to be freed

- We can run `dcc --leak-check` to see whether there's leaking memory
- What do we find?
- There are pieces of memory we've allocated that we're not freeing!

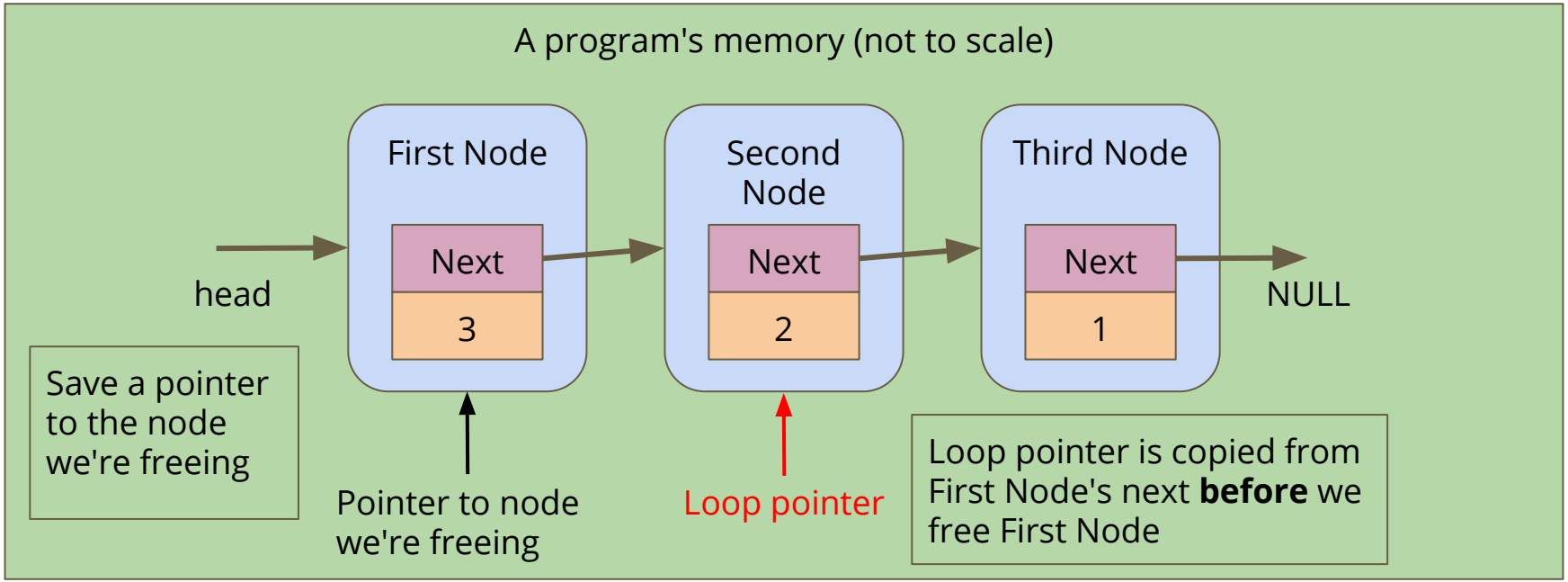
Let's write a function that frees a whole linked list

- Loop through the list, freeing the nodes
- Just be careful not to free one that we still need the pointer from!

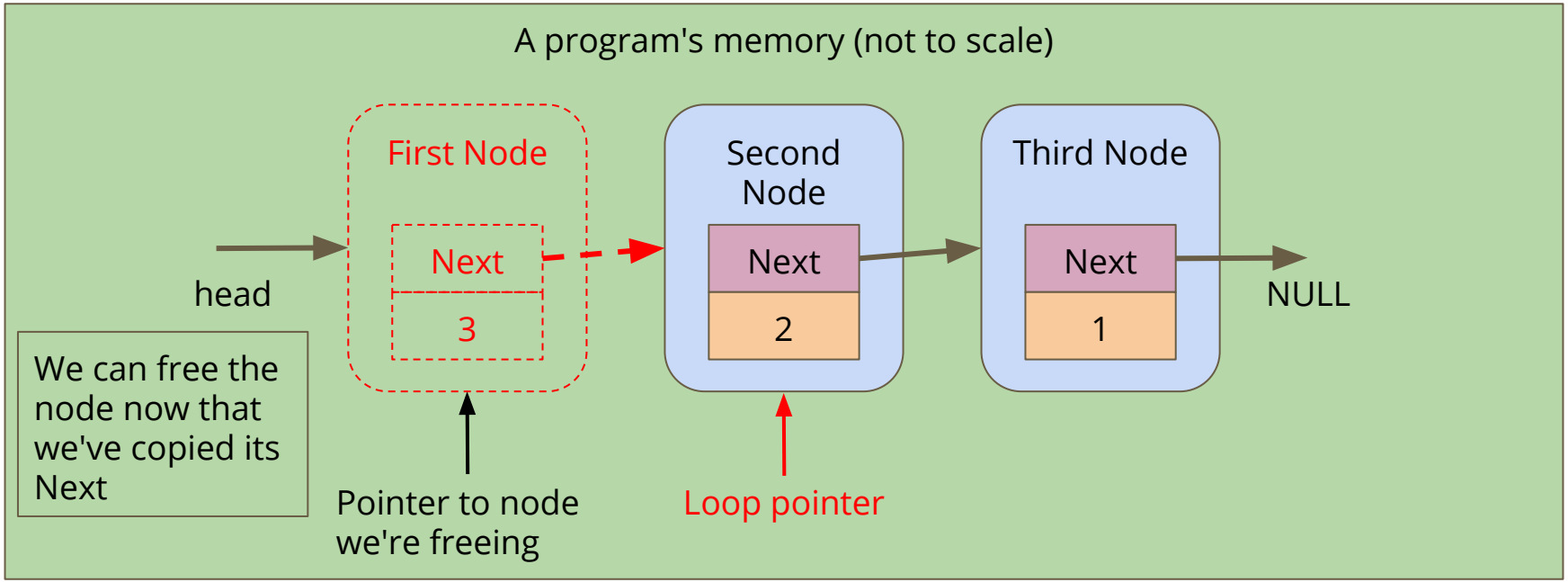
Looping to free nodes



Looping to free nodes



Looping to free nodes



Code to free a linked list

```
// Loop through a list and free all the allocated memory
void freeList(struct node *n) {
    while(n != NULL) {
        // keep track of the current node
        struct node *remNode = head;

        // move the looping pointer to the next node
        n = n->next;

        // free the current node
        free(remNode);
    }
}
```

Battle Royale, the Linked Lists demo

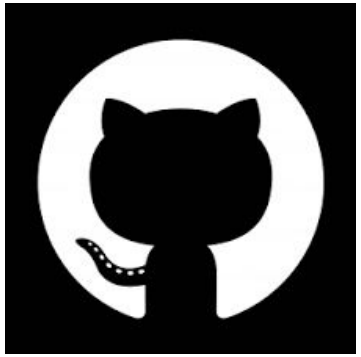
What have we written in this program?

- Creation of nodes
- Looping through a list
- Insertion of nodes into specific locations
- Finding locations using loops
- Removal of nodes
- Managing memory

Break Time

Keeping track of your own code projects

- Using **git** is a really handy way to keep backups of your work
- GitHub and BitBucket are two providers that will give you free online repositories to store your code
- Graphical Interfaces are available for git (GitHub Desktop and Sourcetree respectively)
- It takes some time to get familiar with how these work . . . but you can start practicing now!



Assignment 2 - The Pokémon Universe

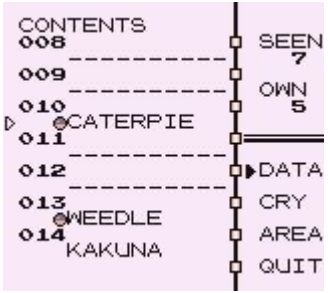
What is a Pokémon?

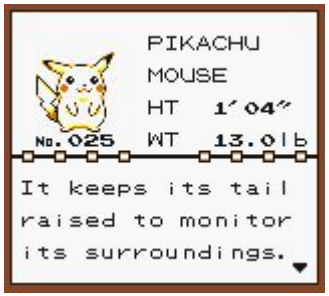
- A Pokémon is a fictitious creature made by Nintendo
- Pokémon are part of a massive gaming franchise based on their capture, training and use in battle
- CSE and UNSW does not condone the capture of wild animals for use in combat (and Pokémon need to have a good look at their ethics :P)

Assignment 2 - The Pokédex

What is a Pokédex?

- A Pokédex is an Encyclopedia that catalogues Pokémon
- One of the aims of the Pokémon games is to "Catch 'em all"
- When you encounter Pokémon in the game, you will be able to update your Pokédex with information about it



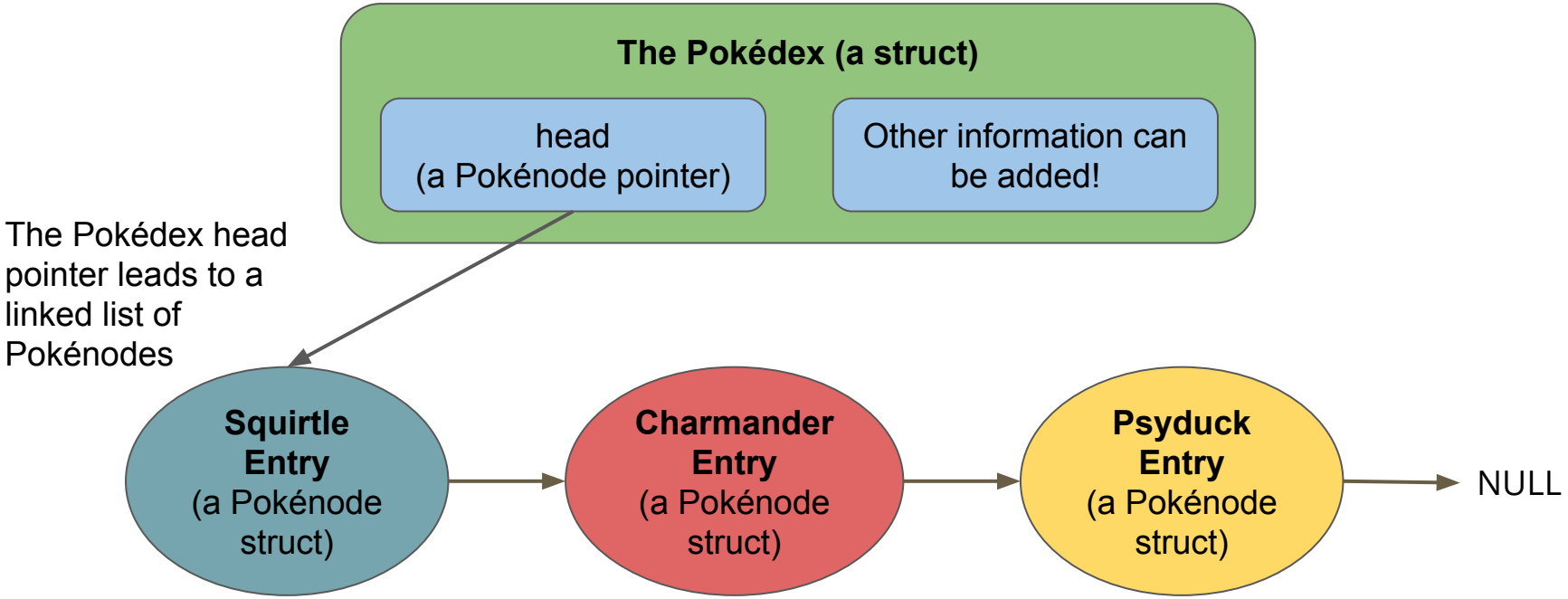


How does our Pokédex work?

If you ever have this question, you can use the reference solution

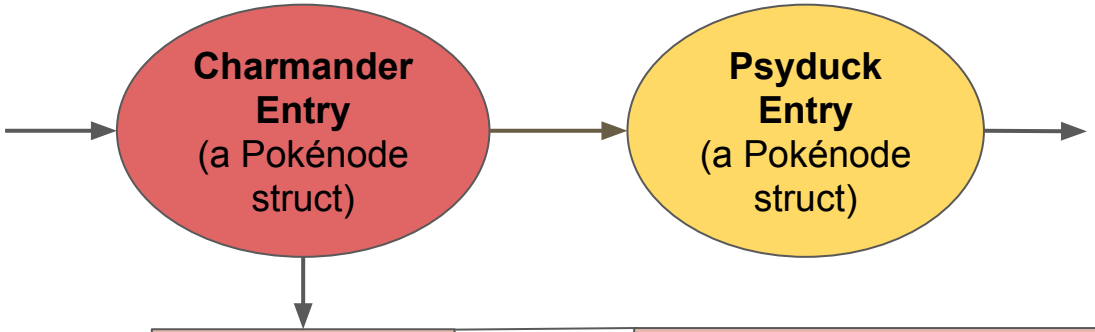
- "1511 pokedex_reference"
- Remember the ? command will always show the list of commands
- We can add and remove Pokémon
- We can mark Pokémon as found
- We can list our entire Pokédex
- and much more . . .

The Pokédex

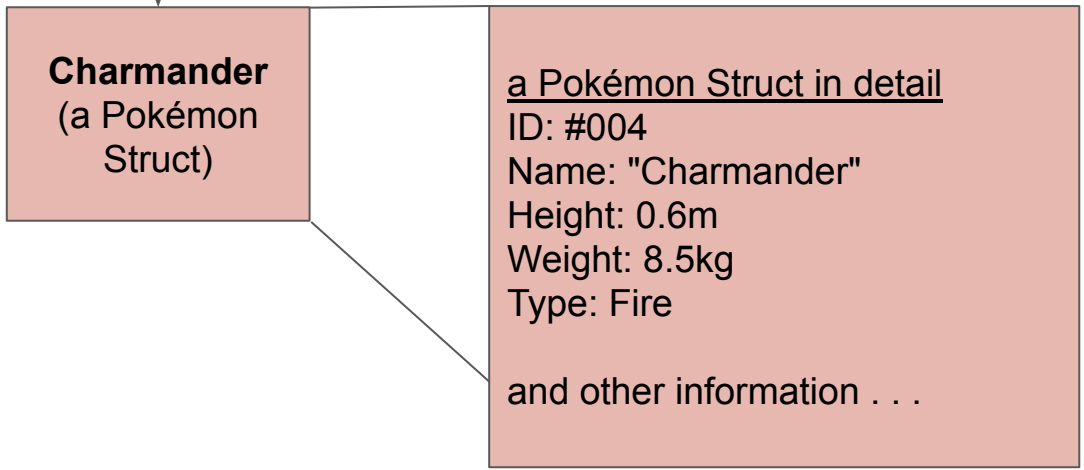

The Pokédex (a struct)
head
(a Pokénode pointer)
Other information can be added!
The Pokédex head pointer leads to a linked list of Pokénodes
Squirtle Entry
(a Pokénode struct)
Charmander Entry
(a Pokénode struct)
Psyduck Entry
(a Pokénode struct)
NULL

Pokénodes

Part of the linked list of Pokénodes



Each Pokénode has a pointer to a Pokémon struct that describes the Pokémon



How to get started

Setting up the Assignment

- You'll need to get access to the Assignment files
- First, create a directory for the Assignment (on VLAB or a CSE computer)
- Then, in that directory, run the copy and link commands from the Assignment page
- You will receive `pokedex.h, pokedex.c, pokemon.h, pokemon.c, main.c, test_pokedex.c`
- Note that most of the files will be links and not actual files
- This is because these files cannot be edited for the assignment!
- It also means if we need to update them, you'll automatically receive the newest versions

The Pokédex Project

You will be creating an implementation of a Pokédex

- We have provided a framework of files to work in
- You will only be able to edit `pokedex.c` and `test_pokedex.c`

- A Multi-File Project
  - Header Files contain declarations
  - C Files contain definitions

What's in the files?

pokemon.c and pokemon.h

- Code for Pokémon themselves
- A struct for a Pokémon that you **can't access**
- A series of functions that allow you to create and manipulate Pokémon
- You can safely ignore pokemon.c and just use what's written in pokemon.h
- pokemon.h describes all the functions and what you can do with them
- It's reasonably similar to using functions from <stdio.h> or other libraries

What's in the files?

pokedex.c and pokedex.h

- Similar to pokemon.c and pokemon.h
- Except pokedex.c is **incomplete!**
- Use pokedex.h to read the descriptions of what you must implement in pokedex.c

What's in the files?

main.c vs test_pokedex.c

- main.c is the interactive program that allows you to manually use the Pokédex
- test_pokedex.c (also contains a main function) allows you to automate testing
- You will be marked on the usefulness of the tests you have written in test_pokedex.c

Editing and Compiling

You will be implementing The Pokédex

- You'll be working mainly in `pokedex.c`, part of a multi-file project
- You'll also be creating tests in `test_pokedex.c`
- There are two ways to compile:
- For the interactive program:
  - **`dcc -o pokedex main.c pokedex.c pokemon.c`**
- For the automated testing:
  - **`dcc -o test_pokedex test_pokedex.c pokedex.c pokemon.c`**

Work in Stages

We have provided a staged process for your assignment work

- Progress through the stages in the order provided
- They're designed to work that way
- You will need knowledge of Linked Lists to create your Pokédex

Focus your attention

- Think about only one stage at a time
- Add features one at a time to save from confusing yourself
- Write your tests in `test_pokedex.c` before moving on

Working with your Linked List

The Linked List starts partially implemented

- The pokenode struct is already set up as a linked list node struct
- You will be expected to make some functions that use and modify the linked list
- As you progress, you will find you need to make the struct more complicated
- Add complexity only by necessity!

Testing

test_pokedex.c has some tests in it already

- You can run this to test some of the early stages functionality
- test_pokedex.c is not complete!
- However it does show you a nice way of setting up automated testing of individual functions
- This is often called **"Unit Testing"**

Assert

A valuable tool in testing

```
#include <assert.h>
// Asserts will test a code expression
int main (void) {
    int number = 2;
    int result = number * 3;

    // if this assert is false, the program will end here
    assert(result == 6);
}
```

We can use asserts to force our code to exit if one of our assumptions turns out to be false. If our program is running correctly, our asserts have no effect

Marks breakdown and Submission

More emphasis on Testing than Assignment 1

- 70% Performance Marks
- 10% Testing (mark will be given based on `test_pokedex.c`)
- 20% Code Style and Readability

Marked Files

- Only `pokedex.c` and `test_pokedex.c` can be submitted
- No other files will be accepted or marked
- Every submission via **`give`** is saved . . . use it as often as saving your files
- Marking will be done on CSE computers and all compilation with dcc

Marking - Pass

Pass Mark - Readable Code that implements Stage 1

- Add nodes to a linked list
- Extract information from individual nodes, using `pokemon.h` functions, including whether it has been found
- Be able to traverse a linked list, extracting information as you go
- Added some tests to `test_pokedex.c`

Marking - Credit

Credit - Stage 2

- Stages 1 and 2 working
- Tracking a selected pokemon
- Cleaning up data and memory
- Testing in test_pokedex.c that proves that functionality works
- Reasonable effort put into style and readability

Marking - Distinction

Distinction - Stage 3 and more

- Completion of stages 1-3 plus some of 4-5
- Looping through a list and finding information in a more complex way
- Editing very specific information in a list
- Very readable and re-usable code
- Comprehensive testing that shows the ability to deal with different situations

Marking - High Distinction

High Distinction - Stage 4

- Implement stages 1 - 4
- Evolutions that link Pokémon to each other
- Testing ranges of possibilities including issues that might not come up often, but could cause functionality issues
- Clean, clear, understandable code that is fully explained where necessary

Marking - Full Marks

Full Marks - All Stages

- Full implementation of functionality in the assignment
- Creation of sub-lists, new lists that include copies of some of the nodes from your main Pokédex list
- A comprehensive test file that tests all functionalities and uncommon "edge cases" in possible use of functions
- Elegant code that could be used to teach people about Linked Lists and other structures used because it's so easy to understand

What did we cover today?

Linked Lists

- Removal
- Memory cleaning

Assignment 2 - Pokédex

- Theme
- Structure
- Testing
- How to approach the assignment
- Marking Scheme