Welcome!

COMP1511 18s1
Programming Fundamentals
Overview

after this lecture, you should be able to...

have a better understanding of linked lists

write code to create a linked list

write code to traverse a linked list

solve simple problems using linked lists

(note: you shouldn’t be able to do all of these immediately after watching this lecture. however, this lecture should (hopefully!) give you the foundations you need to develop these skills. remember: programming is like learning any other language, it takes consistent and regular practice.)
Don’t panic!

assignment 2
(if you haven’t started yet, start ASAP)

deadline extended to Sunday 13th May

assignment 1

tutor marking/feedback in progress

week 9 weekly test out now

don’t forget about help sessions!

see course website for details
Help Sessions

Wednesday
6-8pm, J17 201

Thursday
6-8pm, J17 201

Friday
10am-12pm, Brass Lab (J17 Level 3)
2pm-4pm, Brass Lab (J17 Level 3)
4pm-6pm, Oboe Lab (J17 Level 3)

note: Brass Lab = Bugle/Horn
The node struct

```c
struct node {
    int data;
    struct node *next;
};
```
Interacting with a node struct

```c
struct node {
    int data;
    struct node *next;
};

// "struct node hello" (no *)
// "hello" is an actual node in the function's memory
struct node hello;
hello.data = 10;
hello.next = NULL;

// in the function's memory
// ______
// hello | 10 |
//       |-----|
//       | NULL |
//       |______|
```
Making a new node

// Allocates memory for a new node; returns its address
struct node *make_node(int value) {
    struct node *new = malloc(1 * sizeof(struct node));
    new->data = value;
    new->next = NULL;

    return new;
}

// "struct node * hello"
// "hello" is a pointer to a node,
// it just stores the _address_
// (of the memory we get from malloc)
struct node *hello = make_node(10);

// in the heap (malloced memory)
//
// hello |  10  |
//       |------|
//       |   NULL   |
//       |______|
Freeing a node

// In accordance with Newton's 3rd Law of Memory Allocation
// "For every malloc, there is an equal and opposite free"

void free_node(struct node *node) {
    free(node);
}

struct node *hello = make_node(10);
free_node(hello);
Node pointers vs allocated nodes

**Reference** to a node

```c
struct node *curr ...
```

vs

making (**allocating**) a new node

```c
... = malloc(1 * sizeof(struct node));
```
Node pointers vs allocated nodes

reference to a node

\[ \text{struct node} \ast \text{curr} \ldots \]

vs

making (allocating) a new node

\[ \ldots = \text{malloc}(1 \ast \text{sizeof(struct node)}); \]
Node pointers vs allocated nodes

Reference to a node (arrow) vs making (allocating) a new node (circle)

```
struct node * curr
```

```
struct node
```

```
memory allocated by malloc
```

```
struct node * curr = malloc(1*sizeof(struct node))
```
array/list “traversal”

(going through every element)
Traversing... an Array

```c
void fillArray (int array[ARRAY_SIZE], int value) {
    int i = 0;
    while (i < ARRAY_SIZE) {
        array[i] = value;  // set the value
        i++;                // move to next element
    }
}
```
void fillList (struct node *list, int value) {
    struct node *curr = list;
    while (curr != NULL) {
        curr->data = value; // set the value
        curr = curr->next;   // move to next node
    }
}