Welcome!

COMP1511 18s1

Programming Fundamentals

COMP1511 18s1

— Lecture 18 —

Fruit Bot + More Linked Lists

Andrew Bennett

<andrew.bennett@unsw.edu.au>

Overview

after this lecture, you should be able to…

start on the last assignment: **Fruit Bot**

work with **multi-file** C programs

understand the purpose of **.h** files

have a better understanding of **linked lists**

write code to **free** 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.)

Admin

Don’t panic!

assignment 2 due yesterday

(if you haven’t started yet…. ☹)

assignment 3 out now!

this week’s tute/lab help you get started

week 10 weekly test due **thursday**

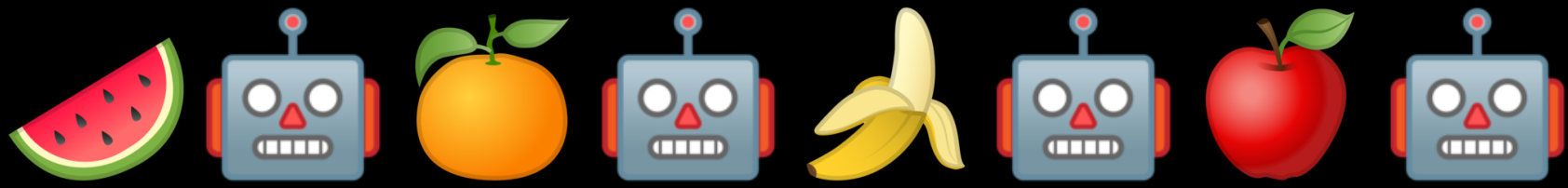
don’t forget about **help sessions**!

see course website for details

introducing: Fruit Bot

(assignment 3)

...



Assignment Spec

https://cgi.cse.unsw.edu.au/~cs1511/18s1/assignments/ass3/index.html

Fruit Bot

note: **do not change the structs**

stateless bot

(similar to *Intensity*)

world is **randomly generated**

(you can't assume any particular fruit is/isn't there)

An Aside: Multiple C Files

until now we've only had **one** .c file per program

but, we can make programs with **multiple** .c files

Scope

to call a function you need to know about its

return type

input parameters

(we call this the **API**)

Using Functions From Another File

first.c

```c
void hello(void);
int square(int n);

int main(void) {
    hello();
    printf("%d", square(5));
}

void hello(void) {
    printf("Hello!\n");
}

int square(int n) {
    return n*n;
}
```

```
$ dcc -o first first.c
$ ./first
Hello!
25
```

Using Functions From Another File

second.c

```
int main(void) {
    hello();
    printf("%d", square(5));
}
```

```
$ dcc -o second second.c
????
$ ./second
????
```

Using Functions From Another File

first.h

```c
#ifndef FIRST_H
#define FIRST_H

void hello(void);
int square(int n);

#endif
```

Using Functions From Another File

second.c

```c
#include "first.h"

int main(void) {
    hello();
    printf("%d", square(5));
}
```

```
$ dcc -o second second.c first.c
$ ./second
Hello
25
```

and now for some more
Linked Lists

<REVIEW>

The node struct


```
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


```c
// 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

arrow

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

vs

making (**allocating**) a new node

circle

```
... = malloc(1 * sizeof(struct node));
```

Node pointers vs allocated nodes

reference to a node

arrow

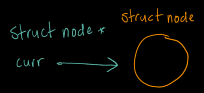
```
struct node *curr ...
```

vs

making (**allocating**) a new node

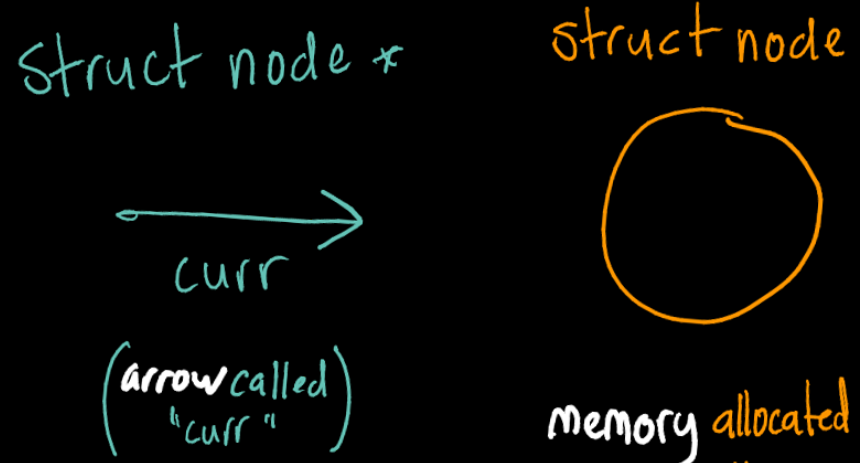
circle

```
... = malloc(1 * sizeof(struct node));
```



Node pointers vs allocated nodes

reference to a node (arrow) vs
making (**allocating**) a new node (circle)




```
struct node * curr = malloc(1 x sizeof(struct node))
```

array/list “traversal”

(going through every element)

Traversing… an Array


```
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
    }
}
```

Traversing… a Linked List


```
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
    }
}
```

The Standard List Loop


```
struct node *curr = list;

while (curr != NULL) {

    ?????

    curr = curr->next;
}
```

The Standard List Loop – List Length

How can we calculate the length of a list?

i.e. how many nodes are in the list

```
struct node *curr = list;

int num_nodes = 0;

while (curr != NULL) {

    num_nodes += 1;

    curr = curr->next;
}
```

The Standard List Loop – List Sum

How can we sum all of the elements in a list?

i.e. add the values of all of the nodes together

```
struct node *curr = list;

// int num_nodes = 0;
?????

while (curr != NULL) {

    // num_nodes += 1;
    ?????

    curr = curr->next;
}
```

</REVIEW>

More List Iteration

does the list contain a certain **value**?

Freeing a List

“For every malloc, there is an equal and opposite free.”

we need to free our entire list when we’re done with it