COMP1511 18s1 — Lecture 12

The Data Went Data Way

~~Andrew Bennett~~
~~<andrew.bennett@unsw.edu.au>~~

Jashank Jeremy
<jashank.jeremy@unsw.edu.au>

Overview

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

- work with composite data types,
- reason about the scope and lifetime of a value,
- use dynamic memory management functions

(**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** ...
  - spec released by tomorrow
  - discussing it in the lecture
- **Weekly test #4** ... due Wednesday 23:59:59 AEST
- **week 6 challenges** ... extended to Friday 23:59:59 AEST
- **week 8** is **quiet week**!
  - no lectures! no tutorials! no labs!
  - ... help sessions still running

Manipulating Data

Computer science is about manipulating data.
(Especially complex data!)

All programming languages have
mechanisms for dealing with *composite* data:
grouping together related information
into a single logical unit

... Arrays?

Arrays are good, but sometimes they're not:
unknown size, uniform type, ...

... Arrays (cont'd)

```c
#define N_STUDENTS 1087
#define MAX_NAME_LEN 64

int studentID[N_STUDENTS];
char name[N_STUDENTS][MAX_NAME_LEN];
int tutorial[N_STUDENTS];
int ass1_mark[N_STUDENTS];
```

- ... what if student 39 drops?
- ... what if 39 students drop?
- ... what if we want more fields?
- ... what if things drift out of step?

What do we want?

- grouping related data
- data of differing types
- accessing each datum

`struct`

a way to group together
related data of **differing types**
we refer to the individual pieces of data
as **fields** or **members**

```
typedef struct _type-name {
    type member;
    [...]
} type-name;
```

Lifetimes and Scope

Review: Stack Frames

On the stack:

previous frame, return address

parameters, return values

local variables

... values relevant for a function's invocation

Boundedness

Values on the stack will only live
as long as the stack frame does.

we can say a variable has a lifetime,
bounded by the stack frame.

… Boundedness

```c
#define ARRAY_SIZE 10

int *makeArray (int initialValue);
void printArray (int array[ARRAY_SIZE]);

int main (void) {
    int *xs = makeArray (17);
    printArray (xs);
    return 0;
}

int *makeArray (int initialValue) {
    int array[ARRAY_SIZE];
    int i = 0;
    while (i < ARRAY_SIZE) {
        array[i] = initialValue;
        i++;
    }
    printArray (array);
    return array;
}
```

... Boundedness

The value in `array` won't live long enough!

Pass a Lower Reference

Take a reference *lower* on the stack,
pass it up the stack to called functions

Escape Hatches

Global variables! `static` variables!

however:
Style Guide
... §Global and Static Variables

Summarily: **evil**.
don't use global variables,
don't use `static` variables.