COMP1511 17s2

— Lecture 10 —

Structure

~~Andrew Bennett~~
~~<andrew.bennett@unsw.edu.au>~~
Jashank Jeremy
<jashank.jeremy@unsw.edu.au>

review: pointers
structured data

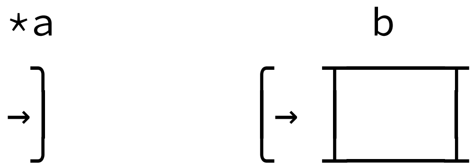
Don’t panic!

assignment 0 postmortem:
so much plagiarism aaaaaa

milestone 2
due Wed 30 Aug, 23:59

census date:
31 August

Review: Pointers!



pointers are like worm-holes…
they refer to an object somewhere else
by memory address

In C, we have two pointer operations:
& and *

arrays don’t have well-defined size,
so when passed to functions,
pass as references: as pointers!

Review: Arrays!

3	0	0	0	0	17	0	0	0	0
0	1	2	3	4	5	6	7	8	9

a collection of array elements
each element must be the same type

we refer to arrays by their index
valid indices for n elements are $0 \ldots n - 1$

we cannot assign, scan, or print whole arrays…
but we can assign, scan, and print elements

Review: Character Arrays!

```
{ 'S', 'e', 'v', 'e', 'n', 't', 'e', 'e', 'n', '\0' }
"Seventeen"
```

text between double quotes is a string;
an array of characters,
ending with a null terminator

nominally, `char []`,
but often we just talk about `char *`

how long is a piece of string?

Why Arrays?

When we introduced arrays, we talked about statistics…

```
int mark_student0, mark_student1, mark_student2, ...;
mark_student0 = 73;
mark_student1 = 42;
mark_student2 = 99;
...
```

… and how, with arrays, this got better:

```
int mark[550];
mark[0] = 73;
mark[1] = 42;
mark[2] = 99;
...
```

What if...?

What if I wanted to build a student database?
student ID, tutorial, marks
… a whole pile of disparate arrays!

```
#define N_STUDENTS 550
#define MAX_NAME_LENGTH 64

int studentID[N_STUDENTS];
char name[N_STUDENTS][MAX_NAME_LENGTH];
int tutorial[N_STUDENTS];
int week01mark[N_STUDENTS];
int assign0mark[N_STUDENTS];
...
```

What if…?

What if I wanted to build a student database?
student ID, tutorial, marks
… a whole pile of disparate arrays!

aaaaaAAAAAAAAARGH!
We're not approaching the problem right.

We need a way to store
related data of differing types
together… and that's not arrays.

struct and typedef

keeping it together

What if...?

What if I wanted to build a student database?
student ID, tutorial, marks
... let's create a struct.

```
#define MAX_NAME_LENGTH 64

typedef struct {
    int studentID;
    char name[MAX_NAME_LENGTH];
    int tutorial;
    int week01mark;
    int assign0mark;
} student;
```

Aside: `typedef` syntax

`typedef` *`existing-type new-type-name`*`;`

`typedef` lets us create our own types,
that can be shorter than types we already have

```
typedef unsigned char byte;
typedef char *string;
```

`struct` syntax

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

In this course, we forbid tagged `struct`s, and we prefer `typedef`s.

The Complex Mathematics Cheat-Sheet

(wherein it's been too long since I did MATH1131)

for all $a, b \in \mathbb{R}$
$w \in \mathbb{C}$ gives $w = a + bi$
$\Re(w) = a$ and $\Im(w) = b$

we have a symbol i floating around...
it's equal to $\sqrt{-1}$, but we don't *really* care

for $w, z \in \mathbb{C}$:

$$w + z = (a + bi) + (c + di) = (a + c) + (b + d)i$$
$$w - z = (a + bi) - (c + di) = (a - c) + (b - d)i$$
$$wz = (a + bi)(c + di) = (ac - bd) + (bc - ad)i$$

oh, also: $w = a + bi$ is equivalent to
$w = r\mathbf{e}^{i\theta}$ and $w = r(\cos\theta + i\sin\theta)$, where
$r = |w| = \sqrt{a^2 + b^2}$, and
$\theta = \arg(w) = \tan^{-1}(y/x)$