COMP1511 17s2
— Lecture 8 —
Pointers!

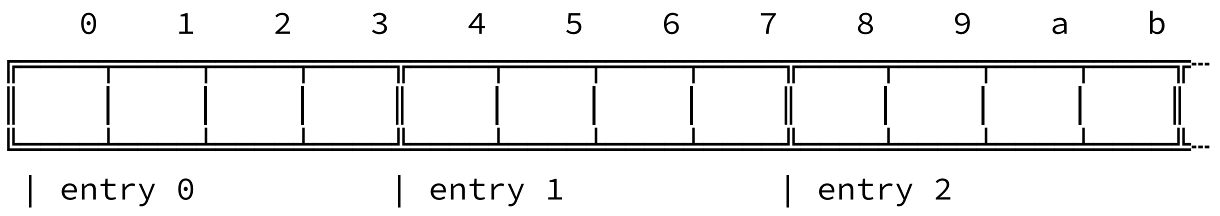
Andrew Bennett
`<andrew.bennett@unsw.edu.au>`

pointers
strings
arrays

Don’t panic!

Review: Arrays

a series of boxes in memory
with a common type,
all next to each other



Review: Arrays in C

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

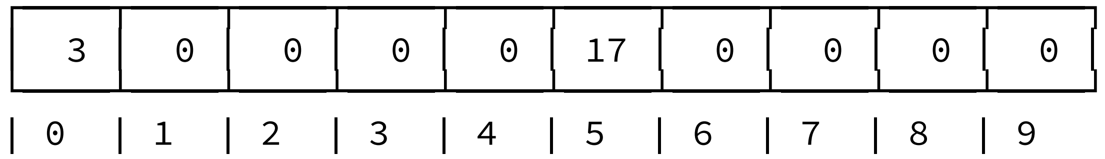
we refer to arrays by their index
valid indices for n elements are $0 \ldots n - 1$

no real limit on number of elements

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

Review: Arrays in C

```c
// Declare an array with 10 elements
// and initialises all elements to 0.
int myArray[10] = {0};
// Put some values into the array.
myArray[0] = 3;
myArray[5] = 17;
```



Review: Array Manipulation

`scanf()` can't read an entire array.
`printf()` can't print an entire array.
the `=` operator won't copy an array.

instead, you must perform an operation on each element:

```
int i = 0;
while (i < ARRAY_SIZE) {
    array2[i] = array1[i];
    i++;
}
```

Review: Array-ception

an array may have elements of any type…
including of array type!
we call these multi-dimensional arrays

here's an array of arrays of `int`:

```
int matrix[3][3] = {
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9}
};
```

it's a two-dimensional array.

```
printf ("%d\n", matrix[1][1]); // outputs... ?
```

char-grilled

working with characters and text

long long, long, int, short, char

in *Explore Memory*
we saw a range of new types
for storing numeric data:
`long long`, `long`, `int`, `short`, `char`

the difference between them?
the range of data they can store,
and the amount of space they need.

char

a char behaves just like a small number
but we also have a mapping from
values to displayed characters which we call
ASCII
(the American Standard for Computer Information Interchange)

ASCII codes are just
a way of specifying numbers.

in C,

```
int c = 'A';
```

is the same as

```
int c = 65;
```

Working with `char`

in `<stdio.h>`, there are two useful functions:

`getchar`
read a character from “standard input”

`putchar`
prints a character to “standard output”

usually,
“standard input” and “standard output”
get connected to the terminal you’re working in.

(Autotest is an example of a program that
connects standard input and standard output
to itself to test your programs.)

the `getchar` pattern

```c
int c = getchar ();
while (c != EOF) {
    // do something with `c`
    c = getchar ();
}
```

getchar and putchar

```
int c = getchar ();
while (c != EOF) {
    putchar (c);
    c = getchar ();
}
```

Pointers

Pointer slides continue on slide 14.

Things Stay The Same

consider this:

```
int ten = 0;
f (ten);
g (ten);
printf ("%d\n", ten);

void g (int x) {
    x = x + 6;
}

void f (int x) {
    x = x + 4;
}
```

what does this print?
why?

Refresher: Memory

(in case you forgot)

an array of consecutive boxes to store data in

different types are different sizes
`char` = 1 byte
`short` = 2 bytes
etc.

everything in memory has an address;
these addresses are just numbers.
it’s convenient to talk about addresses in hexadecimal

Exploring Memory

```c
int num1;
int num2;

printf("Address of num1 is: %p\n", &num1);
printf("Address of num2 is: %p\n", &num2);

printf("Size of num1 is: %lu\n", sizeof(num1));
printf("Size of num2 is: %lu\n", sizeof(num2));
```

```
Address of num1 is: 0xffffd86c
Address of num2 is: 0xffffd868

Size of num1 is: 4
Size of num2 is: 4
```

Exploring Memory

```
#define ARRAY_SIZE 5

int firstArray[ARRAY_SIZE];

printf ("Address of firstArray is: %p\n", &firstArray);
printf ("Looking at the addresses of firstArray:\n");

int i = 0;
while (i < ARRAY_SIZE) {
    printf ("The address of firstArray[%d] is %p\n",
        i, &firstArray[i]);
    i++;
}
```

```
Address of firstArray is: 0xffffd848

Looking at the addresses of firstArray:
The address of firstArray[0] is 0xffffd848
The address of firstArray[1] is 0xffffd84c
The address of firstArray[2] is 0xffffd850
The address of firstArray[3] is 0xffffd854
The address of firstArray[4] is 0xffffd858
```

“Address-Of”

if we declare a variable called `x`,
the place where `x` is stored
(a “pointer” to,
the “address” of `x`,
a “reference” to `x`)
is denoted

```
&x
```

“Dereference”

if we have
a pointer, an address, a reference
to a value `x`,
and want to manipulate the value it holds,
we dereference it

`*x`

essentially,
`*` cancels out `&`

Pass by Reference

```
int ten = 0;
f (&ten);
g (&ten);
printf ("%d\n", ten);

void g (int *x) {
    *x = *x + 6;
}

void f (int *x) {
    *x = *x + 4;
}
```

what does *this* print?

(We’ve already seen this – `scanf`!)

More Arrays

passing arrays as arguments to functions

```
void showArray (int arraySize, int array[]);
```

More Arrays

arrays are similar to pointers.

how long is an array?
we don't know. we must always specify size.

C doesn't let you pass arrays.
instead, you get a pointer

More Arrays

modifying arrays from functions

write a function `negativeArray`,
that takes an `int` array
and make all values negative