Welcome!

COMP1511 18s1

Programming Fundamentals

COMP1511 18s1

— Lecture 6 —

Loops + Arrays

Andrew Bennett

<andrew.bennett@unsw.edu.au>

loops inside loops

stopping loops

arrays

Before we begin…

introduce yourself to the person sitting next to you

why did they decide to study **computing**?

Feedback

upload **lecture code**

upload/incorporate **diagrams**
in lecture recordings

more **diagrams**

go through programs **as a whole**
before running them

lecture **subtitles**

Overview

after this lecture, you should be able to…

write programs using **nested while loops** to solve simple problems

understand how to **stop** loops
using **loop counters** and **sentinel variables**

understand the basics of **arrays**

understand the basics of **designing a solution** to a problem

(time permitting)

(**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!

lecture recordings are on WebCMS3

I’ll try to add drawings/diagrams + upload to YouTube

course **style guide** published

weekly test due wednesday

don’t be scared!

assignment 1 coming soon

more on that tomorrow!

don’t forget about **help sessions**!

see course website for details

Loops

what if we want to do something multiple times?

Use a loop!

*keep doing this **while** this condition is true*

Anatomy of a Loop

initialisation

set up our variables

condition

while “something”…

statements

things we do inside our loop

update

move along to the next iteration

Anatomy of a Loop

```c
// initialisation
int i = 0;

// condition
while (i < n) {
    // statements -- do something in the loop
    printf("Hello!\n");

    // update
    i++;
}
```

Stopping: Loop Counters

```c
// Print out "hello, world!" n times,
// where n is chosen by the user.

int num;
printf ("Enter a number: ");
scanf ("%d", &num);

int i = 0;
while (i < num) {
    printf ("hello, world!\n");
    i = i + 1;
}
```

Stopping: Sentinel Value (Flag)

```c
// Print out the number that the user entered
// Stop when they type 0

int n = 1;
while (n != 0) {
    printf ("You entered: %d\n", n);
    scanf("%d", &n);
}
```

Nested Loops

```
while (something) {
    while (somethingElse) {

    }
}
```

Nested Loops

```
int i = 0;
while (i < n) {
    int j = 0;
    while (j < n) {
        // do something
        printf("!!");
        j++;
    }
    i++;
}
```

revisiting: variables

Variables

variables are like a **box**
that can hold a **value**
of a certain **type**

```
int i = 5;
 ______
|      |
|   5  |
|______|
```

Variables

we can have as many variables as we like

```
int i = 5;
 ______
|      |
|   5  |
|______|

int age = 18;
 ______
|      |
|  18  |
|______|

double pi = 3.14;
 ______
|      |
| 3.14 |
|______|
```

Lots of Variables

sometimes we want to store a lot of related variables

```
int mark_student0 = 85;
 ______
|      |
|  85  |
|______|

int mark_student1 = 90;
 ______
|      |
|  90  |
|______|

int mark_student2 = 45;
 ______
|      |
|  45  |
|______|
```

Lots of Variables

sometimes we want to store a lot of related variables

```
int mark_student0 = 85;
int mark_student1 = 90;
int mark_student2 = 45;
```

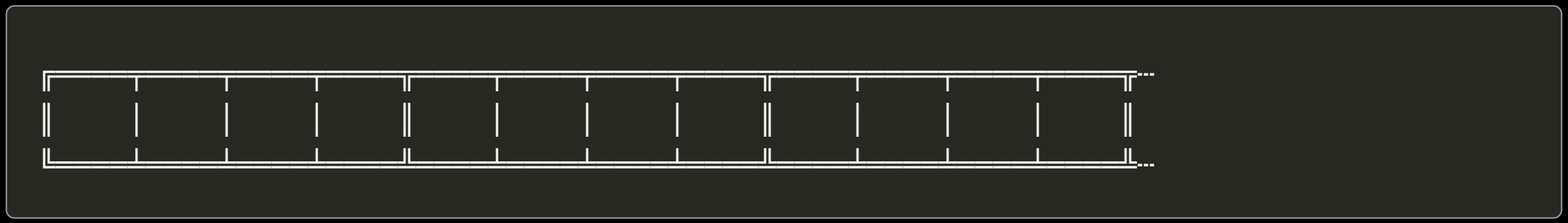
```
double averageMark = (mark_student1 + mark_student2 + mark_student3)/3;
printf("The average mark was: %lf\n", averageMark);
```

This doesn't scale!

introducing: **arrays**

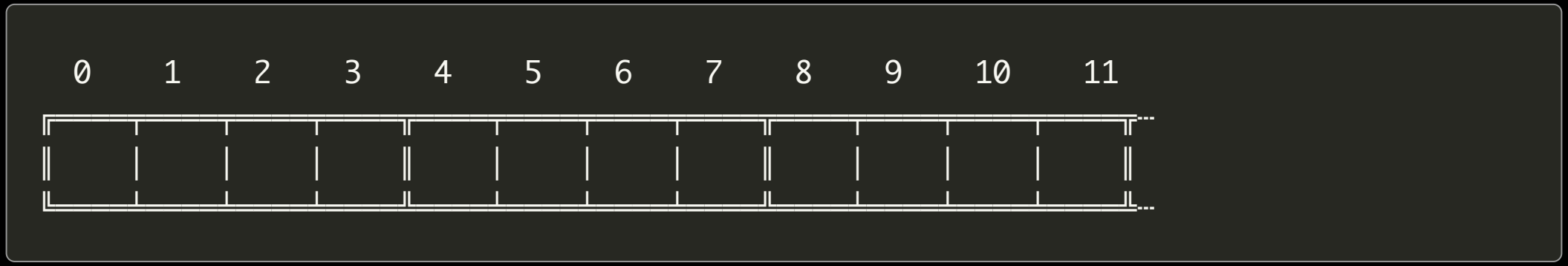
Arrays

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



Arrays

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



Why?

Suppose we need to compute statistics on class marks...

```
int mark_student0, mark_student1, mark_student2, ...;
mark_student0 = 85;
mark_student1 = 90;
mark_student2 = 45;
...
```

becomes unfeasible if dealing with a lot of values

... we'd need hundreds of individual variables!

Why?

Solution: Use an array!

```
int mark[1160];
mark[0] = 85;
mark[1] = 90;
mark[2] = 45;
...
```

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

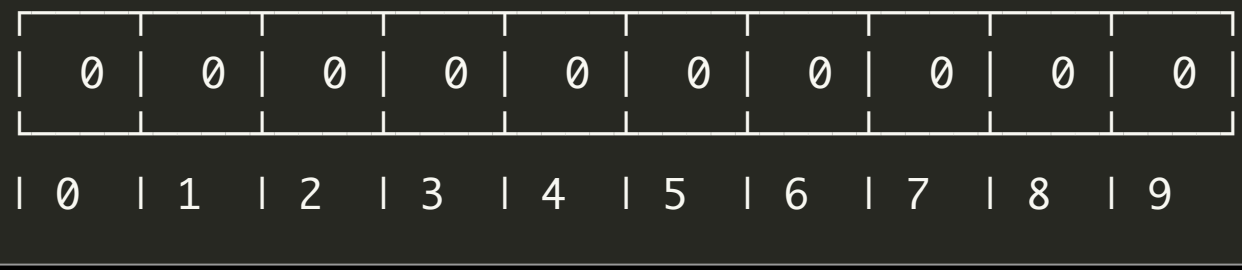
$$0 \ldots n-1$$

no real limit on number of elements

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

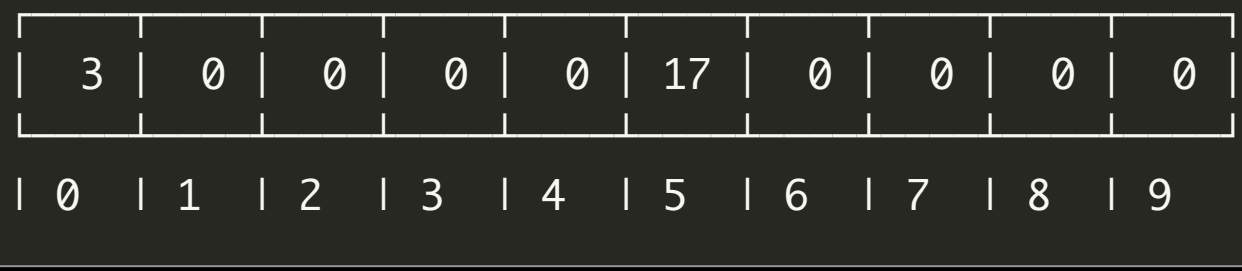
Arrays in C

```c
// Declare an array with 10 elements
// and initialises all elements to 0.
int myArray[10] = {0};
```



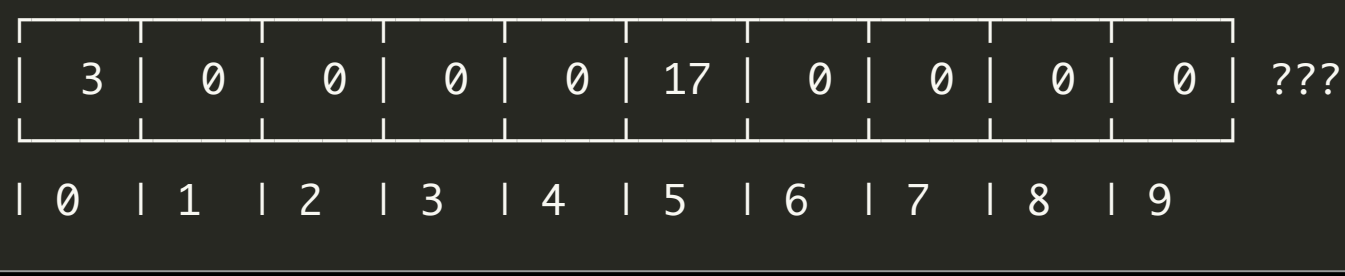
Arrays in C

```c
int myArray[10] = {0};
// Put some values into the array.
myArray[0] = 3;
myArray[5] = 17;
```



Arrays in C

```c
int myArray[10] = {0};
// Put some values into the array.
myArray[0] = 3;
myArray[5] = 17;
myArray[10] = 42; // <-- Error
```



Reading an Array

`scanf()` can't read an entire array.

this will only read 1 number:

```
#define ARRAY_SIZE 42
...
int array[ARRAY_SIZE];
scanf ("%d", &array);
```

instead, you must read the elements one by one:

```
int i = 0;
while (i < ARRAY_SIZE) {
    scanf ("%d", &array[i]);
    i++;
}
```

Printing an Array

`printf()` also can't print an entire array.

this won't compile...

```
#define ARRAY_SIZE 42
...
int array[ARRAY_SIZE];
printf ("%d", array);
```

instead, you must print the elements one by one:

```
int i = 0;
while (i < ARRAY_SIZE) {
    printf ("%d", array[i]);
    i++;
}
```

Copying an Array

given:

```
#define ARRAY_SIZE 5
int array1[ARRAY_SIZE] = {1, 4, 9, 16, 25};
int array2[ARRAY_SIZE];
```

this won't compile...

```
array2 = array1;
```

instead, you must copy the elements one by one:

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