Welcome!

COMP1511 18s1
Programming Fundamentals
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.)
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
// initialisation
int i = 0;

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

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

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

```java
int i = 5;
```
Variables

we can have as many variables as we like

```java
int i = 5;

int age = 18;

double pi = 3.14;
```
Lots of Variables

sometimes we want to store a lot of related variables

```c
int mark_student0 = 85;
```

```
| 85 |
```

```c
int mark_student1 = 90;
```

```
| 90 |
```

```c
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...

```c
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!

```c
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
// 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;
```

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>17</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
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:

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

instead, you must read the elements one by one:

```c
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...

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

Instead, you must print the elements one by one:

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

given:

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

this won't compile...

```c
array2 = array1;
```

instead, you must copy the elements one by one:

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