Welcome!

COMP1511 18s1

Programming Fundamentals

COMP1511 18s1

— Lecture 2 —

An Iffy Question

Andrew Bennett

<andrew.bennett@unsw.edu.au>

review: variables

decisions and conditions

constants with `#define`

Before we begin…

introduce yourself to the person sitting next to you

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

Feedback

talk **louder**

talk **slower**

lecture/content **pace**
(too fast, too slow)

lesson **summary**/**objectives**

Overview

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

create and use **variables** with type `int` and `double`

use `#define`s to represent constants in your code

construct a **flow control diagram** for a given situation

write code using **if statements**: `if`, `else if`, `else`

write code using **nested if statements**

(**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

make sure you have **home computing** set up

In Review: Variables

declare

the first time a variable is mentioned,
we need to specify its type.

initialise

before using a variable we need to assign it a value.

assign

to give a variable a value.

```
int num; // Declare
num = 5; // Initialise (also Assign)
...
num = 27; // Assign
```

In Review: Printing Variables Out

No variables:

```
printf ("Hello World\n");
```

A single variable:

```
int num = 5;
printf ("num is %d\n", num);
```

In Review: Printing Variables Out

More than one variable:

```
int num1 = 5;
int num2 = 17;
printf("num1 is %d and num2 is %d\n", num1, num2);
```

The order of arguments
is the order they will appear:

```
int num1 = 5;
int num2 = 17;
printf ("num2 is %d and num1 is %d\n", num2, num1);
```

In Review: Numbers in: `scanf`

```
int num = 0;
scanf ("%d", &num);
printf ("num = %d\n", num);
```

Note that the variable is still initialised.
(Not necessary, but good practice.)

Note the & before the variable name.
Don’t forget it!

In Review: Compiling

remember: we write C programs for **humans** to read.

a C program must be translated into *machine code* to be run.

this process is known as *compilation*,
and is performed by a *compiler*.

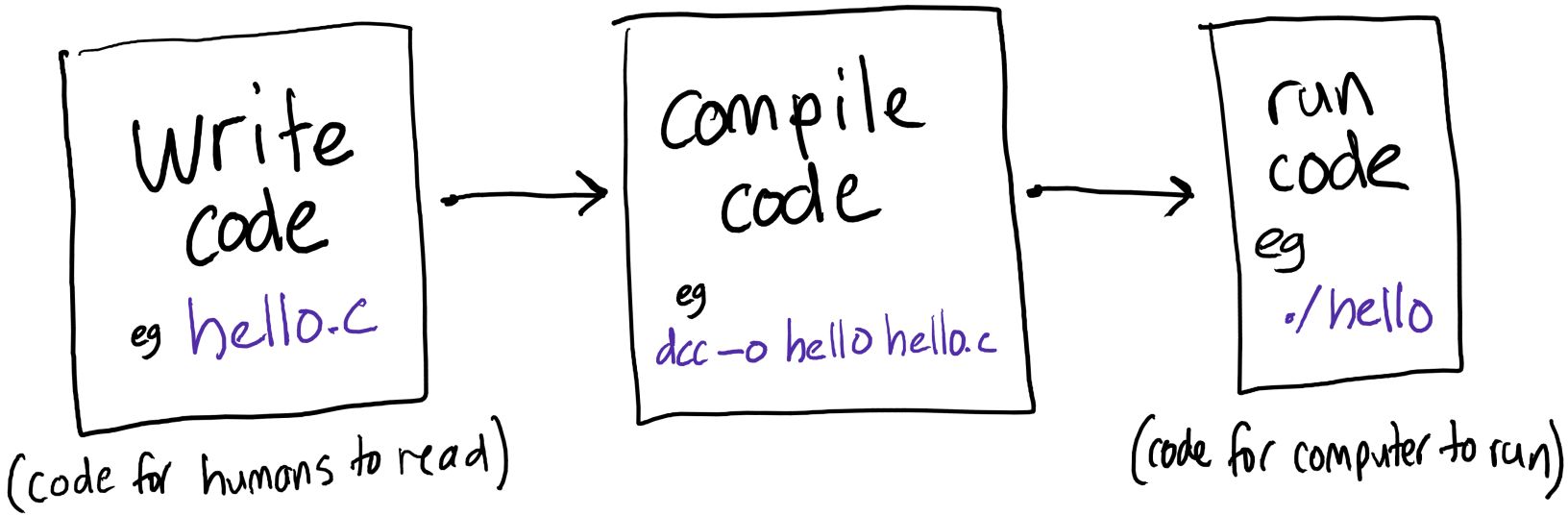
we will use a compiler named **dcc** for COMP1511
dcc is actually a custom wrapper around a compiler named clang.

another widely used compiler is called **gcc**.

The overall process

"What's the difference between the dcc command and the ./ command?"

There are three steps to writing + running code:



making **decisions**

different behaviour in different situations

In Review: Driving, Take 1

Write a program which asks the user to enter their age.

If they are at least 16 years old,
then, display “You can drive.”

Then, whether or not they can drive,
display “Have a nice day.”

In Review: Driving, Take 1

Write a program which asks the user to enter their age.

If they are at least 16 years old,
then, display “You can drive.”

Then, whether or not they can drive,
display “Have a nice day.”

In Review: Driving, Take 1: Step by Step

… Print “How old are you?”

… Read in their age.

… If their age is ≥ 16: print “You can drive”.

… Print “Have a nice day.”

```c
// Can a user drive?
// Andrew Bennett
// 2018-03-06

#include <stdio.h>

int main (void) {
    printf ("How old are you? ");
    int age = 0;
    scanf("%d", &age);
    if (age >= 16) {
        printf ("You can drive.\n");
    }

    printf ("Have a nice day.\n");

    return 0;
}
```

Driving, Take 2

Write a program which asks the user to enter their age.

If they are at least 16 years old,
then, display “You can drive.”
Otherwise, display “You can’t drive.”

Then, whether or not they can drive,
display “Have a nice day.”

Driving, Take 2

Write a program which asks the user to enter their age.

If they are at least 16 years old,
then, display “You can drive.”
Otherwise, display “You can’t drive.”

Then, whether or not they can drive,
display “Have a nice day.”

Driving, Take 2, Step by Step

… Print “How old are you?”

… Read in their age.

… If their age is ≥ 16: print “You can drive”.

… Otherwise, print “You can’t drive”.

… Print “Have a nice day.”

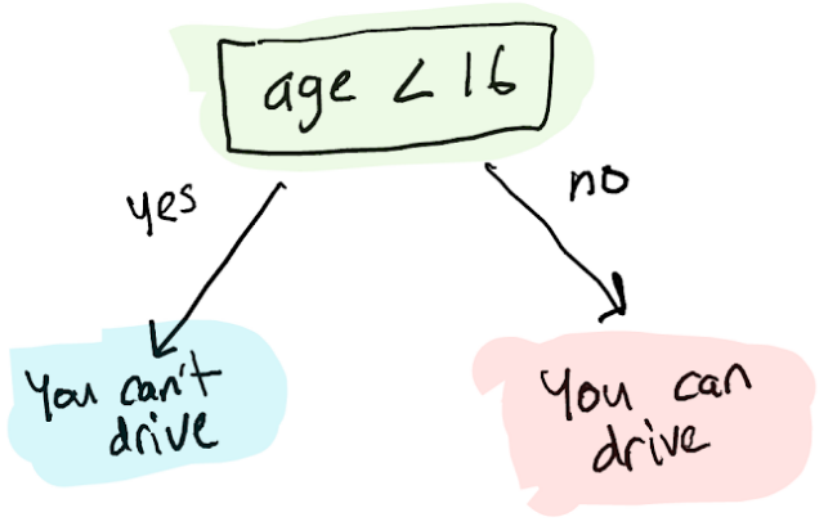
```c
// Can a user drive?
// Andrew Bennett
// 2018-03-06

#include <stdio.h>

int main (void) {
    printf ("How old are you? ");
    int age = 0;
    scanf("%d", &age);
    if (age >= 16) {
        printf ("You can drive.\n");
    } else {
        printf ("You can't drive.\n");
    }
    printf ("Have a nice day.\n");

    return 0;
}
```

Flow Control Diagrams




```
if (age < 16) {
    printf ("You can't drive\n");
} else {
    printf ("You can drive\n");
}
```

Detour: Defining Constant Values

Using the same value numerous times in a program
becomes high maintenance if the value changes…
and needs to be changed in many places.
(You may miss one!)

Other developers may not know (or you may forget!)
what this magical number means.

```
#define MIN_DRIVING_AGE 16
```

note

there is no semicolon at the end of this line

```c
// Can a user drive?
// Andrew Bennett
// 2018-03-06

#include <stdio.h>

#define MIN_DRIVING_AGE 16

int main (void) {
    printf ("How old are you? ");
    int age = 0;
    scanf("%d", &age);
    if (age >= MIN_DRIVING_AGE) {
        printf ("You can drive.\n");
    } else {
        printf ("You can't drive.\n");
    }

    printf ("Have a nice day.\n");

    return 0;
}
```

More Conditions!

Sometimes, we want to consider
more than two options for paths.

In the case of the driving scenario,
we want to make sure the age is ≥ 0 and ≤ 120…

Driving, Take 3

```
printf ("How old are you? ");
int age = 0;
scanf("%d", &age);
if (age < 0) {
    printf ("Invalid input.\n");
} else if (age < MIN_DRIVING_AGE) {
    printf ("You can't drive.\n");
} else if (age <= MAX_DRIVING_AGE) {
    printf ("You can drive.\n");
} else {
    printf ("Invalid input.\n");
}

printf ("Have a nice day.\n");
```

Conditions in C

less than

in maths, $<$; in C, `<`

less than or equal to

in maths, $\leq$; in C, `<=`

greater than

in maths, $>$; in C, `>`

greater than or equal to

in maths, $\geq$; in C, `>=`

equal to

in maths, $=$; in C, `==`

not equal to

in maths, $\neq$; in C, `!=`

Equals: Equality vs Assignment

equality: is this equal to?

e.g.

```
if (age == 18) {
    // then do something
}
```

assignment: make this be equal to

e.g.

```
// Set the variable 'age' to have the value 18
age = 18;
```

Nested Conditions

```
if (age >= MIN_DRIVING_AGE) {
    if (age <= MAX_DRIVING_AGE) {
        printf ("You can drive.\n");
    }
}
```

Logical Operators in C

useful when we want to check
multiple conditions in a single if statement.
C uses **Boolean logic**:

AND

in maths, $\wedge$; in C, `&&`

both expressions must be true

OR

in maths, $\vee$; in C, `||`

either or both expressions must be true

NOT

in maths, $\neg$; in C, `!`

the expression must be false

Nested Conditions, Redux

```
if (age >= MIN_DRIVING_AGE) {
    if (age <= MAX_DRIVING_AGE) {
        printf ("You can drive.\n");
    }
}
```

is the same as

```
if (age >= MIN_DRIVING_AGE && age <= MAX_DRIVING_AGE) {
    printf ("You can drive.\n");
}
```

An Iffy Answer

```
if (condition 1) {
    // Do stuff
} else if (condition 2) {
    // Do something else
} else if (condition 3) {
    // Do something completely different
} else {
    // In all other cases, do this.
}
```

programming style

what makes “good” code?

why do we care?

Indentation

```
if (condition 1) {
// Do stuff
} else if (condition 2) {
// Do something else
} else if (condition 3) {
// Do something completely different
} else {
// In all other cases, do this.
}
```

Indentation

```
if (condition 1) {
    // Do stuff
} else if (condition 2) {
    // Do something else
} else if (condition 3) {
    // Do something completely different
} else {
    // In all other cases, do this.
}
```

Indentation

```
if (condition 1) {
if (condition 2) {
if (condition 3) {
// Do stuff
} else if (condition 4) {
// Do something else
}
} else if (condition 5) {
if (condition 6) {
// Do something completely different
}
} else {
// In all other cases, do this.
}
}
```

Indentation

```
if (condition 1) {
    if (condition 2) {
        if (condition 3) {
            // Do stuff
        } else if (condition 4) {
            // Do something else
        }
    } else if (condition 5) {
        if (condition 6) {
            // Do something completely different
        }
    } else {
        // In all other cases, do this.
    }
}
```

re-introducing int

int

maximum possible value?

minimum possible value?

what happens if we go over/under?

introducing **double**

Working with doubles

declaring

```
double height = 1.65; // metres
```

scanf

```
scanf("%lf", &height);
```

printf

```
printf("Your height is: %lf metres\n");
```