Welcome!

COMP1511 18s1

Programming Fundamentals

COMP1511 18s1

— Lecture 3 —

Functions

Andrew Bennett

<andrew.bennett@unsw.edu.au>

even more if statements

functions

`while` loops?

Before we begin...

introduce yourself to the person sitting next to you

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

Overview

after this lecture, you should be able to…

represent more **complex** situations with **if statements**

understand what a **function** is

understand **why** we use functions

write **simple functions**

(**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

make sure you can send and receive **uni emails**

Review

More If Statements

demo: license.c

and now for something **new**…

Wouldn’t it be nice if…

… we didn’t have to **copy and paste** blocks of code?

… we could make parts of our code **reusable**?

… make our main function **smaller and simpler**?

… make our programs **nicer to read**?

introducing: **functions**

What is a Function?

you’ve already seen functions outside programming:

$\cos, \sin, \ldots$

functions are like a black box.

What is a Function?

you've already seen functions *inside* programming!

`printf`, `scanf`

```
int main (int argc, char *argv[]) { ...
```

What is a Function?

functions are way of achieving **abstraction**

Abstraction

"… creating *units* which can be *reused*,
and whose internal details
are *hidden* from outside inspection …"

Abstraction via Functions

Functions allow us to:

separate out, or **encapsulate**
a piece of code serving a single purpose

test and **verify**
a piece of code

reuse
a piece of code

shorten our programs,
making it easier to
modify and **debug**

Anatomy of a Function

return type

function name

parameters

(inside parens, comma separated)

return statement

```
int addNumbers (int num1, int num2) {
    int sum = num1 + num2;
    return sum;
}
```

let's try it!

Functions with No Parameters

parameter list: **void**

```
int getRandomNumber (void) {
    // chosen by fair dice roll...
    // guaranteed to be random
    return 4;
}
```

Functions with No Return Value

return type: **void**

no return statement necessary

```
void printAsterisks (void) {
    printf ("*****");
}
```

Function Prototypes

every function has a **function prototype**:
tells the compiler that
the function exists,
and the structure it has.

includes **key information**
about the function.

```
int addNumbers (int num1, int num2);
int getRandomNumber (void);
void printAsterisks (void);
```

Noteworthy Features

a function can have zero or more parameter(s)

a function can only return zero or one value(s)

```
* * *
```

a function stores a local copy of parameters passed to it

the original values of variables remain unaltered

parameters received by the function,
and local variables created by the function,
are all **discarded** when the function returns