

COMP1511/1911 Programming Fundamentals

Week 3 Lecture 1

Functions and Style

Announcements: This week

Reminder: **Lab 2 due** tonight 6pm.

Help Session Schedule

- [COMP\(1511|1911\) 26T3 – COMP\(1511|1911\) Help Sessions](#)
- Please note that help sessions are not in labs and are BYOD (Bring Your Own Device) sessions. 

Drop in Sessions

Email the course account if you feel you need more one on one time and would like to book a longer one-on one session for extra help.

Announcement: Public Holiday

Next Week on Monday is a Public Holiday

Pre-recorded lecture will be uploaded

Students in Monday **tutorial/labs** can book tickets to attend extra make up classes

<https://buytickets.at/comp1511unsw/2401086>



Last Week in Lectures

- if statements
 - scanf returns!
- while loops
 - nested while loops
- custom data types
 - structs
 - enums

Today's Lecture

- **Quick Recap**

- Nested while loops, structs, enums, structs containing enums

- **Functions**

- Breaking code into smaller chunks
- Another way to avoid duplicating code

- **Style**

- Style guide and tips for good style

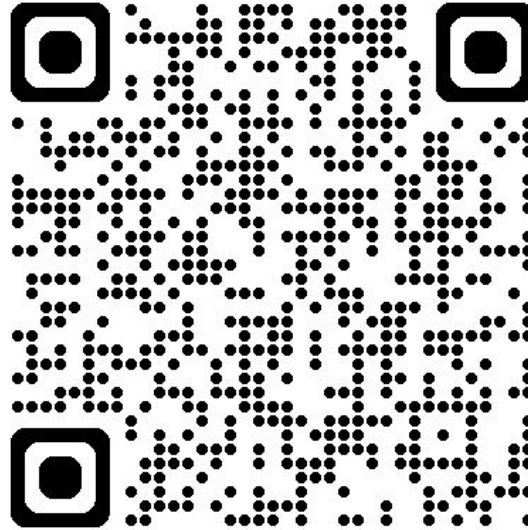
Most students find things are starting to get hard



Be patient and keep practicing

Link to Week 3 Live Lecture Code

https://cgi.cse.unsw.edu.au/~cs1511/26T3/code/week_3/



Recap: Nested Loop Exercise

```
0 0 0 0 0
1 1 1 1 1
2 2 2 2 2
3 3 3 3 3
4 4 4 4 4
```

Create this pattern

```
0 0 0 0 0
1 1 1 1 1
2 2 2 2 2
3 3 3 3 3
4 4 4 4 4
```

Hint: Start with this easier pattern first and then modify it

`pattern.c`

Recap: Organising related data with structs

Once we have lots of variables,
we need ways to organise related data



Recap: structs

A way to organise related data

- We define the struct type
- Create variables of that type
- Use . to access members

```
struct student {  
    char first_initial;  
    char last_initial;  
    int age;  
    double lab_mark;  
};
```

```
int main(void) {  
    // Declare a variable  
    // of type struct student  
    struct student brianna;  
    // Assign values to  
    // struct members  
    brianna.first_initial = 'B';  
    brianna.last_initial = 'K';  
    brianna.age = 21;  
    brianna.lab_mark = 9.9;
```

Recap enums

An **enum** defines a group of named integer constants

```
// Note: We will define enum types before the main
// Define an enum with days of the week
// MON will have value 0, TUE 1, WED 2, etc
enum weekdays {MON, TUE, WED, THU, FRI, SAT, SUN};

int main (void) {
    enum weekdays day = SAT;
    // This will print out 5
    printf("The day number is %d\n", day);
    return 0;
}
```

struct with enum members!

- We can have enum members in our structs!

```
enum direction {  
    NORTH, SOUTH, EAST, WEST  
};  
  
struct player {  
    enum direction current_dir;  
    int score;  
};
```

```
struct player hero;  
hero.current_dir = EAST;  
hero.score = 0;
```

Coding Example: structs containing enums

`pokemon.c`

- We can have enum members in our structs!
 - Create an enum for pokemon types FAIRY, WATER, FIRE etc
 - Create a struct called pokemon with a field for the type and some other relevant fields
 - Make a pokemon variable and set it with data
 - Print out the pokemon data

Functions

Have you seen functions before?

- **Yes!** You have been using functions from the start!!!
- You have been writing **main** functions
 - `int main(void)`
- You have also used functions from the **stdio.h** library
 - **printf** and **scanf**
- There are lots of other libraries and library functions
- **Today we will write our own functions**

Why do we use Functions?

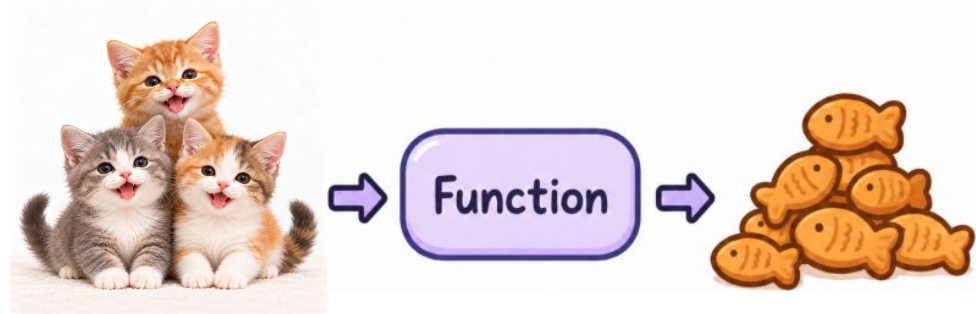
- **Functions are a way to break code into smaller chunks**
 - Break a bigger program into manageable pieces
- **Easier to read**
 - Give each piece a clear job and a meaningful name.
- **Reuse code**
 - Call the same function again instead of duplicating code.

Cat Cafe: Let's write our First Function 🐱

The **Cat Cafe** gives each cat that visits some complimentary treats.

- Each cat gets 2 free treats

We want a function that works out the total number of treats.



Number of cats

Number of treats

Designing our Function

Before we write the code we should design the function

- What should it be **named**?
- What **input** does it need?
 - Number of cats. What type should this be?
- What should it **return**?
 - We want it to calculate and return the number of treats needed for the given number of cats. What type should this be?

Once we do this we are ready to write our function prototype!

Function Prototypes

This is a function prototype

- Gives programmers and the compiler information about how the function can be used

return type:

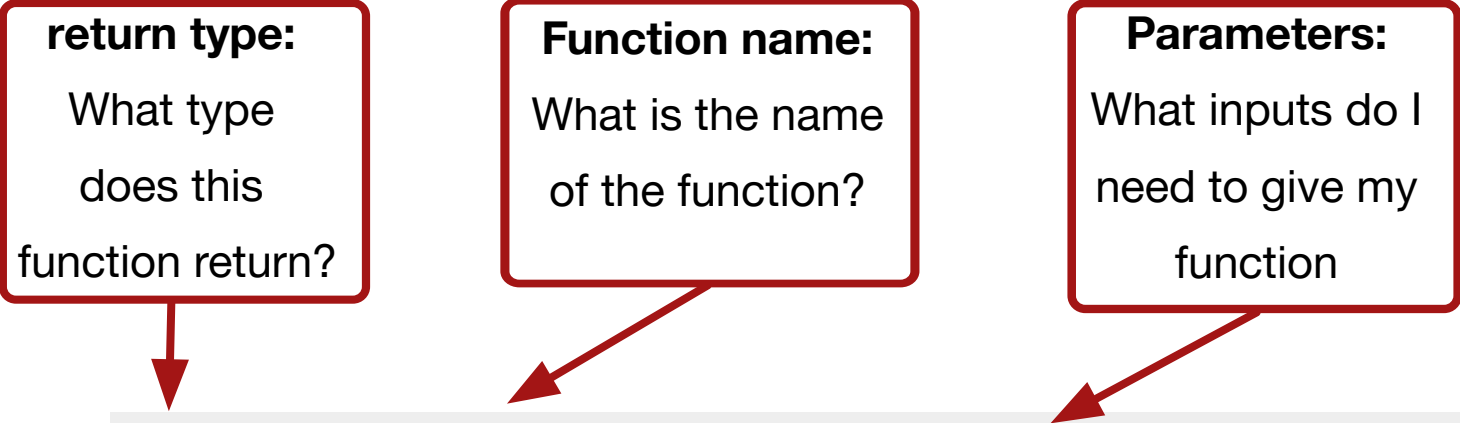
What type
does this
function return?

Function name:

What is the name
of the function?

Parameters:

What inputs do I
need to give my
function



```
int total_cat_treats(int cats);
```

Function Definition

This is an implementation of our function!

```
#define TREATS_PER_CAT 2

// Returns the total number of
// complimentary cat treats
// for the given number of cats
int total_cat_treats(int cats) {
    int total = cats * TREATS_PER_CAT;
    return total;
}
```

The **return** statement

- Ends the function
- Lets us return a value back to the code that called the function.

Function calls

Here is an example of how we could use our function.

```
int total_cat_treats(int cats);

int main(void) {
    // Call the function and pass in an int value
    // Store the return value in the treats variable
    int treats = total_cat_treats(7);
    printf("Enjoy %d complimentary treats\n", treats);
    return 0;
}
```

Function Calls

When we **call** a function:

- Pass in the correct sequence of inputs (**arguments**)
 - The correct type
 - The correct order
- The function code **executes**
- The function **returns**
 - This happens when it uses the keyword return or when it reaches the end of the function if it has no value to return (void functions)
- If it returned a value, we can store it or use it!
- **Execution continues** after the function call.

Exercise: Design another Function

Write a function that calculates and returns the **total cost** of the Cat Cafe visit.

The function depends on:

- the number of cats
- the number of hours spent at the cafe

What should the function prototype be?

- What should the function be called?
- What are the parameters?
- What should it return?

Exercise: Write the function

The prototype could be

```
double calculate_total_cost(int cats, double hours) ;
```

Assume the cafe uses these fixed costs:

- \$8.50 per cat per hour
- \$12.00 cleaning

Assume the cost is $\text{cats} * \text{hours} * \$8.50 + \$12.00$

Write the function and call it in the main cat cafe function we are working on.

Let's write another function

We want to write a function that takes a char and prints the associated cat sound

What should the function prototype be?

- **Function Name**
 - What should the function be called?
- **Parameters**
 - What input does it need?
- **Return type**
 - Does it have a return a value?

char	Text to print
'p'	Purr
'm'	Meow
'h'	Hiss
Anything else	Stares Silently

void functions

Our function prints to the terminal.

- It does not return any values back to be used in the code.
- This means we use a **void** return type.

This is what our prototype might look like

```
void print_cat_sound(char sound) ;
```

Note: Functions with void return types are called **void functions** or sometimes **procedures**

void function example

```
void print_cat_sound(char sound);

int main(void) {
    print_cat_sound('p');
    return 0;
}

void print_cat_sound(char sound) {
    if (sound == 'm') {
        printf("Meow!\n");
    } else if (sound == 'p') {
        printf("Purr...\n");
    } else if (sound == 'h') {
        printf("Hiss!\n");
    } else {
        printf("The cat stares silently.\n");
    }
}
```

void parameters

Some functions do not need parameters.

- We use void in the parameter list in these cases
- This function has void for both return type and parameter list

```
void welcome(void) ;

int main(void) {
    welcome() ;
    return 0 ;
}

void welcome(void) {
    printf("=====\n") ;
    printf(" Welcome to the Cat Cafe!\n") ;
    printf("=====\n") ;
}
```

Exercise: cat_cafe.c

Write a function to

- prompt the user to enter the number of cats
- read it in with scanf
- return the number of cats.

Use this function in the main function to read in the number of cats from the user.

Prototypes and Style

- It is good style to have
 - main function at the top of the file
 - Implement additional user defined functions below it.
- To do this write prototypes above main function
 - The compiler processes the program code top-down
 - A prototype declares the function before its definition appears later
 - The compiler must see the prototype before the function is called

Function Comments and Style

- Every function must have a comment placed before the function implementation describing
 - the purpose of the function
 - any side-effects the function has
- As always, choose meaningful names for your functions

Summary: Function Calls and Execution Flow

- The code of a function is only executed when requested via a **function call**
- When a function is called
 - Current code execution is halted
 - Execution of the function body begins
 - Reaching the last statement of the function or reaching a return statement stops execution of a function
- When the function completes, execution resumes at the instruction after the function call.

Summary: Function Parameters and return

- Functions have **parameters**
 - define what type of arguments (inputs) a function needs
 - **void** in the parameter list means it needs no arguments
 - Arguments must be passed in the correct order with correct types
- Functions may **return** a single value
 - A function's type is determined by its return type
 - A return type of **void** means no value is returned
 - **return** can still be used to end a void function
 - A void function can't be used where a value is required:
 - `int x = some_function(3); //invalid for void function`

Summary: Benefits of functions

- **Modularity**: Breaks complex programs into simpler, manageable pieces, easier to read and understand.
Allows us to **test** and **debug** smaller chunks of code in isolation
- **Reusability**: Avoids code duplication, as you can reuse the functions
- **Abstraction**: Hides the implementation details and allows you to focus on higher-level logic.

Exercise: pokemon_functions.c

Let's try to improve the style and remove duplication from our pokemon program by moving code into functions.

Style

Style

- The code we write is for human eyes
- We want to make our code:
 - easier to read
 - easier to understand
- This results in
 - Fewer mistakes
 - Faster development time

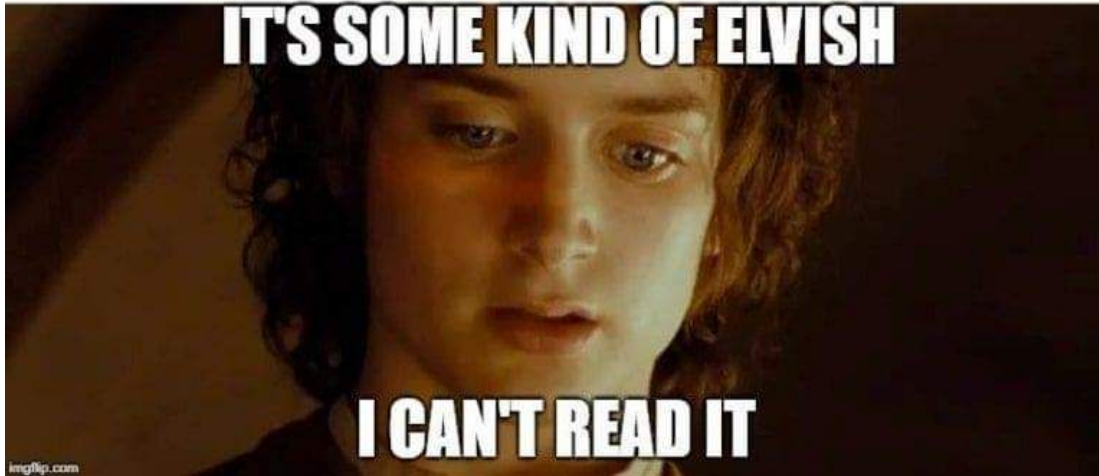


Style Guide

- We have a style guide in 1511 that we encourage you to use to establish good coding practices early:
 - https://cgi.cse.unsw.edu.au/~cs1511/26T3/resources/style_guide.html
- Different organisations will have their own style guides. However, the basics remain the same.
- Style and marks in this course
 - Your assignment will have **style marks** attached to it
 - We need to mark your code in the final exam manually. It is good if it is not a dog's breakfast.

Bad Style Demo

When you trying to look at
the code you wrote a month ago



Let's look at
`bad_style.c`

What are some
things we should fix?

Tips: Clean as you go

- Write comments where they are needed
- Name your variables based on what that variable is there to do
- In your block of code surrounded by {}:
 - Indent 4 spaces
 - Vertically align closing bracket with statement that opened it
- One expression per line
- Consistency in spacing
- Watch your code width (≤ 80 characters)
- Watch the nesting of IFs - can it be done more efficiently?
- Break code into functions

Things are getting harder...

- If you do not understand something, do not panic!
 - It is normal not to understand a concept the first time it is explained
- If you can't solve a problem
 - Break it into smaller and smaller steps until there is something you can do
 - Ask lots of questions and come to help sessions!
- Learning is hard and takes time
- Solving problems is hard and takes practice
- We are here to help you!!!

Feedback Please!

Your feedback is valuable!

If you have any feedback from today's lecture, please follow the link below or use the QR Code.

Please remember to keep your feedback constructive, so I can action it and improve your learning experience.



<https://forms.office.com/r/qK0Z00Y0gX>

What did we learn today?

- Recap of while loops, nested while loops
 - pattern.c
- Recap of structs
 - struct_student.c
- Enums
 - enum_weekdays.c
- Enums and structs
 - pokemon.c

What did we learn today?

- Functions
 - cat_cafe.c
 - print_pokemon.c
- Style
 - bad_style.c

Coming up next lecture....

More about functions and a very **important topic - arrays.**

Reach Out

Content Related Questions:

[Course Forum](#)

Admin related Questions email:

cs1511@unsw.edu.au



Student Support - I Need Help With...

My Feelings and Mental Health

Feeling depressed, overwhelmed
or not your usual self?



Mental Health Connect
student.unsw.edu.au/mhc



TalkCampus
student.unsw.edu.au/Talkcampus



UNSW (24/7) Mental Health Support: [+61\(2\) 9385 5418](tel:+61293855418)

Text support After Hours: [0485 826 595](tel:0485826595)



Offshore
Call 24-hour Medibank Hotline: [+61 2 8905 0307](tel:+61289050307)

Uni and Life in Australia

Stress, financial, visas,
Accommodation & More



Student Support
Indigenous Student Support

student.unsw.edu.au/advisors
nura-gili-centre-indigenous-programs

Reporting Sexual Assault/Harassment



Equity Diversity & Inclusion (EDI)

edi.unsw.edu.au/sexual-misconduct

Educational Adjustments

To manage my studies and
disability / health condition



Equitable Learning Services (ELS)

student.unsw.edu.au/els

Academic and Study Skills



Academic Skills

student.unsw.edu.au/skills

Special Consideration

Because life impacts our
studies and exams



Special Consideration

student.unsw.edu.au/special-consideration

Physical Health

Doctors visits, Vaccinations



Health Service

student.unsw.edu.au/hsu