COMP1511 Programming Fundamentals

Week 3 Lecture 1

FUNctions and Style

Last Week

- if statements
- scanf returns!
- while loops
- nested while loops
- structs
- enums

This Week

- Reminder: Lab 2 due tonight 6pm.
- Help Session Schedule
  - https://cgi.cse.unsw.edu.au/~cs1511/25T3/help-sessions
  - Please note that help sessions are not in labs and are BYOD (Bring Your Own Device) sessions. 💻
  - Please check out **this link** for more information about help sessions(Teams join code:ijnb1b0)

Next Monday

- Public Holiday
  - Pre-recorded lecture will be uploaded
  - Students in monday tutorial/labs, will be able to get tickets to attend other time slot in the week.
  - Ticketing Link : **https://buytickets.at/comp1511unsw/1857310** (Access code is “COMP1511”)
  - Lab 3 Due Week 4 Tuesday 6pm instead of Monday 6pm

Today's Lecture

- Recap of nested while loops, structs, enums
- structs with enums members
- Functions
- Style

Most students start to find things are getting hard
Be patient and keep practicing.

Link to Week 3 Live Lecture Code

https://cgi.cse.unsw.edu.au/~cs1511/25T3/code/week_3/

A Brief Recap

- Nested While Loops

  `pattern.c`

- Structs

  `struct_student.c`

  `enum_weekdays.c`

struct with enum members!

- We can have enum members in our structs!

```c
enum student_status {
    ENROLLED, WITHDRAWN, LEAVE
};

struct student {
    enum student_status status;
    double wam;
};
```

```c
struct student z123456;
z123456.status = ENROLLED;
z123456.wam = 95.9;
```

Coding Example:

`pokemon.c`

- We can have enum members in our structs!
- Create a 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

What is a function?

Functions

- A function is an
  - independent
  - reusable block of code
  - that performs a specific task

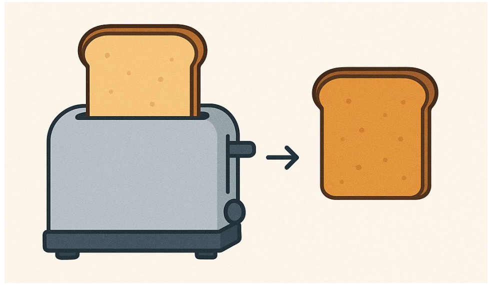
Have you seen functions before?

Functions

- Yes you have seen functions before!
- You have been writing **main** functions
- You have also used functions from the **stdio.h** library
    - **printf** and **scanf**
- There are lots of other libraries and library functions
- And we can write our own functions too

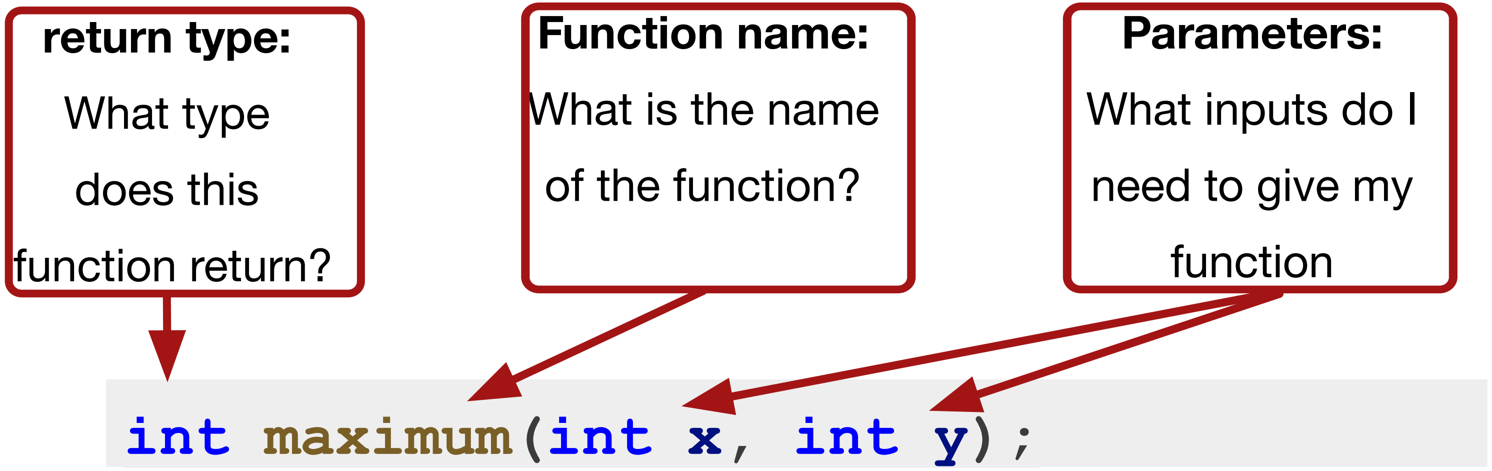
Using Functions

- We do not need to know how the code in the function works - just what it does and how to use it.
- To use a function, we do a **function call**
  - Pass in the correct sequence of inputs (arguments)
    - the correct type
    - the correct order
  - If our function has a return value
    - we may wish to use or store it



Function Prototypes

- This is a function prototype
  - it 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

```c
int maximum(int x, int y);
```

Functions calls

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

```c
int maximum(int x, int y);

int main(void) {
    int num = 7;
    int max = maximum(10, num);
    printf("The maximum value is %d\n", max);
    return 0;
}
```

Function Definition

This is an implementation of our function!

```c
// Returns the maximum of the given values x and y
int maximum(int x, int y) {
    int max;
    if (x > y) {
        max = x;
    } else {
        max = y;
    }
    // returns an int value
    return max;
}
```

Let’s try writing another one

We want to create a function to print out a square of a given size.

For example this is a square of size 4

```
****
****
****
****
```

What would the prototype be?

void functions

Our function does not need to return any values

It just prints things out to the terminal.

This is what our prototype might look like

```
// Prints a square of '*' characters
// of the given size
void print_square(int size);
```

Note: Functions with void return types are sometimes called procedures

void functions

```c
void print_square(int size);

int main(void) {
    int square_size;
    printf("Enter the size: ");
    scanf("%d", &square_size);
    print_square(square_size);
    return 0;
}
```

```c
void print_square(int size){
    int row = 0;
    while (row < size) {
        int col = 0;
        while (col < size) {
            printf("#");
            col = col + 1;
        }
        printf("\n");
        row = row + 1;
    }
}
```

Functions that need no arguments

```c
void print_warning(void);

int main(void) {
    print_warning();
    return 0;
}
// Display a helpful warning on the terminal
void print_warning(void) {
    printf("########################\n");
    printf("Warning: Don't plagiarise\n");
    printf("########################\n");
}
```

We can use the keyword **`void`** in the parameter list too!

Recap: Function Parameters and return

- Functions have **parameters**
  - define what type of arguments (inputs) the functions need
  - `void` in the parameter list means it needs no arguments
- Functions may **return** a single value
  - The type of the function is the type that it returns
  - A return type of `void` means it does not return a value
    - `return` can still be used to end the function without a return value

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.

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 we need to write prototypes above main function
  - the compiler processes the program code top-down
  - This lets the compiler know that the definition (implementation) for these functions can be found somewhere else.
  - A compile error occurs if a function call is encountered before the function prototype.

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

Code demo

area_triangle.c

print_pokemon.c

Benefits of functions

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

Style

Style

- The code we write is for human eyes
- We want to make our code:
  - easier to read
  - easier to understand

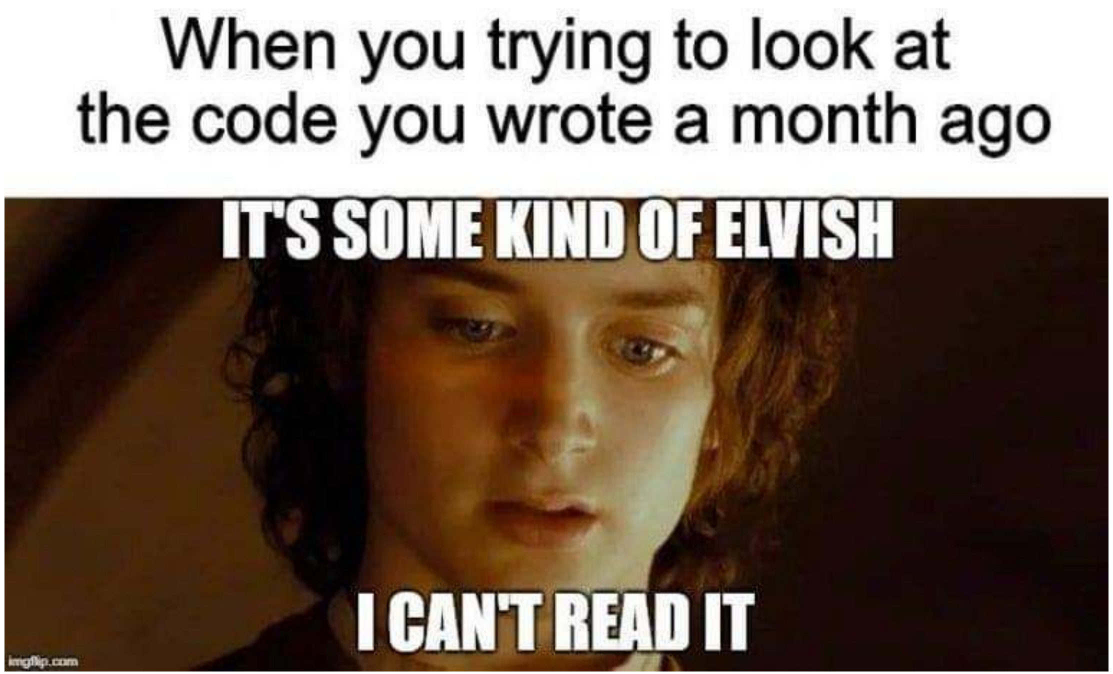
Benefits of Good Style

- less possibility for mistakes
- helps with faster development time
- you also get marks for style in assignments
- if we need to mark your code in the final manually it is good if it is not a dog’s breakfast

What is Good Style?

- Indentation and Bracketing
- Names of variables and functions
- Structuring your code
- Nesting
- Repetition
- Comments
- Consistency

Bad Style Demo



Let's look at bad_style.c

- What are some things we should fix?

Style Guide

- Often different organisations you work for, will have their own style guides, however, the basics remain the same across
- Your assignment will have style marks attached to it
- 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/~cs1511/25T3/resources/style_guide.html**

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 perfectly normal to not understand a concept the first time it is explained to you
    - ask questions in lectures
    - try and read over the information again
    - rewatch lectures
    - ask questions in the tutorial and the lab
    - ask questions on the forum
    - go to help sessions
    - go to revision sessions

Things are getting harder...

- If you can't solve a problem
    - break down the problem into smaller and smaller steps until there is something that you can do
    - ask us lots of questions!
- Remember 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
  - max_function.c square_function.c warning_function.c area_triangle.c pokemon_functions.c
- Style
  - bad_style.c

Coming up next lecture....
Function recap, more about functions and a very important topic - arrays.

Reach Out

Content Related Questions:

Forum

Admin related Questions email:

cs1511@unsw.edu.au