COMP1511 PROGRAMMING FUNDAMENTALS

LECTURE 5

Lets get into code style and functions!

LAST LECTURE...

LAST WEEK, WE TALKED:

- Played with making some decisions and using IF statements with conditionals
- Looped the loop (WHILE)
- Talked about scanf() and how eccentric it is
- Started to learn about structs and enums

THIS LECTURE...

TODAY...

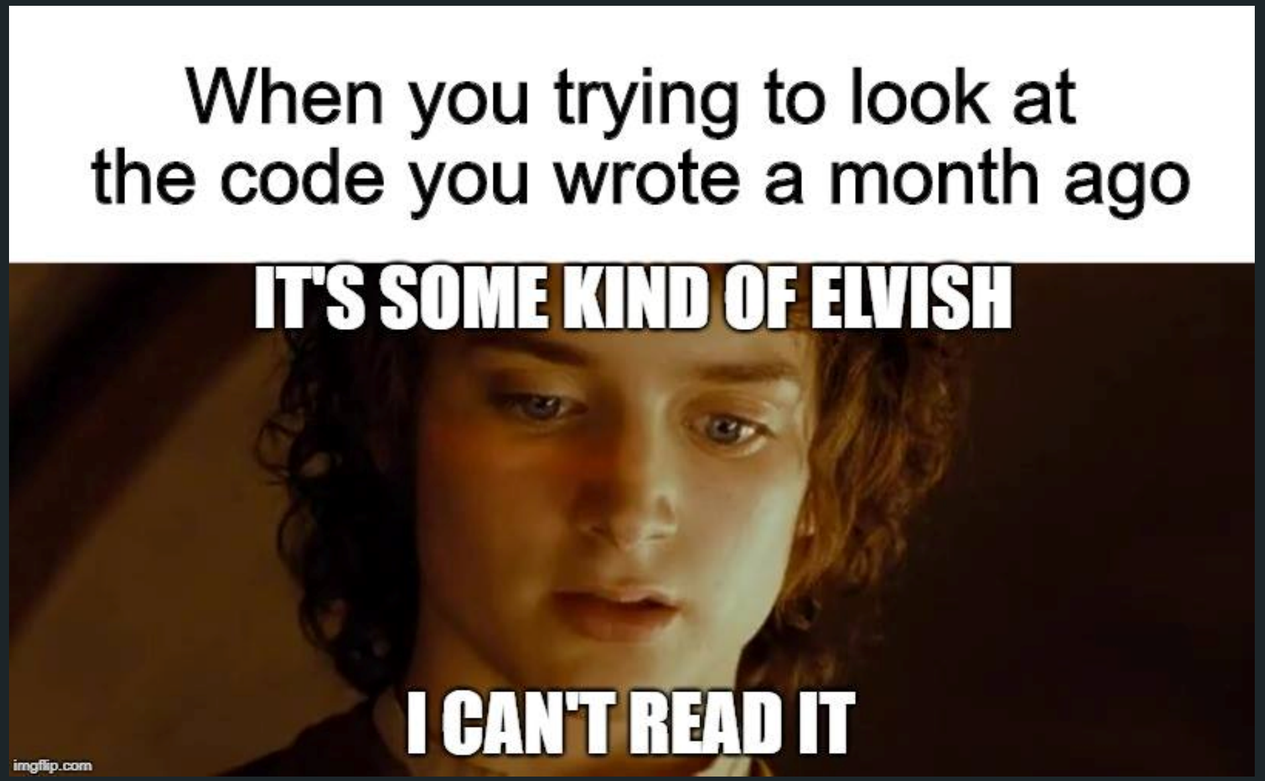
- Style
- Functions

WHERE IS THE CODE?

Live lecture code can be found here:

HTTPS://CGI.CSE.UNSW.EDU.AU/~CS1511/22T2/LIVE/WEEK03/

WHAT IS STYLE?
WHY STYLE?



WHAT IS STYLE? WHY STYLE? IS IT WORTH IT?

- The code we write is for human eyes
- We want to make our code:
  - easier to read
  - easier to understand
  - neat code ensures less possibility for mistakes
  - neat code ensures faster development time
- Coding should always be done in style - it is worth it...

WHAT IS GOOD STYLE?

- Indentation and Bracketing
- Names of variables and functions
- Structuring your code
  - Nesting
  - Repetition
- Comments where comments need to be
- Consistency

When I read your code, I should be able to understand what that code does just from your structure and variable names

BAD STYLE

:(

bad_style.c

- Let’s have a look at some bad style…
- How are you guys feeling? Have you fainted in shock and in horror?
- Let’s work with this code to tidy it up before I develop a permanent eye twitch…
  - Start from the smallest things that are easy to do straight away
  - What can you attack next?

KEEP IT CLEAN AS YOU GO

MUCH EASIER THAN MAKING YOUR WAY THROUGH A DUMPSTER FIRE OF MESS

- 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
  - line up closing bracket with the statement that opened them vertically
- One expression per line
- Consistency in spacing
- Watch the nesting of IFs - can it be done more efficiently?

1511 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/~cs1511/22T2/resources/style_guide.html

SOME NEAT SHORTHAND

INCREMENTING AND REPEATING OPERATIONS

- Increment count by 1

```
count = count + 1;
count++;
```

- Decrement count by 1

```
count = count - 1;
count--;
```

SOME NEAT SHORTHAND

INCREMENTING AND REPEATING OPERATIONS

- Increment count by 5

```
count = count + 5;
count += 5;
```

- Decrement count by 5

```
count = count - 5;
count -= 5;
```

- Multiply count by 5

```
count = count * 5;
count *= 5;
```

BREAK TIME

TIME TO STRETCH

Pick a positive number (any number). If the number is even, cut it in half; if it's odd, triple it and add 1. Can you pick a number that will not land you in a loop?

https://www.quantamagazine.org/why-mathematicians-still-cant-solve-the-collatz-conjecture-20200922/

FUNCTIONS

FINALLY!

- So far, we have talked about ”procedures” in our C programs as a way to bunch together a few commands
- We have also used these things like printf(), scanf() and the main(), and have heard the word function being thrown around... but what does this actually mean? What’s going on here?

FUNCTIONS VS PROCEDURES

- A function is a way to break down our codes into smaller functional bits
  - Each function performs some sort of operation
  - Each function has inputs and an output (you may still have an empty input or output, depending on what the role of that function is)
  - We can call our function from anywhere in our code to perform its job and then return something to the spot it was called from

FUNCTIONS VS PROCEDURES

(THEY ARE THE SAME THING)

return type:
procedures always return nothing (void)

name:
What will I name my function/procedure?

input/ arguments:

```
void print_instructions (void) {
    printf(...);
    do_something();
}
```

This is a procedure. Why? Because it returns void (nothing)

FUNCTIONS

WHAT DO WE NEED TO KNOW?

return type: What type does this function return?

name of function: What will I name my function?

input/ arguments: What am I giving my function?

```
int add (int number_one, int number_two) {
    int sum;
    sum = number_one + number_two;
    return sum;
}
```

A function, which adds two numbers together and returns the result

To finish I return an int (sum), which is what I said I would return when I wrote my function

COMPARE

```
void my_procedure (int x, int y) {
    printf(...x, y);
    do_something();
}
```

```
int my_function (int x, int y) {
    int sum;
    sum = number_one + number_two;
    return sum;
}
```

SUM.C

DEMO
(FOLLOW ALONG)

FUNCTIONS

TELLING C I HAVE SOME FUNCTIONS THAT I WANT TO USE: PROTOTYPES

So now we have moved two steps out to be ther own functions. We now have a function to add two numbers together:

```
int add (int die_one, int die_two)
```

And a function to compare:

```
void comparison (int sum)
```

Just to remind you that C reads things in order from top to bottom, so it will not know these functions exist when we call to them! What can we do to fix that?

FUNCTIONS

PROTOTYPES

```
 7
 8 #include <stdio.h>
 9
10 // 1. Scan in the two dice (scanf()))
11 // 2. Add the numbers together (+)
12 // 3. Check the sum against the target number (#define)
13 // 4. Output greater or less than (printf())
14
15 #define TARGET 9
16
17 int add(int die_one, int die_two);
18 void comparison(int sum);
19
20 int main (void) {
```

We let C know in the very beginning before main about each function that we will use, by creating a function prototype:

- This is a very basic definition of the function to let C know those functions are included somewhere in this file! It is like declaring a variable, but I am declaring a function - note the semi colon at the end of each statement! So for our add and compare functions:

```
int add (int die_one, int die_two);

void comparison (int sum);
```

Feedback please!

I value your feedback and use it to pace the lectures and improve your overall learning experience. If you have any feedback from today's lecture, please follow the link below. Please remember to keep your feedback constructive, so I can action it and improve the learning experience.

https://forms.microsoft.com/r/dKssTn3AU4

WHAT DID WE LEARN TODAY?

STYLIN'

bad_style.c

FUNCTIONS

breaking down the problem into actionable steps

function_demo.c

REACH OUT

CONTENT RELATED QUESTIONS

Check out the forum

ADMIN QUESTIONS

cs1511@cse.unsw.edu.au