

COMP1511 PROGRAMMING FUNDAMENTALS

# LECTURE 5

Stylin' and let's finally look at what a function is

# 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

# THIS LECTURE...

## TODAY...

- Style
- Functions

“

**WHERE IS THE CODE?**

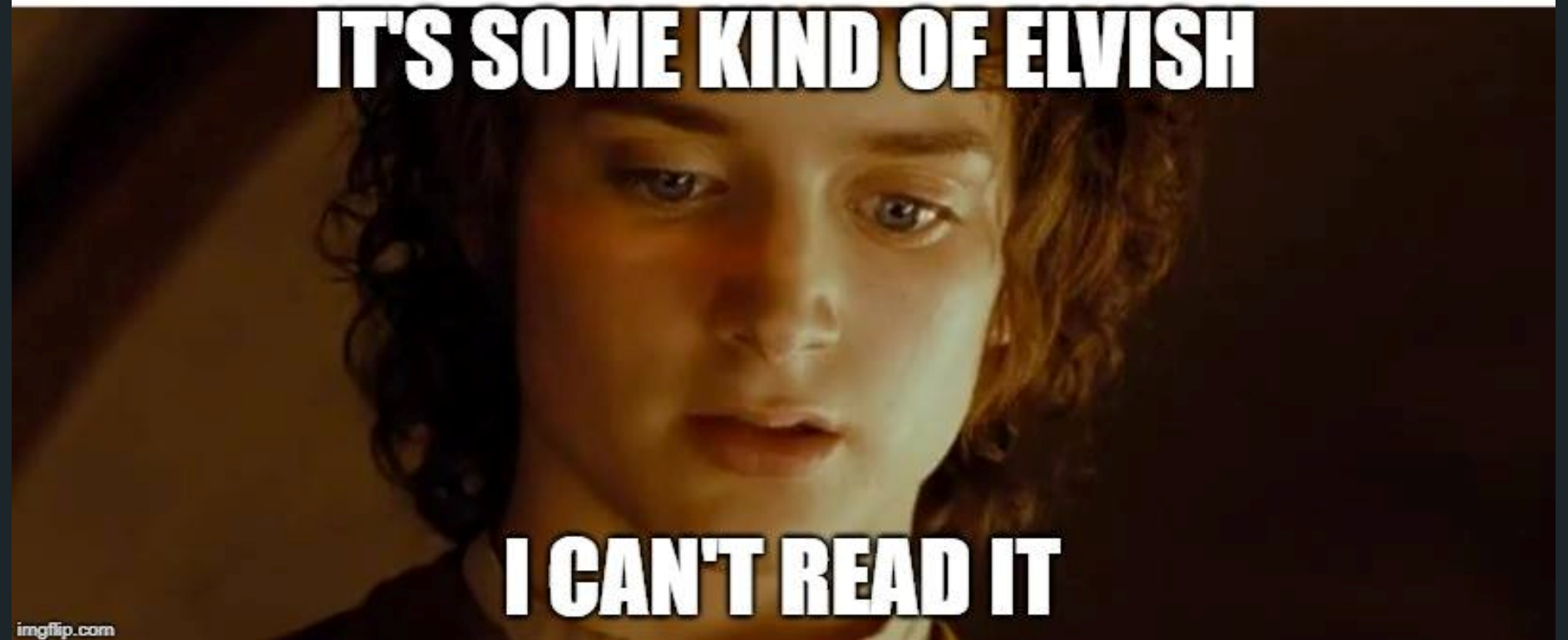
**Live lecture code can be found here:**

[HTTPS://CGI.CSE.UNSW.EDU.AU/~CS1511/22T1/LIVE/WEEK03/](https://cgi.cse.unsw.edu.au/~cs1511/22T1/LIVE/WEEK03/)

**WHAT IS  
STYLE?  
WHY  
STYLE?**

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

**IT'S SOME KIND OF ELVISH**



**I CAN'T READ IT**

imgflip.com

# 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/~cs1511/22T1/resources/style\\_guide.html](https://cgi.cse.unsw.edu.au/~cs1511/22T1/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;`

# OTHER NEAT SHORTHAND

## ASKING QUESTIONS INSIDE OUR CONDITION OR RETURNING AN OPERATION

- Remember when we checked that `scanf()` returned something by doing this:

```
int scanf_return;  
scanf_return = scanf("%d", &number);  
if(scanf_return != 1) {...
```

- You can actually call functions inside your if statements or your while loops, as long as that function returns something that can be checked

```
if (scanf("%d", &size) != 1)
```

**AT THIS  
POINT IN  
WEEK 3,  
THIS IS  
WHERE  
THINGS  
START TO  
GET 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 - try and read over the information again, ask questions in the tutorial and the lab - we are here to help you and to make sure that you are comfortable with the content.
- If you can't solve a problem, break down the problem into smaller and smaller steps until there is something that you can do and ask us lots of questions!
- Remember learning is hard and takes time
- Solving problems is hard and needs practice

# 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, you have heard me refer to `printf()`, `scanf()` and the `main()` as a function... but what does this actually mean?



# FUNCTIONS

**FINALLY!**

- 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

## WHAT DO WE NEED TO KNOW?

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

**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;  
}
```

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

# FUNCTIONS

## WHAT DOES IT LOOK LIKE VISUALLY?

```
1 // Demonstrating functions with the program we created last week:
2 // Dice checker program, which checks whether the sum of the dice
3 // is equal to the target number. You win the game if it is.
4
5 // Sasha Vassar, Week 3 Lecture 5
6
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 main (void) {
18     int die_one;
19     int die_two;
20     int sum = 0;
21     int scanf_return;
22
23
24     printf("Enter the dice rolls (two numbers): ");
25     // scanf returns an int
26     scanf_return = scanf("%d %d", &die_one, &die_two);
27
28     // Since we have learnt the shorthand way of doing this...
29     // Can we include our scanf() function - because it returns an int! -
30     // inside our if statement?
31
32     if (scanf_return != 2) {
33         printf("You have not entered two numbers for the dice!\n");
34         printf("The program will now exit!\n");
35         return 1;
36     }
37
38     sum = die_one + die_two;
39
40     if (sum > TARGET) {
41         printf("The sum of the dice is greater than the target number!\n");
42     } else if (sum < TARGET) {
43         printf("The sum of the dice is less than the target number!\n");
44     } else {
45         printf("You have guessed the target number!\n");
46     }
47
48     return 0;
49 }
```

# FUNCTIONS

## TAKING THE ADDITION OUT AS A SEPARATE STEP

```
37  
38 sum = die_one + die_two;  
39
```

If I take the addition step out of the main function and move to its own step (function):

1. What do I have to give this function for it to work?
2. What should I name my function so that I know what to call it each time I need it?
3. What does this function have to return, so I can keep working?

```
51 int add (int die_one, int die_two){  
52     int sum;  
53     sum = die_one + die_two;  
54     return sum;  
55 }  
56  
57 // OR A NEATER WAY TO DO IT WITH NO EXTRA VARIABLES:  
58  
59 int add (int die_one, int die_two){  
60     return die_one + die_two; // RETURN THE RESULT OF ADDITION  
61 }
```

Can always call this from the main by referring to the function by name and saying what inputs I am giving this function, i.e. call by: `add(die_one, die_two);`

# FUNCTIONS

## TAKING THE COMPARISON OUT AS A SEPARATE STEP

Call function when I need it by name:

**comparison(sum)**

If I take the comparison step out of the main function and move to its own step (function):

1. What do I have to give this function for it to work?
2. What should I name my function so that I know what to call it each time I need it?
3. What does this function have to return, so I can keep working?

```
40  if (sum > TARGET) {
41      printf("The sum of the dice is greater than the target number!\n");
42  } else if (sum < TARGET) {
43      printf("The sum of the dice is less than the target number!\n");
44  } else {
45      printf("You have guessed the target number!\n");
46  }
```

```
63 void comparison(int sum) {
64     if (sum > TARGET) {
65         printf("The sum of the dice is greater than the target number!\n");
66     } else if (sum < TARGET) {
67         printf("The sum of the dice is less than the target number!\n");
68     } else {
69         printf("You have guessed the target number!\n");
70     }
71 }
```



# FUNCTIONS

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

So now we have moved two steps out to be their 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 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://www.menti.com/yzp3c6wtnp>



# 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](mailto:cs1511@cse.unsw.edu.au)