

### **COMP1511 PROGRAMMING FUNDAMENTALS**

# Lecture 5

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





COMP1511 Programming Fundamentals

WE...

- Played with decision making (IF statements)
- Looped (WHILE)
- Talked about scanf() in more detail
- Discovered structs

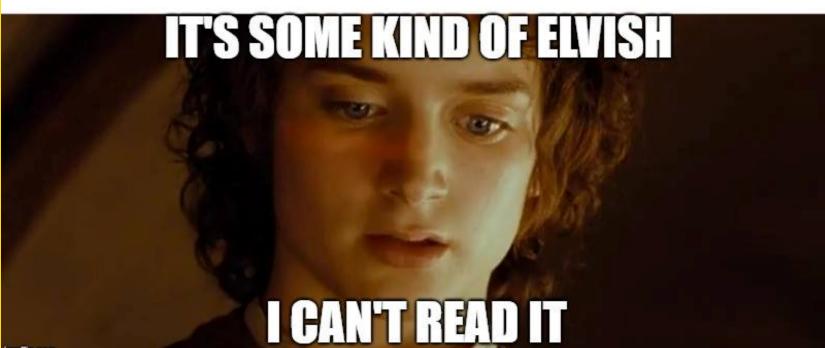
# IN WEEK 2,



COMP1511 Programming Fundamentals

- Style
- Functions

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





## WHERE IS THE CODE? LIVE LECTURE CODE CAN BE FOUND HERE:

https://cgi.cse.unsw.edu.au/~cs1511/21T3/live/Week03/

## WHAT IS STYLE? WHY STYLE? **IS IT WORTH IT?**

- eyes

#### The code we write is for human

• 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?**



- Names of variables and functions
- Structuring your code
  - Nesting
  - Repetition
- 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

Indentation and Bracketing

Comments where comments need

# **BAD STYLE**

- style...
  - bad\_style.c
- 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?

COMP1511 Programming Fundamentals

#### Let's have a look at some bad

## WHAT ARE SOME SPECIFIC ISSUES THAT YOU CAN SEE?

COMP1511 Programming Fundamentals

## **KEEP IT CLEAN AS** YOU GO - MUCH **EASIER THAN MAKING YOUR** WAY THROUGH A **DUMPSTER FIRE OF** MESS

COMP1511 Programming Fundamentals

- needed
- Name your variables based on what that variable is there to do
- {}:

  - 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?

• Write comments where they are

• In your block of code surrounded by

## **1511 STYLE GUIDE**

- 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/~cs1511/21T3/resources/style\_guide.html

Often different organisations you

- work for, will have their own style
- guides, however, the basics remain

### SOME NEAT SHORTHAND

### INCREMENTING AND REPEATING OPERATIONS



[count = count + 1] Increment count by 1

## count--;

[count = count - 1] Decrement count by 1

## count+=5;

[count = count + 5] Increment count by 5

## count-=5;

[count = count - 5] Decrement count by 5

count\*=5;

[count = count \* 5] Multiply count by 5

### **OTHER NEAT** SHORTHAND

### **ASKING QUESTIONS INSIDE OUR CONDITION OR RETURNING AN OPERATION**

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

int scanf\_return; scanf\_return = scanf("%d", &size); if (scanf\_return !=1)

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

### **THIS IS PERHAPS** WHERE THINGS **START TO GET A BIT HARDER**



- and ask us lots of questions!
- practice

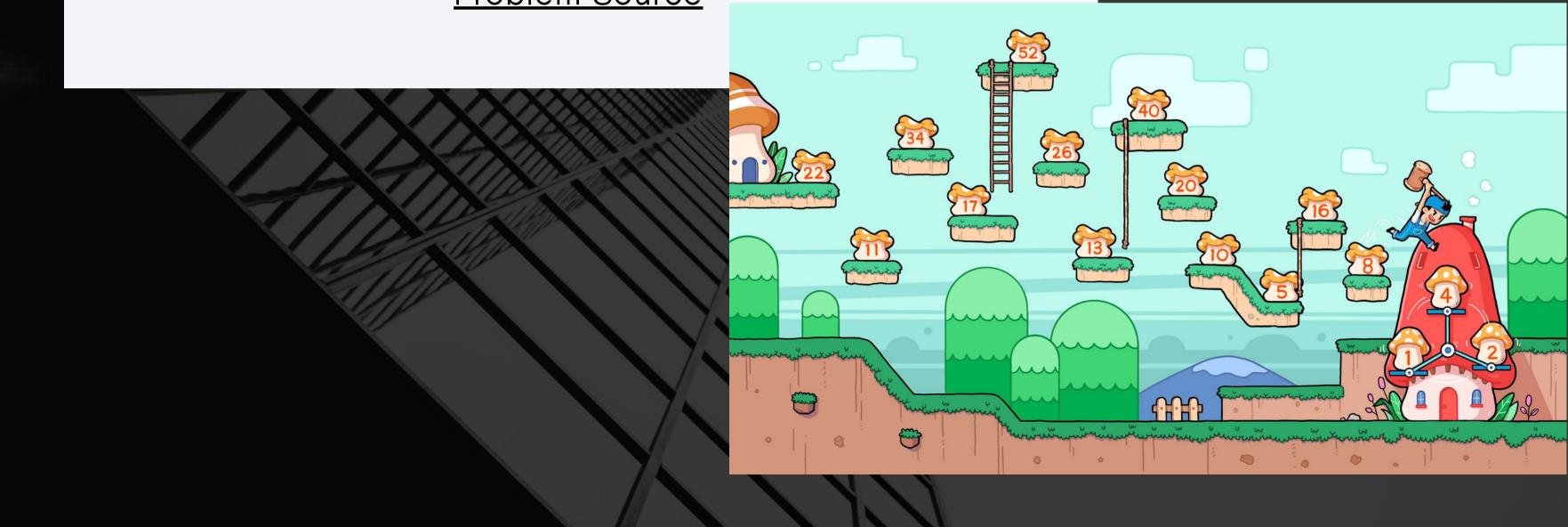
• If you do not understand something, do not panic! It is perfectly normal to not understand a concept the first time it is explained to you - try and read over some information again, ask questions in the tutorial and the lab - we are here to help you and to make sure that you are comfortable with the 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

• Remember learning is hard and takes time Solving problems is hard and needs

### **BREAK TIME (5 MINUTES)**

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? <u>Problem Source</u>



### **FINALLY I CAN STOP FEELING BAD EVERY TIME I MENTION FUNCTIONS :)**

- A function is a way to break down our
  - - operation

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

codes into smaller functional bits

Each function performs some sort of

 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

# WHAT DOES IT LOOK LIKE VISUALLY?

function\_demo.c

```
1 // Demonstrating the use of functions with code
 2 // from last week
 3 // Sasha Vassar, Week 2 Lecture 3
 5 #include <stdio.h>
 6
 7//1. Scan in numbers
 8 //2. Check for error on scanning
9//3. Add the numbers
10 //4. Compare to the sum for number of digits
11
12 int main (void) {
13
14
      int sum;
      int number_one;
15
      int number two;
16
17
18
      //1. Scan in numbers
      printf("Please enter two numbers: ");
19
20
21
      //2. Check for error on scanning
22
      if (scanf("%d %d", &number one, &number two) != 2) {
          printf("Error, a number was not scanned in\n");
23
24
          return 1;
25
26
27
      //3. Add the numbers
28
      sum = number one + number two;
29
30
      //4. Perform the comparison of sum to see number of digits
31
      if (-10 < sum \&\& sum < 10) {
32
          printf("%d has 1 digit\n", sum);
      } else if ((10 <= sum && sum < 100) || (-10 >= sum && sum > -100)) {
33
          printf("%d has 2 digits.\n", sum);
34
35
      } else if (sum >= 100 || sum <= -100) {</pre>
          printf("%d has more than 2 digits.\n", sum);
36
37
38
     return 0;
39 }
40
```

### **TAKING THE ADDITION OUT AS A SEPARATE STEP**

function\_demo.c

//3. Add the numbers sum = number one + number two;

```
Move step from main to a function, I 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(number_one, number_two);
```

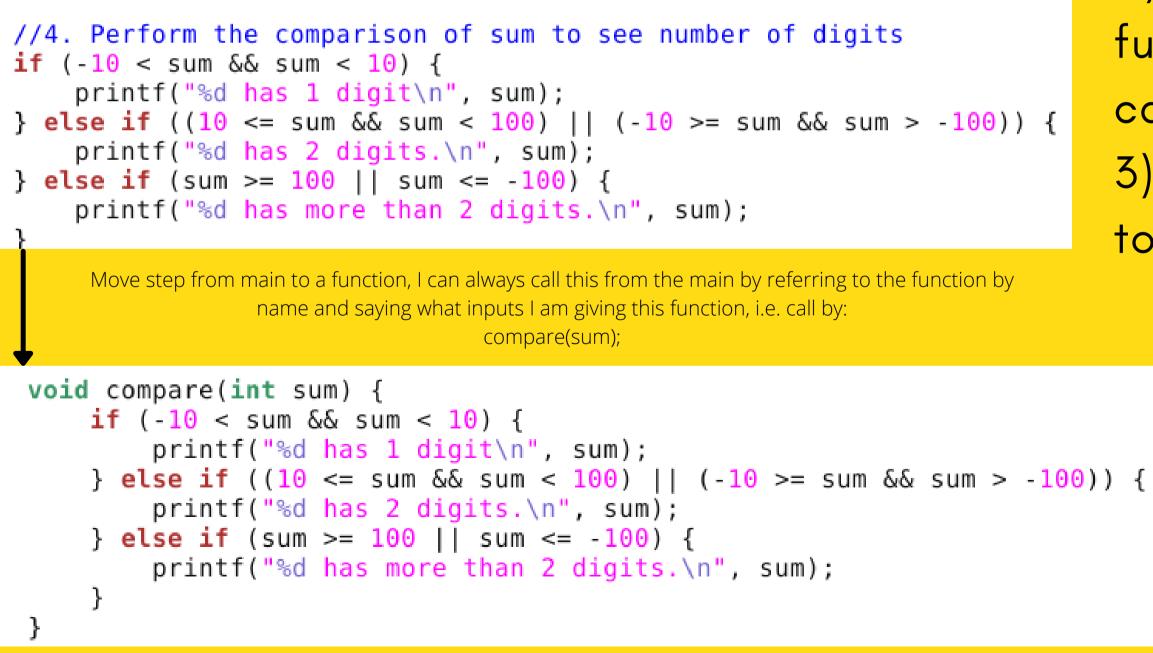
```
int add (int num one, int num two) {
    int sum;
    sum = number one + number two;
    return sum;
// OR A NEATER WAY TO DO IT WITH NO EXTRA VARIABLES
int add (int num one, int num two) {
    return number one + number two;
                                     //return the result of the addition straight away
```

need it?

- 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
- 3) What does this function have to
- return, so I can keep working?

### TAKING THE COMPARISON OUT AS A SEPARATE STEP

#### function\_demo.c



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?

### **TELLING C I HAVE SOME FUNCTIONS THAT I WANT TO USE**

function\_demo.c

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 num\_one, int num\_two) { And a function to compare: void compare (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?

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

function\_demo.c

#include <stdio.h>

```
//1. Scan in numbers
//2. Check for error on scanning
//3. Add the numbers
//4. Compare to the sum for number of digits
int add (int num_one, int num_two);
void compare(int sum);
int main (void) {
    int sum;
    int number_one;
    int number two;
```

We let C know in the very beginning <u>before main</u> about each function that we will use, but creating a function prototype. All it is is a very basic definition of the function to let C know those functions are included somewhere in this file! So for our add and compare functions:

int add (int num\_one, int num\_two);
void compare (int sum);

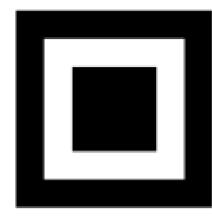
It is like declaring a variable, but I am declaring a function – note the semi colon at the end of each statement!

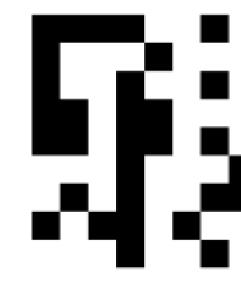
## **FEEDBACK?**

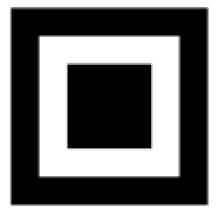
### PLEASE LET ME KNOW ANY FEEDBACK FROM TODAY'S LECTURE!

# www.menti.com

Code: 1356 5312









## WHAT DID WE LEARN TODAY?



bad\_style.c

### FUNCTIONS (BREAKING DOWN THE PROBLEM INTO ACTIONABLE STEPS)

function\_demo.c

### ANY QUESTIONS? DON'T FORGET YOU CAN ALWAYS EMAIL US ON CS1511@CSE.UNSW.EDU.AU FOR ANY ADMIN QUESTIONS

PLEASE ASK IN THE FORUM FOR CONTENT RELATED QUESTIONS

