LECTURE 5

FuncTIONS!
Classically stylish
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
FUNCTIONS

STYLE
WHERE IS THE CODE?

Live lecture code can be found here:

HTTPS://CGI.CSE.UNSW.EDU.AU/~CS1511/23T1/LIVE/WEEK03/
So far, you have heard me refer to `printf()`, `scanf()` and the `main()` as a function... but what does this actually mean?

- You have heard me rattle on about procedures
- Let’s take a quick step to now talk about what these actually are :)
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
```c
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.
Demonstrating the use of functions with a program that takes in two die rolls and checks whether the sum of the dice is equal to the target number - you win if that is the case!

Extending the problem: Otherwise you can roll again...

How can we break this problem down?

- Ask the user for the result of rolls (printf)
- Scan in the two dice (scanf)
- Add the numbers together (+)
- Check the sum against the target number (#define)
- Output the result against target number (printf)
 FUNCTIONS

LET'S WRITE SOME CODE AND DECOMPOSE IT INTO FUNCTIONS

dice_roll_functions.c

- I will show the next five slides at the lecture/during the lecture (so that we can discuss the code that we are writing.
  - Lecture slides will be updated to include the code that we wrote in the lecture and these slides post lecture
#include <stdio.h>

#define TARGET 9

int main (void) {
    int die_one;
    int die_two;
    int sum = 0;
    int scanf_return;

    printf("Enter the dice rolls (two numbers): ");
    // scanf returns an int
    scanf_return = scanf("%d %d", &die_one, &die_two);

    // Since we have learnt the shorthand way of doing this...
    // Can we include our scanf() function - because it returns an int! -
    // inside our if statement?
    if (scanf_return != 2) {
        printf("You have not entered two numbers for the dice!\n" "The program will now exit\n");
        return 1;
    }

    sum = die_one + die_two;

    if (sum > TARGET) {
        printf("The sum of the dice is greater than the target number!\n");
    } else if (sum < TARGET) {
        printf("The sum of the dice is less than the target number!\n");
    } else {
        printf("You have guessed the target number!\n");
    }

    return 0;
}
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?

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:

```c
add(die_one, die_two);
```
If I take the comparison step out of the main function and move it 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?

Call function when I need it by name: `comparison(sum)`
So now we have moved two steps out to be their own functions. We now have a function to add two numbers together:

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

And a function to compare:

```c
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?
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:

```c
int add (int die_one, int die_two);
void comparison (int sum);
```

```c
#include <stdio.h>

// 1. Scan in the two dice (scanf())
// 2. Add the numbers together (+)
// 3. Check the sum against the target number (#define)
// 4. Output greater or less than (printf())
#define TARGET 9

int add(int die_one, int die_two);
void comparison(int sum);

int main (void) {
```
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?

BACK TO IT!

KAHOOT TIME!

Let’s Kahoot....
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
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?

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

KEEP IT CLEAN AS YOU GO

MUCH EASIER THAN MAKING YOUR WAY THROUGH A DUMPSTER FIRE OF MESS
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/~cs1511/23T1/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;
  ```
Remember when we checked that scanf() returned something by doing this:

```c
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

```c
if (scanf("%d", &size) != 1)
```
You have so far learnt about looping with the keyword: while
Some of you have asked about looping with a for loop
They are very similar! Some people have preference for which one they like to use more, my rule of thumb is
- while loops when I do not know the number of iterations ahead of time
- for loops when I do know the number of iterations ahead of time
WHAT ABOUT FOR LOOPS?

For example:

- FOR loop to iterate over an array because I know how big my array is (on Wednesday!)
- FOR loop when you know the loop should execute n times
- WHILE loop for reading a file into a variable - we will not do this in 1511!
- WHILE loop when asking for user input.
- WHILE loop when the increment value is not a standard increment

In the end it is your choice, so don’t get stressed about which one is right!
FOR LOOP STRUCTURE

**initialisation:** executed before loop begins

**expression:** Evaluated before each iteration, exits when false

**increment:** executed at the end of each iteration

```c
for (int count = 0; count < 10; count++) {
    do something;
}
```
WHILE VERSUS FOR LOOP

```java
1 // FOR LOOP
2 for (int i = 0; count < 10; count++) {
3     // do something
4 }
5
6 // VERSUS
7
8 // WHILE LOOP (doing the same thing!)
9 int count = 0;
10 while (count < 10) {
11     // do something
12     count++;
13 }
```
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
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://www.menti.com/alyb4q928a7m
WHAT DID WE LEARN TODAY?

FUNCTIONS
breaking down the problem into actionable steps
function_demo.c

STYLIN'
bad_style.c
CONTENT RELATED QUESTIONS
Check out the forum

ADMIN QUESTIONS
cs1511@unsw.edu.au