COMP1511/1911 Programming Fundamentals

Week 3
Lecture 1

Code of Conduct

- All students have a right to learn
  - You will NOT hinder anyone else's learning
- Anything connected to COMP1511, including social media, will follow respectful behaviour
  - No discrimination of any kind
  - No inappropriate behaviour
    - No harassment, bullying, aggression, or sexual harassment
  - Full respect for the privacy of others
- We may simply turn off the live chat
  - This will only impede you and your future learning

Previously On...

- if statements
  - Making decisions
- while loops
  - Repeating code
- structs
  - Grouping variables
- enums
  - Counting with names

Today

- Functions
  - Breaking down code into smaller chunks
  - Another way of not repeating code
- Style
  - Looking good and cool

Where is the Code?

- Lecture code is updated every hour on the hour
- Live lecture code can be found here
  - https://cgi.cse.unsw.edu.au/~cs1511/26T2/

Why Bother?

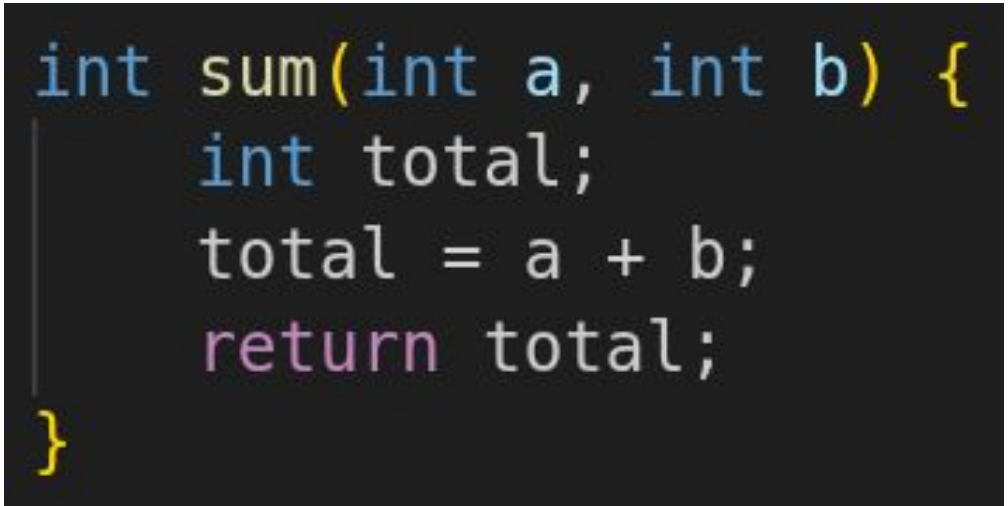
- Writing everything in main gets messy quickly
  - Breaking our code into problems makes it more readable, and easier for us (and others) to comprehend
- Style is important when working in teams
  - Consistent and comprehensible code makes collaboration cohesive

Functions

- A way to break code into smaller chunks
- Each function
  - Performs some sort of operation
  - Has inputs and outputs
    - Both may be empty, but that still counts
- We call our function from anywhere in the code, and then return its output to the spot we called it from
- We've been using functions already!
  - printf
  - scanf

Functions

- Consider the following function
  - What does it do?
  - What type does it return?
  - What values does it take?



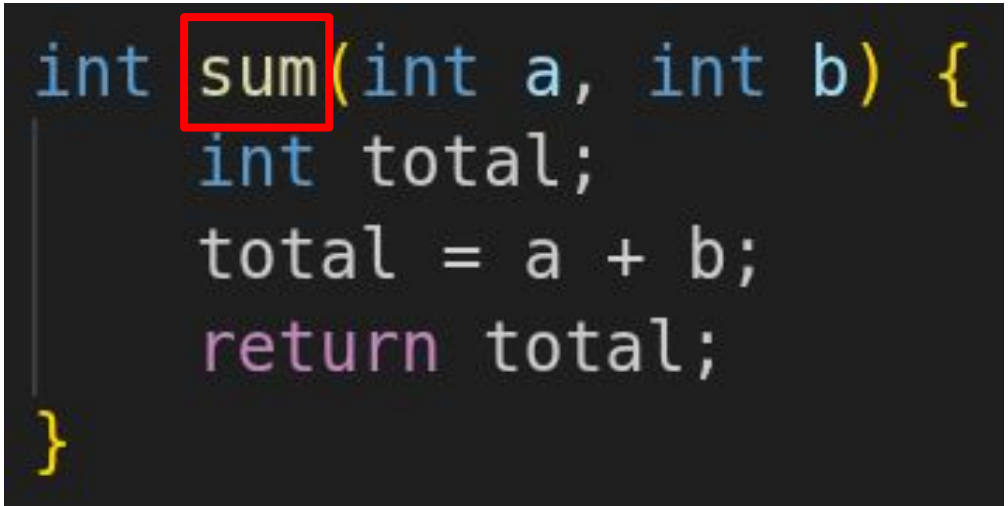
Functions

- Return type
  - Tells us what type the function will return

```
int sum(int a, int b) {
    int total;
    total = a + b;
    return total;
}
```

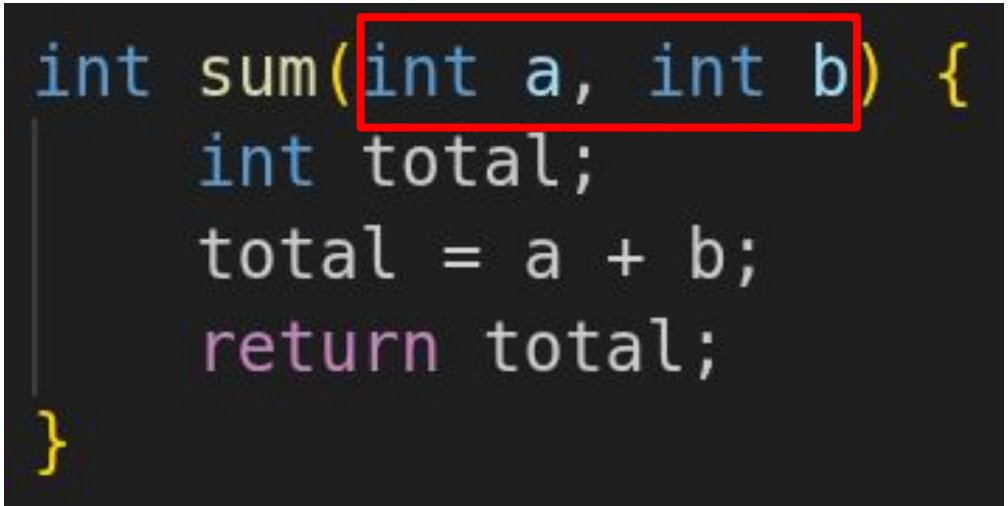
Functions

- Function name
  - What we type when we want to call our function
    - int value = sum(12, 13);



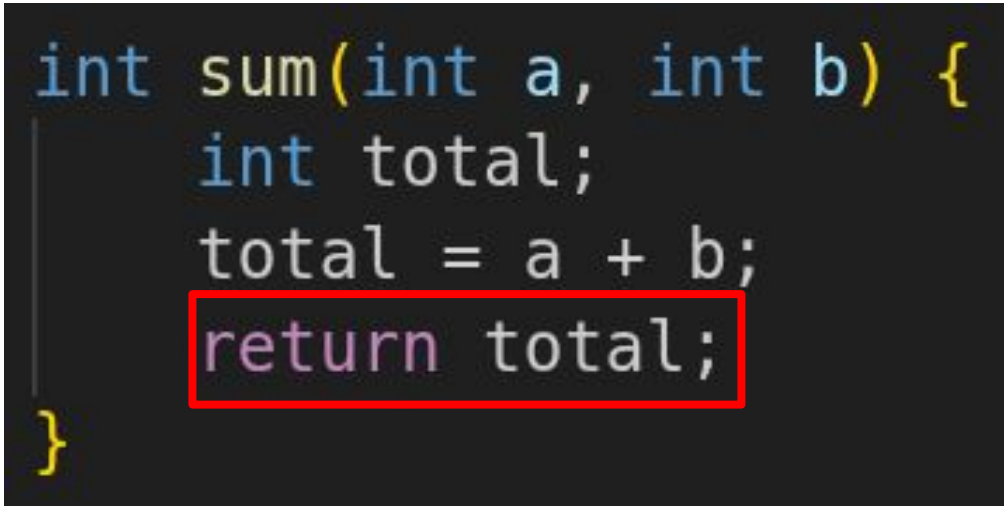
Functions

- Inputs/arguments
  - Values our functions take
    - Have specified types



Functions

- Return statement
  - How we return things from our function, to our code



Function Prototypes

- When using functions, we must define them before main
  - This is the a basic definition to let C know these functions exist in the file somewhere
- C reads from top to bottom
  - If we call a function and it C doesn't know it exists, the code won't work!

```c
#include <stdio.h>

int sum(int a, int b);
int multiply(int a, int b);

int main(void) {
    int step_one;
    step_one = sum(12, 13);
    int step_two;
    step_two = multiply(step_one, step_one);
    printf("%d\n", step_two);
    return 0;
}

int sum(int a, int b) {
    int total;
    total = a + b;
    return total;
}

int multiply(int a, int b) {
    int total;
    total = a * b;
    return total;
}
```

Function Scope

- Variables in a function only exist in that function, unless passed elsewhere
  - So we can reuse variable names between functions if it fits
- This is a very difficult concept that a lot of people struggle with! So don't panic!

```c
#include <stdio.h>

int sum(int a, int b);
int multiply(int a, int b);

int main(void) {
    int step_one;
    step_one = sum(12, 13);
    int step_two;
    step_two = multiply(step_one, step_one);
    printf("%d\n", step_two);
    return 0;
}

int sum(int a, int b) {
    int total;
    total = a + b;
    return total;
}

int multiply(int a, int b) {
    int total;
    total = a * b;
    return total;
}
```

Dice Roll Example

- 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…
- What constants do we need?
- What functions do we need?

Dice Roll Examples?

- What functions do we need?
  - We want to print some kind of intro to the game
  - We want to play the round
  - We want to figure out if someone has won
    - To do this, we need to calculate the total

Dice Roll Examples?

- What functions do we need?
  - We want to print some kind of intro to the game
  - We want to play the round
  - We want to figure out if someone has won
    - To do this, we need to calculate the total

```
void play_intro(void);
void play_round(void);
int calculcate_sum(int a, int b);
int is_winner(int sum);
```

The Full Code

- Wow

```c
#include <stdio.h>

void play_intro(void);
void play_round(void);
int calculcate_sum(int a, int b);
int is_winner(int sum);

int main(void) {
    int sum = 10;
    sum++;
    play_intro();
    play_round();
    return 0;
}

void play_intro(void) {
    printf("Welcome to my epic dice game\n");
    printf("If you roll a total of 6 or 7, you win the game!\n");
    return;
}

void play_round(void) {
    int die_1;
    int die_2;
    printf("Enter the roll of die 1: ");
    scanf("%d", &die_1);
    printf("Enter the roll of die 2: ");
    scanf("%d", &die_2);
    int sum = calculcate_sum(die_1, die_2);
    int won = is_winner(sum);
    if (won == 1) {
        printf("Congratulations! Go away now\n");
    } else {
        printf("Do you want to play again?\n");
    }
    return;
}

int calculcate_sum(int a, int b) {
    int total;
    total = a + b;
    return total;
}

int is_winner(int sum) {
    if (sum == 6 || sum == 7) {
        printf("You win!\n");
        return 1;
    } else {
        printf("Boo-hoo, loser\n");
        return 0;
    }
}
```

Break time!

...probably

Let's Do Another Example!

- Build a calculator app. The calculator can add, subtract, multiply, divide, and square. The user enters a number (1-5) to pick their function, or 6 to quit the calculator.
- What functions do we need?

Calculator Example

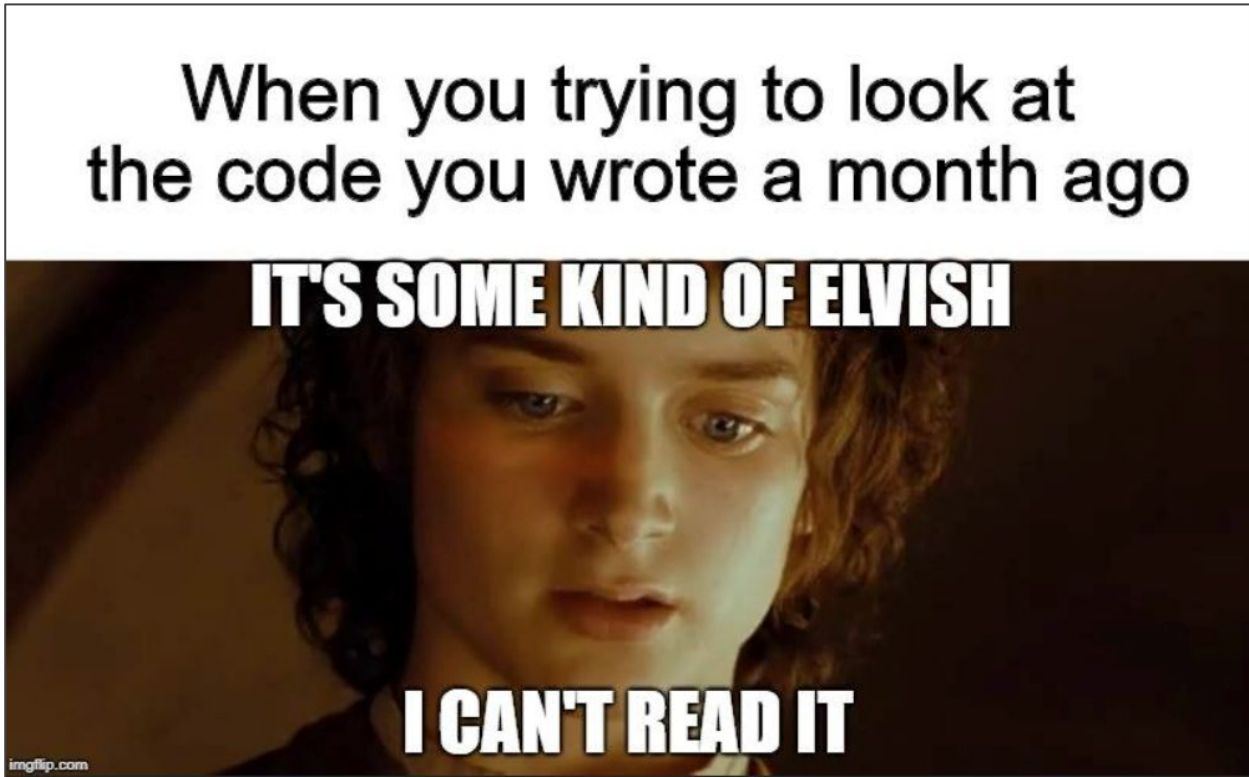
- What functions do we need?
  - A function to print the welcome
  - A function to sum
  - A function to subtract
  - A function to multiply
  - A function to divide
  - A function to square

The Full Code Again!

- Henry will update this slide after the lecture as well!

Style

- What is style?
- Why is style?



Is Style Worth It?

- The code we write is for human eyes (or eye in my case)
- We want to make our code
  - Easier to read
  - Easier to understand
- Neat code ensures
  - Fewer mistakes
  - Lower probability of mistakes
  - Faster development time
- Coding should ALWAYS be done in style

What is Good Style?

- Indentation and bracketing
- Names of variables and functions
- Structuring your code
  - Nesting
  - Repetition
- Comments where comments need to be
- Consistency
- I should be able to understand your code from the structure and variable names alone

Bad Style

- Let’s look at some bad style

Bad Style

- This hurts me to look at
  - It is perfectly functional code!
    - Who wants to work on it?
- In fact, let's work on it and tidy it up
  - Where can we start?
- Start from the smallest things that are easy to do straight away
  - What can you do next?

```c
#include<stdio.h>
#define T 7
int main(){

int a,b,c;char d='y';

printf("dice game\n");

while(d=='y'){
printf("1:");
scanf("%d",&a);
printf("2:");
scanf("%d",&b);

c=a+b;

if(c==T)
printf("win\n");
else
printf("lose\n");

printf("again?");
scanf(" %c",&d);}
printf("bye\n");
}
```

Keep It Clean as You Go

- Write comments where they are needed
  - There is such thing as over documentation
- Name your variables based on what the variable is there to do
- In your blocks of code (surrounded by {})
  - Indent 4 spaces
  - Line up closing brackets with the statement that opened them vertically
- Consistency in spacing
- Watch out for nested ifs
  - Can it be done more efficiently?

1511 Style Guide

- Different organisaiton will have different style guides
  - The basics remain the same
- Your assignments will have style marks attached to them
- We have a style guide in 1511, please use it and familiarise yourself with it
  - Building good habits early makes them second nature
    - Important when things get tough
- Style Guide Here

Some Neat Shorthand

- Increment count by 1
  - `count = count + 1;`
    `count++;`
- Decrement count by 1
  - `count = count - 1;`
    `count --;`

Some Neat Shorthand

- Increment count by 5
  - `count = count + 5;`
    `count += 5;`
- Decrement count by 5
  - `count = count - 5;`
    `count -= 5;`
- Multiple count by 5
  - `count = count * 5;`
    `count *= 5;`

Other Neat Shorthand

- You can call functions in your if statements of while loops, as long as the function will return something to check!

```c
#include <stdio.h>

int sum(int a, int b);

int main(void) {
    if (sum(13, 12) == 25) {
        printf("Woohoo!\n");
    }
    return 0;
}

int sum(int a, int b) {
    int total;
    total = a + b;
    return total;
}
```

For Loops

- So far, you have learnt looping with while loops
- Some have asked about looping with for loops
  - They are very similar!
- My rule of thumb
  - If I know how many times I want to do something, use a for loop
  - If I’m uncertain, use a while loop
- while loops can do anything for loops can
  - for loops cannot do anything while loops can

For Loops

- Example
  - for loop to iterate over an array that I know the size of
  - for loop if I know the loop should be executed n times
  - while loop to get user input
  - while loop when the increment value is not standard (e.g not 1)
- They’re very similar, and accomplish the same goal, so don’t stress too hard about it!

For Loop Structure

- What does this for loop do?
  - How many times does it do it?

```
for (int i = 0; i < 10; i++) {
    printf("Iteration %d complete!\n", i);
}
```

For Loop Structure

- Initialisation
  - Happens before the loop
  - Creates variable with desired value

```
for (int i = 0; i < 10; i++) {
    printf("Iteration %d complete!\n", i);
}
```

For Loop Structure

- Expression
  - Evaluates before each iteration
  - Exits when false

```
for (int i = 0; i < 10; i++) {
    printf("Iteration %d complete!\n", i);
}
```

For Loop Structure

- Increment
  - Executed at the end of each iteration

```
for (int i = 0; i < 10; i++) {
    printf("Iteration %d complete!\n", i);
}
```

While vs For

- These two loops accomplish the exact same task

```c
for (int i = 0; i < 10; i++) {
    printf("Iteration %d complete!\n", i);
}

int i = 0;
while (i < 10) {
    printf("Iteration %d complete!\n", i);
    i++;
}
```

This is Where People Really Start Struggling

- 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
    - Ask us lots of questions!
- Remember learning is hard and takes time
- Solving problems is hard and needs practice

What We Learnt

- Functions
  - How to break our code into smaller blocks
- Style
  - How to make our code readable
- For loops
  - An alternative to while loops

Reach Out

- Check the course forum for questions!
  - Post there too!
- Admin questions
  - cs1511@unsw.edu.au

Thank You

Questions?