COMP1511 - Programming Fundamentals

Week 3 - Lecture 5

What did we learn last week?

- **`if` statements** - branching code
- **Problem solving** - thinking carefully while programming
- **`while` loops** - repeating code

What are we covering today?

Code Style

- What is Code Style? Why does it matter?

Code Reviews

- What is a Code Review?
- What can we learn from Code Reviews?

Functions

- An introduction to what a function is
- How we use functions in C

While Loops Recap

What do we know about While Loops?

- They have a specific syntax
- They test an expression and run repeatedly while it’s true
- We can make them stop after a specific number of iterations
- We can make them stop after a certain condition is met
- We can run any other code inside a while loop

Will it ever stop? I don’t know…

It’s easy to make it start, but make sure you can stop it!

- Create every loop with the idea of how it stops
- Let’s review how we stop loops

While Loop with a Loop Counter

How to make a loop run an exact number of times

```
// an integer outside the loop
int i = 0;

while (i < 10) {
    // Code in here will run 10 times

    i = i + 1;
}
// When i hits 10 and the loop's test fails
// the program will exit the loop
```

Using a Sentinel Variable with While Loops

A sentinel is a variable we use to intentionally exit a while loop

```
// an integer outside the loop
int endLoop = 0;

// The loop will exit if it reads an odd number
while (endLoop == 0) {
    int inputNumber;
    scanf("%d", &inputNumber);
    if (inputNumber % 2 == 0) {
        printf("Number is even.\n");
    } else {
        printf("Number is odd.\n");
        endLoop = 1;
    }
}
```

Code Style

Why do we write code for humans?

- Easier to read
- Easier to understand
- Less mistakes
- Faster overall development time

Good Coding Practices

What is good style?

- Indentation and Bracketing
- Names of variables and functions
- Repetition (or not) of code
- Clear comments
- Consistency

The easier it is to read and understand, the less mistakes we'll make

Poor Code Style

Can we work with code that's hard to read?

- I'd like to show you something I prepared earlier . . .
- CodeStyleBad.c is functionally our Dice Checking program

Let's have a look at the code . . .

What went wrong?

We want more than: “Oh wow, that’s a mess”

What are the specific improvements that can make this better?

In the face of disaster, keep a clear head and focus on what can be fixed

Specific Issues

- Header comment doesn’t show the program’s intentions
- No blank lines separating different components
- Multiple expressions on the same line
- Inconsistent indenting
- Inconsistent spacing
- Variable names don’t make any sense
- Comments don’t mean anything
- Inconsistent bracketing of if statements
- Bracketing is not indented
- Inconsistent structure of identical code blocks
- The easter egg - there’s actually incorrect code also!

Keeping your house (code) clean

Regular care is always less work than a big cleanout

- Write comments before code
- Name your variables before you use them
- `{` everything inside gets indented 4 spaces
- `}` line up your closing brackets vertically with the line that opened them
- One expression per line
- Maintain consistency in spacing

Comments before code

Comments before code. It’s like planning ahead

- Making plans with comments
- You can fill them out with correct code later
- Some of these comments can stay even after you've written the code

```
// Checking against the target value
if () {
    // success
} else if () {
    // tie
} else {
    // failure (all other possibilities)
}
```

Naming Variables

Variable names are for humans

- Can you describe what a variable is in a word or two?
- If your lab partner was to read this name, would it make sense?
- Does it distinguish it well against the other variables?

Indentation

A common convention is to use 4 spaces for indentation

```
int main (void) {
    // everything in here is indented 4 spaces
    int total = 5;
    if (total > 10) {
        // everything in here is indented 4 more
        total = 10;
    }
    // this closing curly bracket lines up
    // vertically with the if statement
    // that opened it
}
// this curly bracket lines up vertically
// with the main function that opened it
```

One expression per line

Any single expression that runs should have its own line

```
int main (void) {
    // NOT LIKE THIS!
    int numOne; int numTwo;
    numOne = 25; numTwo = numOne + 10;
    if (numOne < numTwo) { numOne = numTwo; }
}
```

```
int main (void) {
    // Like this :)
    int numOne;
    int numTwo;
    numOne = 25;
    numTwo = numOne + 10;
    if (numOne < numTwo) {
        numOne = numTwo;
    }
}
```

Spacing

Operators need space to be easily read

```
int main (void) {
    // NOT LIKE THIS!
    int a;
    int b;
    int total=0;
    if(a<b&&b>=15){
        total=a+b;
    }
}
```

```
int main (void) {
    // Like this :)
    int a;
    int b;
    int total = 0;
    if (a < b && b >= 15) {
        total = a + b;
    }

}
```

More Information about Coding Style

- The course webpage has a Style Guide
- Wherever you end up coding, there will be different styles
- Our style is only one of them, but a good place to start!

Your assignments have coding style marks (more on this when they release)

The Exam has some style marks also

Break Time

Code Style isn't just to make it look nice

- Reduces errors later in development
- Makes it easier to test and modify
- Overall, speeds up development
- Makes your co-workers hate you less

Weekly Tests

Self Invigilated Weekly Tests start this week

- A mini exam you run yourself
- The detailed rules are in the test itself
- Releases on Thursday and you will have one week to complete it

- Use it as a way to test your progress so far
- Great practice for coding with time pressure and limited resources (exams or job interviews)

Code Review

What is a code review?

- Having other coders look over your code
- Having an active discussion about the code

- Automated testing can test functionality, but not necessarily usability
- Humans can help you improve as a human!

- Similar to proof-reading a document
- Super valuable to discuss different approaches to the same problem

Why do we review code?

As the code writer

- Get feedback on how easy it is to understand our code
- Hear about other people's ideas on solving the same problem

As the code reviewer

- Get to see how someone else writes code
- Learn more about different ways to solve problems

Different ways to review code

Pair Programming

- Lab partners actively discussing solutions
- Live reviewing and discussion while in development

More formal review

- Finish a section of code, then ask people to review it
- Sometimes in person, sometimes using software tools

How to do Pair Programming well

Also, how to learn the most from 1511 labs

- One person on the keyboard
  - Thinking about how to structure the C and syntax
- One person over the shoulder
  - Thinking about how to solve the problem
- Active discussion between the two of you as you go
- This means the code is constantly under review

Programming with others is one of the best ways to learn!

Conducting a Code Review

Reviewing a finished piece of code

- Reviewers will read the code and help with it
- Remember, we're judging the code, not the coder!
- We're all learning . . . this is not about picking at mistakes

Points to Discuss

- Where is it easy or hard to understand the code?
- What are the different possible ways the code can solve the problem?
- Any little issues we can help solve?

What not to do in a Code Review

These things will not help us learn better code:

- “You did this wrong”
- “Your code is bad”
- “Here are all the mistakes in this code”

We’re doing this to help ourselves and others learn more!

No judgement, only help!

What to do in a Code Review

How does one help someone else learn?

- Understand that it’s very hard to put your work up for review
- We’re not here to judge the code’s standard
- We’re here to help everyone learn more

- There is no single right way to solve a problem
- If your way and someone else’s way are different, you can both be right
- Try to learn from other styles of coding that you review

- Letting people know what you don’t understand is one of the most valuable things you can do in a code review

Next week's Tutorial will have a demo Code Review

Your tutor will do the first review so you can see what it's like

- After this, every code review will be lead by students
- You can also get together with other students to review your Lab work
- (just don't do it with Assignments!)

Functions

Let’s introduce at functions

- We’ve already been using some functions!
- **main** is a function
- **printf** and **scanf** are also functions

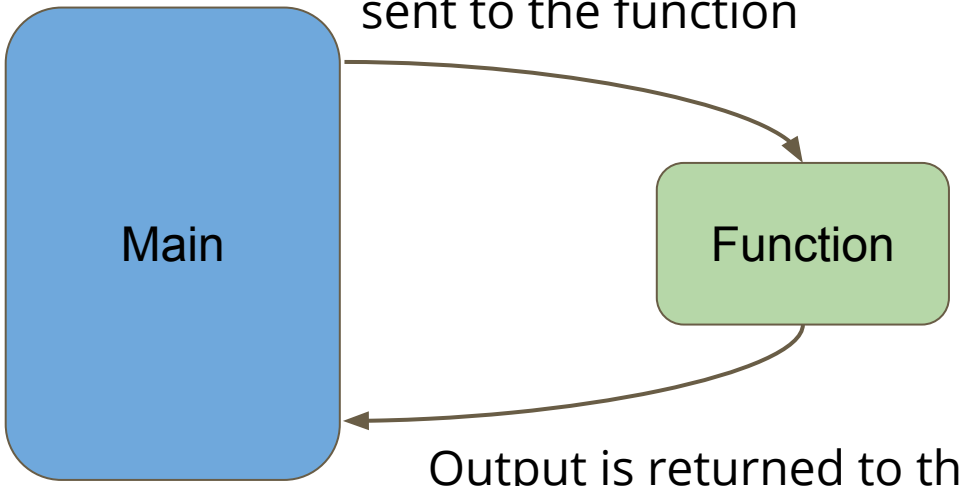
What is a function?

- A separate piece of code identified by a name
- It has inputs and an output
- If we “call” a function it will run the code in the function

Functions

How do they work?

Main calls the Function, asking it to run. Input is sent to the function



Function runs its code using the input

Output is returned to the Main. Main continues from where it was

Function Syntax

We write a function with (in order left to right):

- An output (known as the function's type)
- A name
- Zero or more input(s) (also known as function parameters)
- A body of code in curly brackets

```
// a function that adds two numbers together
int add (int a, int b) {
    return a + b;
}
```

Return

An important keyword in a function

- `return` will deliver the output of a function
- `return` will also stop the function running and return to where it was called from

How is a function used?

If a function already exists (like printf)

- We can use a function by calling it by name
- And providing it with input(s) of the correct type(s)

```
// using the add function
int main (void) {
    int firstNumber = 4;
    int secondNumber = 6;
    int total;

    total = add(firstNumber, secondNumber);
    return 0;
}
```

Compilers and Functions

How does our main know what our function is?

- A compiler will process our code, line by line, from top to bottom
- If it has seen something before, it will know its name

```
// An example using variables
int main (void) {
    // declaring a variable means it's usable later
    int number = 1;

    // this next section won't work because the compiler
    // doesn't know about otherNumber before it's used
    int total = number + otherNumber;
    int otherNumber = 5;
}
```

Functions and Declaration

We need to declare a function before it can be used

```
// a function can be declared without being fully
// written (defined) until later
int add (int a, int b);

int main (void) {
    int firstNumber = 4;
    int secondNumber = 6;
    int total = add(firstNumber, secondNumber);
    return 0;
}

// the function is defined here
int add (int a, int b) {
    return a + b;
}
```

Void Functions

We can also run functions that return no output

- We can use a void function if we don't need anything back from it
- The return keyword will be used without a value in a void function

```
// a function of type "void"
// It will not give anything back to whatever function
// called it, but it might still be of use to us
void add (int a, int b) {
    int total = a + b;
    printf("The total is %d", total);
}
```

What did we learn today?

Code Style

- Making your code understandable and reusable

Code Reviews

- Reviewing your's and other people's code can help you learn and share your skills

Functions

- Separating code to make it easier to read and reuse