COMP1511 - Programming Fundamentals

Term 1, 2019 - Lecture 5
Stream B

What did we learn last week?

- if statements - branching code
- Problem solving - thinking before coding
- Code Style - readability and prevention of errors
- While loops - repeating code

What are we covering today?

Code Review

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

While Loops

- Some recap of the syntax and what they do
- Another example of a while loop (combining with if statements)

Code Review

What is a code review?

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

- Auto-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 program, 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 <u>not</u> 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

Let’s do a mini Code Review

Here’s some code for us to review

- A small program from last week’s tutorial
- CodeReview.c
- Let’s have a look at it and see what we think . . .

What would we think about first?

Is it understandable?

- What don’t you understand about this code?
- What questions do you need to ask to get a grasp on what’s going on?

Can we help?

- Can we suggest any ways to change things to be more understandable?
- What would easily answer the questions above?

Some possibilities

- What's the overall purpose of the code?
- There's a big set of if statements, what is the problem that is being solved by that set of code?
- There's some interesting bracketing happening. Is that a different style from what I'm used to?
- What's the purpose of the "return 0" we're seeing in the if statements?
- That last printf . . . is that always going to run?

Deeper Analysis

Problem Solving in the code

- Has the code definitely solved all the problems expected of it?
- Is the code solving problems the same way you would have?
- Are there any consequences of the different ways this can be done?

Other options

- Is there other structure that would also solve the code?
- Is there any way we can make the code easier to understand?
- Do we want to make changes on behalf of the style guide?

Some Possibilities

- We think it’s solved all the problems (just need to check the spec)
- The returns are interesting . . . do they make it more efficient?
- Do they make it easier to read and understand?
- Do they make the program run more cleanly?
- Could we also do this with if/else and would that be easier to read?
- Do we think this would be easier with more comments explaining things?
- Is there a way to structure that final statement or comment it so that it’s understandable?
- Is the lack of return 0 at the end going to cause any problems?
- Do we want to reformat the bracketing to fit the class style guide?

Where else can this discussion lead?

We’ve barely scratched the surface

- Different reviewers will give you different perspectives
- This process is for both the reviewer and reviewee to learn from
- Expose yourself to different coding styles
- If you don’t understand something, this is the best time to ask (it helps the reviewee as well!)

Break Time

Let's take five minutes break

- Code reviews are ways to have human communication about code
- They expose us to other coding styles
- They lead to interesting discussions into problem solving
- They're a good way to learn different approaches

Problems by Parallel Studio

COMP1511 Resources

How to get help in COMP1511

- **Course Forum** is linked from the course webpage
- Questions are welcome there and will help other students

- **Course Help Sessions**
- Tuesday, Wednesday, Thursday 6-8pm, Bugle and Horn Labs
- Friday 3-5pm, Viola and Cello Labs
- You can go to these and ask about ANYTHING in the course

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 Wednesday and you will have one week to complete it

- Use it as a way to test your progress so far
- Great practice for the time pressure and limited resources in the exam

While Loops Continued

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 counter = 0;

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

    counter = counter + 1;
}
// When counter 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;
int inputNumber;

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

Let’s make another while loop program

I would like some programs that can print out lists of numbers

Let’s make a few programs that can do the following:

1. Multiples of 3 between 0 and 1000
2. Multiples of 3 between 0 and any number
3. Multiples of any number between 0 and another number
4. Multiples of any number between any two numbers

Approaching the Problem

If we need multiples of 3 between 0-1000, we’re doing a bit of repetitive work

First Approach

- Loop through all numbers between 0 and 1000
- Find which ones are divisible by 3 (hint: use mod!)
- Output them

Divisible by 3

```
int main (void) {
    // an integer outside the loop
    int loopCounter = 0;
    int finishNumber = 1000;

    // Loop from 0-1000, checking for mulitples of 3
    while (loopCounter <= finishNumber) {
        // output the number if it’s a multiple of 3
        if (loopCounter % 3 == 0) {
            printf(“%d\n”, loopCounter);
        }
        loopCounter = loopCounter + 1;
    }
    return 0;
}
```

Simple Solutions

It’s always good to start with something that works

- This is a simple solution that will get us our results
- Where’s the wasted effort? Can you see it?

Simple Solutions are a good start

Where’s the wasted effort?

- We’re looping 1000 times to output 333 numbers
- Can we do this by only looping 333 times?

Make thing, Make good.

It’s a simple philosophy that means:

> *“Make sure you do something basic that works, then improve it later.”*

Upgrading our work

```
int main (void) {
    // an integer outside the loop
    int loopCounter = 3; // start with the first multiple of 3
    int finishNumber = 1000;

    // Loop from 0-1000, only using multiples of 3
    while (loopCounter <= finishNumber) {
        printf("%d\n", loopCounter);
        // immediately jump to the next multiple of 3
        loopCounter = loopCounter + 3;
    }
    return 0;
}
```

Now with User Input

Using variables input by the user

- We can take a number for the maximum value of the range
- We can also take a number for our “divisor” number

Multiples with User Input

```
int main (void) {
    int divisor;          // remember the divisor
    int loopCounter;
    int finishNumber;

    // Take the user input numbers
    printf("Please enter the divisor:\n");
    scanf("%d", &divisor);
    printf("Please enter the highest possible value:\n");
    scanf("%d", &finishNumber);

    // Then loop . . . on the next slide
```

Multiples with User Input Continued

```
    // Get the loop ready by setting up the starting number
    loopCounter = divisor;

    // Loop from loopCounter to finishNumber,
    // printing multiples of divisor
    while (loopCounter <= finishNumber) {
        printf("%d\n", loopCounter);
        // immediately jump to the next multiple of divisor
        loopCounter = loopCounter + divisor;
    }
    return 0;
}
```

Code Review

What kind of questions might we ask?

- Does this code make sense?
- Does it explain itself by how it was written?
- Is this the only way to solve this problem? Is there another approach?
- Is this code solving the problem completely?

Can we step it up a notch?

What if our starting value isn't 0?

- What if our user gives us a lowest and highest value?
- How do we know what to start our loopCounter on?

Options (you can explore these yourself)

- We could return to the "brute force" method of checking all numbers
- We could use some hybrid approach . . . brute force until the first multiple, then use our optimised loopCounter

What did we learn today?

Code Reviews

- Reviewing your’s and other people’s code can be very valuable
- Reviewing helps you understand more code
- Being reviewed helps your code be presentable for humans

More Looping

- Start simple, basic problem solving
- Then expand to more interesting things
- Loops don’t just have to be in 1s, sometimes they can jump using larger numbers