COMP1511 - Programming Fundamentals

Term 2, 2019 - Lecture 5

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
- A more complex program with while loops

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 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 <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

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

- **Course Help Sessions**
- Tuesday, 12-2pm, Clavier Lab
- Friday 4-6pm, Mathews 103
- You can go to these and ask about ANYTHING in the course

- **Live Streams**
- We had our first one yesterday, it's linked via the course webpage also

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 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

```c
// 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;
    }
}
```

A more complex program

We’ve been learning about branching and looping code

Let’s make something that’s a little more complex than our previous examples

The following program:

I need a program that will show me all the different ways to roll two dice

If I pick a number, it will tell me all the ways those two dice can reach that total

It will also tell me what my odds are of rolling that number

Break it down

What components will we need?

- We need all possible values of the two dice
- We need all possible totals of adding them together
- Seems like we’re going to be looping through all the values of one die and adding them to all the values of the other die

Let’s start with this simple program then go for our bigger goals later

Code for all dice rolls

```c
int main (void) {
    int diceOneSize;
    int diceTwoSize;

    // User decides the two dice sizes
    printf("Please enter the size of the first die: ");
    scanf("%d", &diceOneSize);
    printf("Please enter the size of the second die: ");
    scanf("%d", &diceTwoSize);

    // Then loop through both dice
```

Code for all dice rolls continued

```
// loop through and see all the values that the two dice can roll
int die1 = 1;
while (die1 <= diceOneSize) {
    int die2 = 1;
    while (die2 <= diceTwoSize) {
        printf(
            "%d, %d. Total: %d\n",
            die1, die2, die1 + die2
        );
        die2++;
    }
    die1++;
}
```

Quick Pause for new C syntax

Incrementing just got a little easier

```
int die1 = 0;
int die2 = 0;

// The following two lines have the same effect
die1 = die1 + 1;
die2++;

// both variables now == 1
```

We can now see all possible dice rolls

- We have all possibilities listed
- We know all the totals
- We could also count how many times the dice were rolled

Let’s try now isolating a single target number

- Check the targets of the rolls and output only if they match our target value

Now with a target number

```c
int main (void) {
    int diceOneSize;
    int diceTwoSize;
    int targetValue;

    // User decides the two dice sizes and target
    printf("Please enter the size of the first die: ");
    scanf("%d", &diceOneSize);
    printf("Please enter the size of the second die: ");
    scanf("%d", &diceTwoSize);
    printf("Please enter the target value: ");
    scanf("%d", &targetValue);
```

Output the rolls that match the target

```c
// loop through and output rolls with totals that match the target
int die1 = 1;
while (die1 <= diceOneSize) {
    int die2 = 1;
    while (die2 <= diceTwoSize) {
        int total = die1 + die2;
        if (total == targetValue) {
            printf(
                "%d, %d. Total: %d\n",
                die1, die2, total
            );
        }
        die2++;
    }
    die1++;
}
```

Getting there!

We now have a program that can identify the correct rolls

- If we want the odds, we just compare the target rolls vs the rest
- If we count the number of rolls that added to the target value
- And we count the total number of rolls
- We can do some basic maths and divide the successful rolls by the total
- That should give us our chances of getting that number

How do we keep track of success vs failure?

We can count using ints

- We can keep a counting variable outside the loop
- This will increment only on successes
- We can either calculate or count our total
- Dividing them will give us the fraction chance of rolling our target number

Measuring Successes

Adding some variables to count results

```
int main (void) {
      int diceOneSize;
      int diceTwoSize;
      int targetValue;
      int numSuccesses = 0;
      int numRolls = 0;
```

Making sure our loop records results

```
// loop through and output rolls with totals that match the target
int die1 = 1;
while (die1 <= diceOneSize) {
    int die2 = 1;
    while (die2 <= diceTwoSize) {
        numRolls++;
        int total = die1 + die2;
        if (total == targetValue) {
            numSuccesses++;
            printf("%d, %d. Total: %d\n",
                    die1, die2, total);
        }
        die2++;
    }
    die1++;
}
```

Output our Percentage

```
// Calculate percentage chance of success
int percentage = numSuccesses/numRolls * 100;
printf("Percentage chance of getting your target number is: %d\n",
        percentage);
```

There’s an issue with the previous code …

Did you notice the issue?

- Our code outputs 0 percent a lot more than it should
- This is even after we know it’s counting the successes correctly

Integers do weird things with division in C

- After a division, the integers will throw away any fractions
- Since our "`numSuccesses/numRolls`" will always be between zero and 1
- Our result can only be the integers 0 or 1
- And anything less than 1 will end up having its fraction dropped!

Doubles to the rescue

Luckily we have a variable type that will store a fraction

- Result of a division will be a double if one of the variables in it is a double
- We need to change one of the variables in our division to a double

```
int main (void) {
    int diceOneSize;
    int diceTwoSize;
    int targetValue;
    int numSuccesses = 0;
    double numRolls = 0;
```

The Challenge . . . did we need to do all this work?

This program didn't actually need everything we did today

- There's a much simpler way to list the rolls that sum to a target number
- There's also a much simpler way to find the total number of rolls
- If we just use a bit more maths and less raw coding . . .

See what you can come up with!

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 Coding with Dice

- Loops within loops
- Variables inside and outside the loops
- Being careful with division and using doubles for fractions