COMP1511 - Programming Fundamentals

Term 1, 2019 - Lecture 6
Stream B

What did we learn on Tuesday

- Code Reviews
- Looking at each other's code
- Learning from what we can discuss about the code we've written

- More work on while loops
- Starting simple and building up solutions
- Remember how to start and stop while loops

What are we covering today?

Debugging

- Bugs (code errors)
- How to find them and remove them

More Looping and Coding

- More code with looping
- And other complexities

Debugging

It’s going to take up most of your time as a programmer!

- What is a bug?
- Different types of bugs
- How to find bugs

Debugging is the process of finding and removing software bugs

What is a Software Bug?

Errors in code are called “bugs”

- Something we have written (or not written) in our code
- Any kind of error that stops the program from running as intended

Two most common types of bugs

- Syntax Errors
- Logical Errors

Syntax Errors

C is a specific language with its own grammar

- **Syntax** - the precise use of a language within its rules
- C is very much more specific than most human languages

- Slight mistakes in the characters we use can result in different behaviour
- Some syntax errors are obvious and your compiler will find them
- Some are more devious and the error message will be confused

Logical Errors

We can write a functional program that still doesn't solve our problem

- Logical errors can be syntactically correct
- But they mean the program doesn't do what it's supposed to

Human error is real!

- Sometimes we read the problem specification wrongly
- Sometimes we forget the initial goal of the program
- Sometimes we solve the wrong problem

How do we find bugs?

Sometimes they find us . . .

- **Compilers** can catch some syntactical bugs
- We'll need to learn how to use compilers to correct our code
- Code Reviews and pair programming help for logical bugs
- **Testing** is always super important!
- Learning how to test is a very valuable skill

Using our Compiler to hunt Syntax Bugs

The Compiler can be trusted to understand the language better than us

- The simplest thing we can do is run dcc and see what happens

What to do when dcc gives you errors and warnings

- Always start with the first error
- Subsequent errors might just be a consequence of that first line
- An error is the result of an issue, not necessarily the cause
- At the very least, you will know a line and character where something has gone wrong

Solving Compiler Errors

Compiler Errors will usually point out a syntax bug for us

- Look for a line number and character (column) in the error message
- Sometimes knowing where it is is enough for you to solve it

- Read the error message and try to interpret it
- Remember that the error message is from a program that reads code
- It might not make sense initially!
- Sometimes it’s an expectation of something that’s missing
- Sometimes it’s confusion based on something being incorrect syntax

Let’s look at some code and fix some bugs

debugThis.c is a program with some bugs in it . . .

What errors did we find?

Just focusing on fixing compiler errors, let's read and fix some code

What did we discover? *(spoilers here . . . try debugging before reading this slide!)*

- Single = if statement.
  - = is and assignment of a value
  - == is a relational operator
- An extra bracket causes a lot of issues (and a very odd error message)
- Scanf not pointing at a variable

Testing

We'll often test parts of all of our code to make sure it's working

- Simple - Run the code
- Try different types of inputs to see different situations (autotest!)
- Try using inputs that are not what is expected

How do you know if the tests are succeeding?

- Use output to show information as the program runs
- Check different sections of the code to see where errors are

Simple Testing

Let's use a good process here that we can apply to all code testing

- Write your program to give you a lot of information
- Test with intention. It's valuable to test with specific goals
- Be able to find out what the code is doing at different points in the code
- Be able to separate different sections of code

Finding a needle in a haystack gets easier if you can split the haystack into smaller parts

Let’s try some information gathering

Some of the tricks we’ll use, continuing with our debugThis.c

- How is it meant to run?
- Decide on some ranges of inputs to test
- Modify the code to give useful information while it’s running

What did we test?

What techniques did we use?

- Try different input ranges, including 0 and negative numbers
- Try outputting x and y values to make sure they’re working
- Try outputting loop information so that we can see our structure

When we do good testing, we will be able to find our logical errors even if they are syntactically correct C code.

Break Time

Five Minute Break

- Debugging - the removal of software errors (known as bugs)
- Syntax Errors
- Logical Errors
- Use of compilers to reduce syntax errors
- Use of testing to learn more about what our program is doing

A more complex program

We've been learning a lot 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

```
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 counter1 = 1;
while (counter1 <= diceOneSize) {
    int counter2 = 1;
    while (counter2 <= diceTwoSize) {
        printf("%d, %d. Total: %d\n",
               counter1, counter2, counter1 + counter2);
        counter2++;
    }
    counter1++;
}
```

Quick Pause for new C syntax

Incrementing just got a little easier

```
int counter1 = 0;
int counter2 = 0;

// The following two lines have the same effect
counter1 = counter1 + 1;
counter2++;

// 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

```
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

```
// loop through and output rolls with totals that match the target
int counter1 = 1;
while (counter1 <= diceOneSize) {
    int counter2 = 1;
    while (counter2 <= diceTwoSize) {
        int total = counter1 + counter2;
        if (total == targetValue) {
            printf("%d, %d. Total: %d\n",
                    counter1, counter2, total);
        }
        counter2++;
    }
    counter1++;
}
```

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 counter1 = 1;
while (counter1 <= diceOneSize) {
    int counter2 = 1;
    while (counter2 <= diceTwoSize) {
        numRolls++;
        int total = counter1 + counter2;
        if (total == targetValue) {
            numSuccesses++;
            printf("%d, %d. Total: %d\n",
                    counter1, counter2, total);
        }
        counter2++;
    }
    counter1++;
}
```

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?

Displaying odds

- Can you make the program display “1 in 6 chance” instead of a percentage?

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?

Debugging

- Software bugs (errors)
- There are different kinds
- Some beginning skills for how to find and fix them

More coding with dice

- Loops within loops
- Checking and counting values
- Being careful with division and using doubles for fractions