COMP1511 - Programming Fundamentals

Term 3, 2019 - Lecture 4

What did we learn yesterday?

- **EVERYTHING!** - A recap of the C we've seen so far
- Problem Solving
- Continuing work with if and else statements
- Showing use of Relational and Logical Operators
- Showing some use of Modulus

What are we covering today?

Code Style

- Why being stylish is cool . . .
- Actually, it’s about readability and reusability of code

Looping

- Repetitive tasks shouldn’t require repetitive coding
- While loops

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 thinking 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 the reason for a variable 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 also 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 later)

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

Executing the same code more than once

Sometimes we need to repeat our work

- C normally executes in order, line by line
- if statements allow us to “turn on or off” parts of our code
- But up until now, we don’t have a way to repeat code
- Copy-pasting the same code again and again is not a feasible solution

While Loops

“while” is a C keyword that lets us loop code

- Format is very similar to an if statement
- The “question” in the (brackets) functions very similarly
- If it’s true, the body of the while loop will run
- If it’s false, the body won’t run and the program will continue
- Once a while reaches the end of its `{ }` it will start again

While Loop Code Format

```
// expression is checked at the start of every loop
while (expression) {
    // this will run again and again
    // until the expression is evaluated as false
}
// When the program reaches this }, it will jump
// back to the start of the while loop
```

While Loop Control

We can use a variable to control how many times a while loop runs

- We call this variable a "loop counter"
- It’s an **int** that’s declared outside the loop
- It’s “termination condition” can be checked in the while expression
- It will be updated inside the loop

We can also use a variable to decide to exit a loop at any time

- We call this variable a "sentinel"
- It's like an on/off switch for the loop

While Loop with a Loop Counter

```
// 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
```

While Loops and Termination

It's actually very easy to make a program that goes forever

Consider the following while loop:

```
while (1 < 2) {
    // Never going to give you up
    // Never going to let you down . . .
}
```

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;
    }
}
```

While loops, if statements and other code

It's all code!

- An if statement is some code
- A while loop is also some code

This means that you can . . .

- Put ifs inside while loops
- Put while loops inside ifs or elses
- Put while loops inside while loops inside if statements
- Etc

Let’s make a program that loops

We're going to build up to drawing a grid of *s

- Start by looping and writing asterisks (*) to the terminal
- Take some user input and write exactly that many asterisks
- Then loop inside a loop to display a square grid
- Make the program run multiple times instead of just ending

Looping and writing a line of asterisks

```
// A simple program for drawing a grid pattern
// Part 1, draw a line of asterisks
// Marc Chee, February 2019

int main (void) {
    int gridSize = 8;
    int counter = 0;
    while (counter < gridSize) {
        printf("*");
    }
    printf("\n");
}
```

Looping based on user input

```
// A simple program for drawing a chessboard
// Part 2, let the user choose the length
// Marc Chee, February 2019

int main (void) {
    int gridSize;
    int counter = 0;

    // let the user choose our grid size
    printf("Please enter the size of the grid: ");
    scanf("%d", &gridSize);

    while (counter < gridSize) {
        printf("*");
        counter = counter + 1;
    }
    printf("\n");
}
```

Now draw a square instead of a line

```
// let the user choose our grid size
printf("Please enter the size of the grid: ");
scanf("%d", &gridSize);

// loop through and print multiple rows
while (y < gridSize) {
    // print a single row
    while (x < gridSize) {
        printf("*");
        x = x + 1;
    }
    // the row is finished, start the next line
    printf("\n");
    y = y + 1;
    x = 0; // reset x for the next line
}
```

Since we're looping, let's repeat

We want the program to run again instead of ending

- Put the main functionality of the program inside a loop
- Make sure whatever variables we used are reset each time

Repeating Grid Drawing

The start of the while loop that contains our previous code

```
int exit = 0; // a sentinel variable

while(exit == 0) {
    // let the user choose our grid size
    printf("Please enter the size of the grid or 0 to exit: ");
    scanf("%d", &gridSize);

    // if the user has chosen to exit
    if (gridSize == 0) {
        printf("Thank you for using Grid Drawer.");
        // setting the sentinel to leave the loop
        exit = 1;
    }
    // the rest of the grid drawing code starts here . . .
```

Repeating Grid Drawing

After the drawing is complete

```
            // looping drawing code finishes here
        }
        // make a gap between different runs
        printf("\n");
        // reset variables for the next run
        gridSize = 0;
        x = 0;
        y = 0;
    }
    // This ends the while loop, which will take us
    // back to the start of the program
```

We have The Grid

You’re only one step away from creating a digital frontier . . .

- We can draw square patterns of different sizes based on user input
- We’ve now used loops inside other loops
- We’ve also made loops that could potentially be infinite

Challenge (for bonus Marcs, may not be equivalent to marks)

- Can you make the asterisks appear only as a border pattern?
- What about a checkerboard pattern?

What did we learn today?

Code Style

- Style is cool. Code Style is even cooler :P
- Readability and reusability is very important in code!
- We have some concrete guidelines for writing neat code

While Loops

- Repeating execution of code
- We've made some loops
- We've shown how to loop inside other loops
- We've shown different ways to end loops