COMP1511 - Programming Fundamentals

Term 1, 2019 - Lecture 8
Stream B

What did we learn on Tuesday?

Computers in Theory

- Processors and Memory
- Turing Machines
- How C uses memory

Arrays

- Collections of variables
- Looping through arrays

What are we covering today?

Arrays

- Recap of what arrays are
- A little bit more information

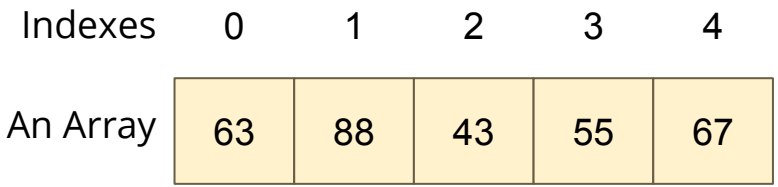
Functions

- We've seen our main function . . . now we introduce other functions
- We'll use both arrays and functions in a program

What is an array?

A collection of identical variables

- Contains multiple variables all of the same type
- Declared using a variable type and a size
- Individual variables are accessed using an index



Using Arrays in C

Some example code of an array

```
int main (void) {
    // declare an array of doubles, size 4, initially all 0
    double myArray[4] = {0};

    // assign a value
    myArray[1] = 0.95;
    // test a value
    if (myArray[2] < 1) {
        // print out a value
        printf("Third element is: %lf", myArray[2]);
    }
```

Accessing multiple values at once

Loops and Arrays go together perfectly

- Accessing all members is a reasonably simple while loop

```
int main (void) {
    // declare an array of doubles, size 4, initially all 0
    double myArray[4] = {0};

    // loop through the array and output the elements
    int counter = 0;
    while (counter < 4) {
        printf("%lf\n", myArray[counter]);
        counter++;
    }
```

Creating Arrays

Arrays start at an exact size and don't change

- When we create an array, we give it a size and a type
- Both of those are fixed and won't change

```
int main (void) {
    // declare an array of doubles,
    // size 4
    double myArray[4] = {0};

}
```

```
int main (void) {
    // This declaration is not
    // possible!
    int arraySize = 4;
    double myArray[arraySize] = {0};

}
```

We can't declare an array with a variable size like this!

Using Constants for Array Sizes

If we do want to be able to change the size in code . . .

- We can use a constant to set the size
- Unlike a variable, this cannot change after it is compiled
- It does make our lives much easier while we're coding though!

```
#define ARRAY_SIZE 4

int main (void) {
    // This declaration allows us to change the
    // array size while coding
    double myArray[ARRAY_SIZE] = {0};
}
```

Functions

Before we finish our program, let's look at functions

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

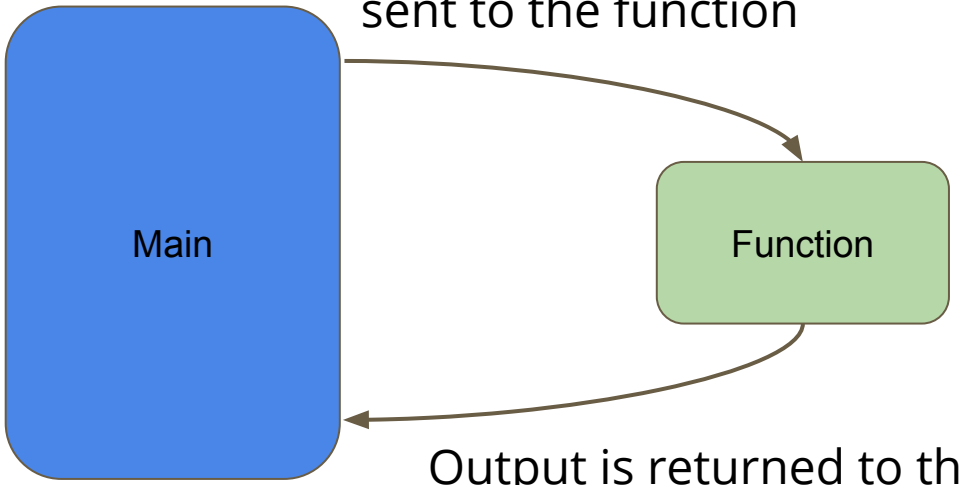
What is a function?

- 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 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?

Given the existence of the function . . .

- 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);
    return;
}
```

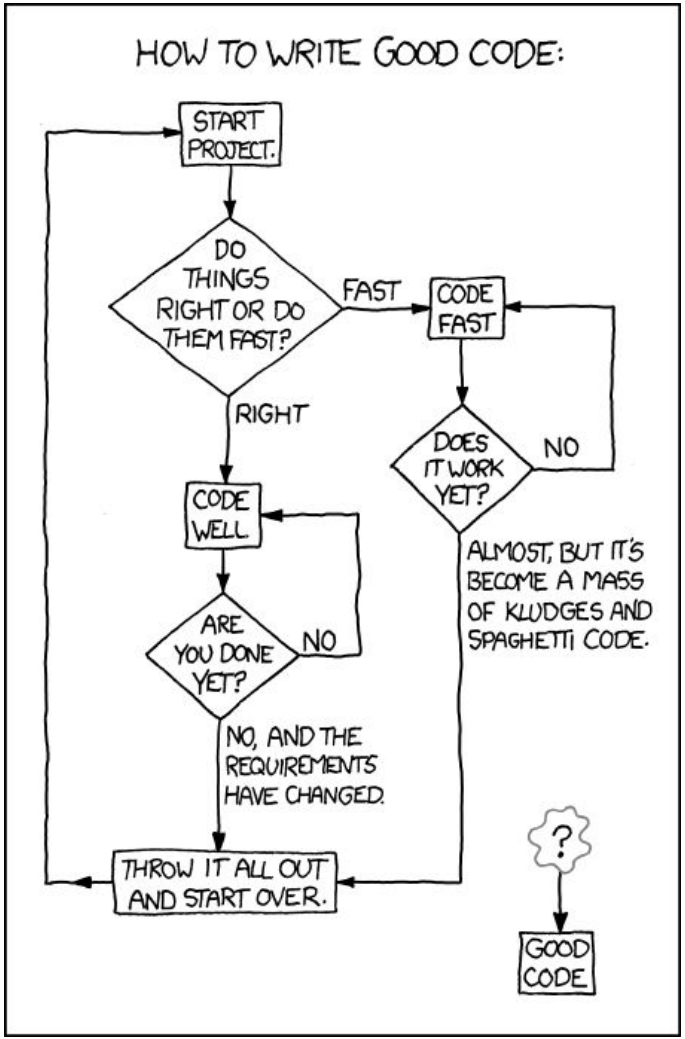
Break Time

Arrays

- Recap
- Declaring Arrays of a certain size

Functions

- Separate code that we can “call”



https://xkcd.com/844/

Let’s play a modified game of Snap

Snap is a simple card game

- We play cards one after the other
- If we play a card that’s the same as the previous, we call out “Snap!”

Our “Snap!” program will be a little different

- We’re going to write a program that takes input numbers one at a time
- It will then call out “Snap!” if it’s seen your input before

Break it down

We have a potentially complex goal, so how do we simplify it?

- Loop the program
- Enter a number each loop
- We need to remember all numbers that have been played
- So we store the numbers in an array

Let’s write some code to remember things first

What features will we start with

- A constant that tells us how many turns are in the game
- A loop that runs once per game turn
- Some useful messages to our user so they know what's happening
- Store the information the user gives us in an array

Starting Code

```
#define NUM_MAX_TURNS 10

int main (void) {
    int prevNums[NUM_MAX_TURNS] = {0};

    printf("Welcome to a game of snap with your computer!\n");
    printf("Please enter numbers when prompted.\n");

    // the main game loop. Each turn of the game is one iteration
    int counter = 0;
    while (counter < NUM_MAX_TURNS) {
        printf("Please enter a number: ");
        scanf("%d", &prevNums[counter]);
        counter++;
    }
```

How do we know if a number is in an array?

Let's think about this problem separately

To find a number in an array . . .

- Loop through the array
- Check the number we want against the number in the array
- If it's true, then we know it's in the array

Let’s write this in a function

What does it need to tell us?

- Output type

What does it need to know to be able to tell us this?

- Input Parameters

What does it do?

- Name

NumberCheck Function

A function for saying Snap! if a number has been found in an array

```
void numberCheck (int prevNumbers[], int number) {
    int counter = 0;
    // print "Snap!" if the number has been seen before
    while (counter < NUM_MAX_TURNS) {
        if (number == prevNumbers[counter]) {
            printf("Snap!\n");
        }
        counter++;
    }
    return;
}
```

Using this Function in our Snap Program

We need to do the following:

- Declare the function before our main
- Add the function code to our C file
- Call the function in our main

Snap with the NumberCheck Function

```c
#define NUM_MAX_TURNS 10

int numberCheck(int number, int prevNums[]);

int main (void) {
    int prevNums[NUM_MAX_TURNS] = {0};
    printf("Welcome to a game of snap with your computer!\n");
    printf("Please enter numbers when prompted.\n");

    // the main game loop. Each turn of the game is one iteration
    int counter = 0;
    while (counter < NUM_MAX_TURNS) {
        printf("Please enter a number: ");
        scanf("%d", &prevNums[counter]);
        numberCheck(prevNums[counter], prevNums); // function call
        counter++;
    }
```

Some Testing

Even if we're free of Syntax Errors, we could have Logical Errors

- What kind of tests should we run?
- All different numbers?
- All the same number?
- Numbers in pairs?
- The same number separated by other numbers?
- Zero?
- Negative Numbers?

What results did we get?

Uh oh!

- There’s something up with our function
- We’re getting a lot false positives!
- We’re getting repetitive Snaps also!

Let’s delve in and see what’s happening

- Get more information with better testing!

NumberCheck with Testing

```
void numberCheck (int prevNumbers[], int number) {
    int counter = 0;
    // print "Snap!" if the number has been seen before
    while (counter < NUM_MAX_TURNS) {
        if (number == prevNumbers[counter]) {
            printf("Snap!\n");
            printf("Found %d at index %d\n", number, counter);
        }
        counter++;
    }
    return;
}
```

What are our test results?

- We’re finding the number we just entered!
- We’re finding ALL the times that number appeared before
- Zero . . . we initialised the array with all zeros, so it’s finding ALL of them!

Fixes

- Let’s check for the number **before** we put it in the array
- To stop the multiple Snaps, we could exit the loop early

Check the Array before inserting the number

```
int main (void) {
    int prevNums[NUM_MAX_TURNS] = {0};
    printf("Welcome to a game of snap with your computer!\n");
    printf("Please enter numbers when prompted.\n");

    // the main game loop. Each turn of the game is one iteration
    int counter = 0;
    while (counter < NUM_MAX_TURNS) {
        printf("Please enter a number: ");
        int currentNumber;
        scanf("%d", &currentNumber);
        numberCheck(currentNumber, prevNums);
        prevNums[counter] = currentNumber;
        counter++;
    }
```

Exit the array if we've "Snapped"

```
void numberCheck (int prevNumbers[], int number) {
    int counter = 0;
    int exitLoop = 0;
    // print "Snap!" if the number has been seen before
    // we only need to find it once
    while (counter < NUM_MAX_TURNS && exitLoop != 0) {
        if (number == prevNumbers[counter]) {
            printf("Snap!\n");
            exitLoop = 1;
        }
        counter++;
    }
    return;
}
```

Nearly done

Snap! is mostly working

- We’re detecting numbers we’ve seen before
- We’re snapping exactly once

What else?

- We’re still not dealing with zeroes
- Can you think of another way to exit the loop in NumberCheck?
- Can you come up with a way to deal with both of these problems at once?

What did we learn today?

Arrays

- Recap of Arrays
- Delcaring arrays with constants

Functions

- Separate code we can write and call
- Has a type (it’s output)
- Has input parameters