COMP1511 - Programming Fundamentals

Term 1, 2020 - Lecture 17

What did we learn last week?

Abstract Data Types

- Using multiple file projects
- Protecting some data from access
- Providing a nice set of functions as an interface to the code

What are we covering today?

Assessment

- The exam
- The format
- How to prepare

A recap of what we've covered in the course

- The first half of COMP1511

What's in the Exam?

Meme credit: Hyperbole and a Half

The Exam - Timing and details

9am (Australian Eastern Standard Time) 5th May for 24 hours

- Completed on your own computer at home
- Expected to take at most 3-5 hours
- Auto testing and submission is available
- You can submit more than once, your final submission will be marked
- Your Week 10 labs will show you what the exam environment is like
- During the exam, we will contact you via your UNSW email if/when we need to

The Exam - "Open Book"

- "Open Book" - meaning you can use any resources from the class or online
- Still an exam! No communication between students!
- No external help! You cannot ask questions online or in discussion groups
- No discussion of the assignment (or sharing of code) with anyone except for COMP1511 subject staff
- The Ed forum will be available during the exam for private non-programming help (admin and infrastructure questions)
- We will also have an email address to contact us: **`cs1511.exam@cse.unsw.edu.au`**

The Exam Format

The following details might change, but only slightly

- **10** short answer "theory" questions
  - Some multiple choice and others a very small amount of text
- **7** practical questions
  - Practical questions will involve actual programming
  - Very similar to Lab/Test questions

Exams - Marc's tips

How to survive a (take-home) exam

- Drink water. Dehydration lowers brain functions
- Eat decent meals. Blood sugar also affects your brain, especially in a stressful situation
- Get sleep. It's a 24 hour exam only to be polite to time zones!
- Take breaks! Humans can't think well for more than about an hour at a time
- You don't need to work for 24 hours, especially in a Pass/Fail course
- This exam was designed to be finished in 3 hours with no external resources, so you don't have to try to 100% it

Exams - Marc's tips

- If you want to prepare, you can look at the list of topics in COMP1511 and write down your own notes in advance for how you want to approach certain types of questions
- Draw Diagrams for any questions you need to think about

I'd say *"chill out, this isn't a big deal"* but no one will believe me

Short Answer Questions

Quick Questions

- These questions will be about whether you understand core coding concepts and the C programming language
- Your answers will either be multiple choice or short answers
- Some are: "What will this code do?"
- Some are: "How does this concept work?"
- Some examples are in the Week 10 Lab

Short Answer Questions - Marc's tips

These Questions are a "warmup"

- Read through them all before answering
- Skim quickly and answer the ones you definitely know
- Then go back to the ones that take some time to think about
- Prioritise! Get the easy marks, then spend time on the ones you're reasonably sure of
- If you're unsure, go back to the lecture slides and do a text search for what you think you need
- Hit up lecture videos if you need to relearn something

Practical Questions

Less questions, more work

- Questions are similar to the Weekly Tests and Labs
- Stages of difficulty from basic to extreme challenge
- Some will have provided code as frameworks
- Each question will need to be written, compiled and tested
- You will have access to an autotest (but it's just a test!)
- There will be no specific style marking, so you don't need to explain your code in comments

Hurdles

Hurdles must be passed to pass the course

- There's an array hurdle, question 1 or 3
- There's a linked list hurdle, question 2 or 4
- You must earn a mark of 50% or more in both the array hurdle and the linked list hurdle
- The simplest thing is to put a serious effort into questions 1 and 2, which will cover both hurdles

Practical Questions - Marc's tips

Solving Problems

- Read all the questions before starting
- Pick the easy ones as you read. Most likely the earlier questions
- Prepare! A couple of minutes thinking and writing a diagram will clarify how you're going to approach a question
- Use your lab/test practice! Debugging and testing will be important here
- Less questions answered completely is better than more questions partially answered
- Don't count the number of autotests. Marks are not based on number of tests passed

Questions 1-2 First Hurdles

Basic C Programming - similar to Weekly Test question 1

- Create C programs
- Use variables (ints and doubles)
- scanf and printf
- if statements and loops
- Read command line arguments (possibly convert to ints and doubles)
- Basic use of arrays of ints/doubles (q1)
- Basic use of linked lists of ints/doubles (q2)

Example Question 1

Loop through an array and gather some kind of information

Eg: Go through all the elements of an array. Print out every even number in the array on its own line.

Edit the function: **`evens(int length, int numbers[])`**

```
% ./evens 13 14 15 16 17
14
16
```

Example Question 2

Perform some computation on a linked list

Eg: Given a linked list, add up all the values stored in it and return that integer.

Edit the function: `int sumList(struct node *head)`

```
% ./sumList 5 4 3 2 1
15
```

Questions 3-4 Harder Hurdles

More advanced C - similar to Weekly Test question 2

- Everything from Questions 1 and 2 as well as . . .
- Looping through possibly more than once
- Testing more difficult conditions and keeping track of more than one concept
- Working with Arrays (q3)
- Working with Linked Lists (q4)

Questions 5-6

Even Harder C - similar to Weekly Test question 3

- Using strings (q5)
- Possibly fgets, fputs, command line arguments etc
- Manipulate linked lists (adding and removing items etc) (q6)
- Potentially use malloc() and free() with structs and pointers
- Again, more complex combinations, and some questions requiring interesting problem solving

Question 7

Challenge Questions for people chasing HDs

- Everything taught in the course might be in these questions
- Think Challenge Exercises, even some of the hard ones!
- Will also test your ability to break a problem down into its parts
- This week's lab has a past Question 8 (we used to have 8) so you can see it

What to study

A little preparation goes a long way

- The basics are important!
- A basic knowledge of all topics is better than an extreme level of knowledge in just one
- Know how to use both **arrays** and **linked lists**
- Try some revision questions from the Tutorials or Labs
- The revision exercises on the course webpage are also very useful (this section will be added to the website this week)

How important are different topics?

Important

- Variables, If, Looping, Functions, Arrays, Linked Lists, Characters and Strings

Things that you will need to understand the important topics

- Pointers, Structs, Memory Allocation

Stretch Goals

- Abstract Data Types
- Multi-file programs

Exam Marking

Most of the marking will be automated

- Make sure your input/output format matches the specification
- Answers for hurdles will also be checked by hand
- Marks will be earnt for correct code, not for passing autotests
- Minor errors, like a typo in an otherwise correct solution, will only result in a small loss of marks
- Results should be ready by around the 22nd May

Special Consideration and Supplementary Exam

- If you attend the exam, it's an indication that you are well enough to sit the exam
- If you are not well enough to sit the exam, apply for Special Consideration and do not attend the exam
- If you become sick during the exam, let us know on Ed and/or the exam email address: **`cs1511.exam@cse.unsw.edu.au`**
- A supplementary exam will be held between the 25th and 29th May. If you think you will need to sit this exam, make sure you are available.

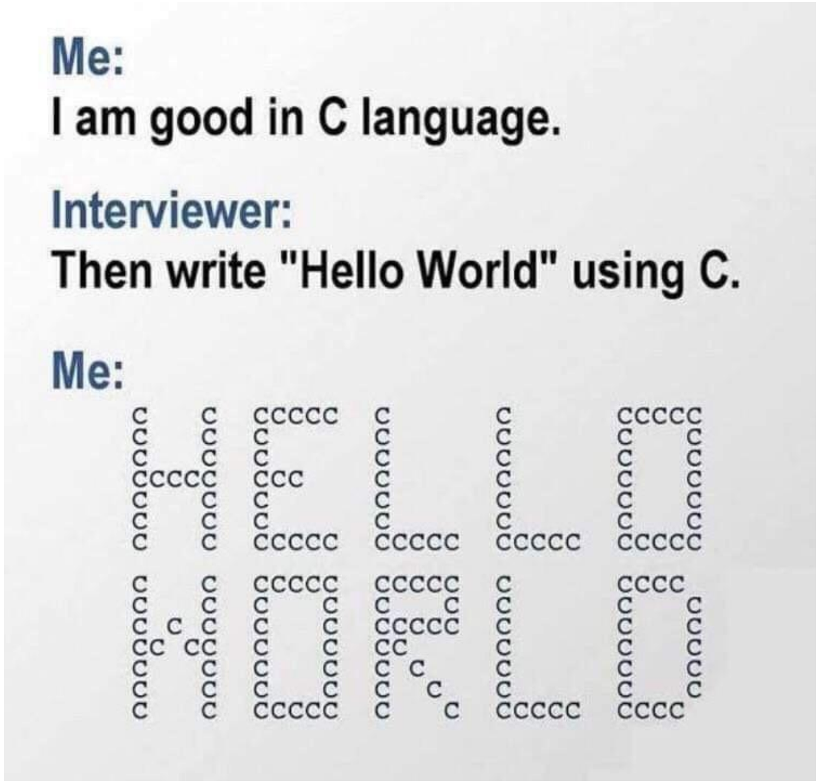
Break Time

Human memory is based on active recall

- You can store something in your long term memory by reminding yourself of it repeatedly
- Active recall means using, not just reading
- Link your memory to things you already know (use examples in your revision code that are things you know well)
- Get some exercise! Active blood flow, even just a bit of walking, helps the brain

What did we learn this term?

Programming in C



Me:
I am good in C language.

Interviewer:
Then write "Hello World" using C.

Me:

```
c   c   ccccc   c       c       ccccc
c   c   c       c       c       c   c
c   c   c       c       c       c   c
ccccc   ccc     c       c       c   c
c   c   c       c       c       c   c
c   c   c       c       c       c   c
c   c   ccccc   ccccc   ccccc   ccccc

c   c   ccccc   ccccc   c       cccc
c   c   c   c   c   c   c       c   c
c c c   c   c   ccccc   c       c   c
cc cc   c   c   cc      c       c   c
c   c   c   c   c c     c       c   c
c   c   c   c   c  c    c       c   c
c   c   ccccc   c   c   ccccc   cccc
```

Programming in C

COMP1511 C Language Techniques in the order they were taught

- Input/Output
- Variables
- If statements
- While statements (looping)
- Arrays
- Functions
- Pointers
- Characters and Strings
- Structures
- Memory
- Command Line Arguments
- Multi-File Projects
- Linked Lists
- Abstract Data Types

C as a programming language

- A compiled language
- We use dcc as our compiler here, but there are others
    - clang
    - gcc
    - etc
- Compilers read code from the top to the bottom
- They translate it into executable machine code
- All C programs must have a **`main()`** function, which is their starting point
- Compilers can handle multiple file projects
- We compile C files while we #include H files

C and Compilation

When my code compiles on the first time

Input/Output

Scanf and Printf allow us to communicate with our user

- `scanf` reads from the standard input
- `printf` writes to standard input
- They both use pattern strings like `%d` and `%s` to format our data in a readable way

```
// ask the user for a number, then say it back to them
int number;
printf("Please enter a number: ");
scanf("%d", &number);
printf("You entered: %d", number);
```

Alternatives for input/output

We can get and put lines and characters also

- `getchar` and `putchar` will perform input and output in single characters
- `fgets` and `fputs` will perform input and output with lines of text
- We can also use handy functions like `strtol` to convert characters to numbers so we can store them in integers

Command Line Arguments

When we run a program, we can add words after the program name

- These extra strings are given to the main function to use
- **`argc`** is an integer that is the total number of words (including the program name)
- **`argv`** is an array of strings that contain all the words

Command Line Arguments in use

```
int main (int argc, char *argv[]) {
    printf("The %d words were ", argc);
    int i = 0;
    while (i < argc) {
        printf("%s ", argv[i]);
        i++;
    }
}
```

When this code is run with: `"./args hello world"`

It produces this: `"The 3 words were ./args hello world"`

Variables

Variables

- Store information in memory
- Come in different types:
  - **int, double, char, structs, arrays** etc
- We can change the value of variables
- We can pass the value of variables to functions
- We can pass variables to functions via pointers

Constants

- **`#define`** allows us to set constant values that won't change in the program

Simple Variables Code

```c
// GOKU will be treated as if it's 9001 in our code
#define GOKU 9001

int main (void) {
    // Declaring a variable
    int answer;
    // Initialising the variable
    answer = 7;
    // Assign the variable a different value
    answer = BATMAN;

    // we can also Declare and Initialise together
    int answerTwo = 88;
}
```

If statements

Questions and answers

- Conditional programming
- Evaluate an expression, running the code in the brackets
- Run the body inside the curly brackets if the expression is true (non-zero)

```
if (x < y) {
    // This section runs if x is less than y
}
// otherwise the code skips to here if the
// expression in the () equates to 0
```

While loops

Looping Code

- While loops allow us to run the same code multiple times
- We can stop them after a set number of times
- Or we can stop them after a certain condition is met

Loops are used for . . .

- Checking all the values in a data structure (**array** or **linked list**)
- Repeating a task until something specific changes
- and any other repetition we might need

Looping

```
x += 50;
```

```
x = x + 50;
```

```
for (int i = 0, i++, i<50)
{
    x++;
}
```

```
x++; x++; x++; x++; x++;
x++; x++; x++; x++; x++;
x++; x++; x++; x++; x++;
x++; x++; x++; x++; x++;
x++; x++; x++; x++; x++;
x++; x++; x++; x++; x++;
x++; x++; x++; x++; x++;
x++; x++; x++; x++; x++;
x++; x++; x++; x++; x++;
x++; x++; x++; x++; x++;
```

While loop code - Arrays

Very commonly used to loop through an array

```
int numbers[10] = {0};

// set array to the numbers 0-9 sequential
int i = 0;
while (i < 10) {
    // code in here will run 10 times
    numbers[i] = counter;
    // increment the counter
    i++;
}
// When counter hits 10 and the loop's test fails
// the program will exit the loop
```

While loop code - Linked Lists

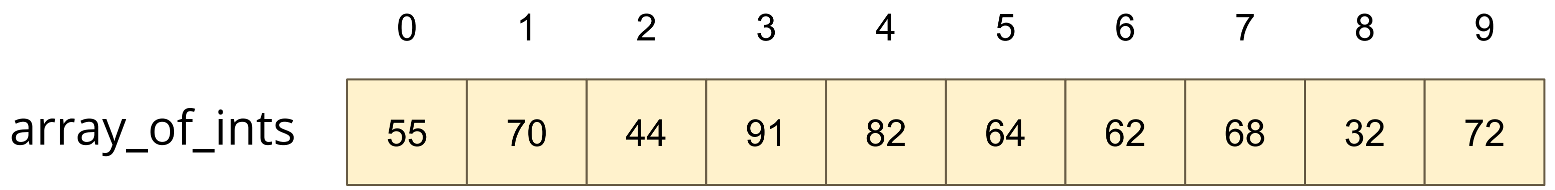
Looping through Linked Lists is also very common

```c
// loopNode starts pointing at the first element of the list
struct node *loopNode = head;

while (loopNode != NULL) {
    // code in here will run until the loopNode pointer
    // moves off the end of the list

    // increment the node pointer
    loopNode = loopNode->next;
}
// When loopNode pointer is aiming off the end of the list
// the program will exit the loop
```

Arrays

Collections of variables of the same type

- We use these if we need multiple of the same type of variable
- The array size is decided when it is created and cannot change
- Array elements are collected together in memory
- Not accessible individually by name, but by index



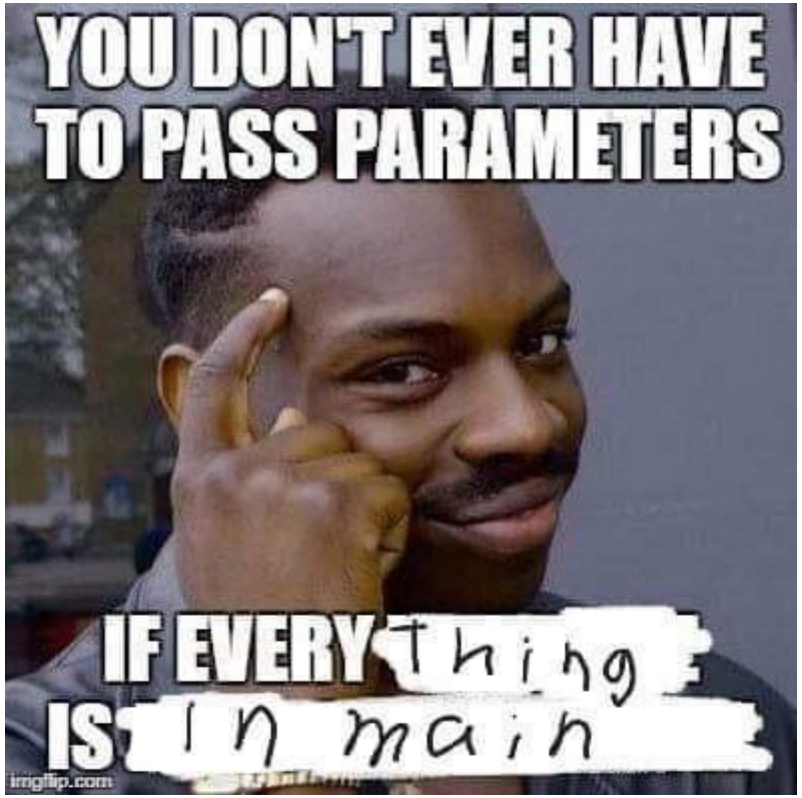
Array Code

```
int main (void) {
    // declare an array, all zeroes
    int marks[10] = {0};

    // set first element to 85
    marks[0] = 85;
    // access using an index variable
    int accessIndex = 3;
    marks[accessIndex] = 50;
    // copy one element over another
    marks[2] = marks[6];
    // cause an error by trying to access out of bounds
    marks[10] = 99;
```

Functions

Code that is written separately and is called by name

- Not written in the line by line flow
- A block of code that is given a name
- This code runs every time that name is "called" by other code
- Functions have input parameters and an output


YOU DON'T EVER HAVE TO PASS PARAMETERS
IF EVERY thing IS in main
imgflip.com

Function Code

```c
// Function Declarations above the main or in a header file
int add (int a, int b);

int main (void) {
    int firstNumber = 4;
    int secondNumber = 6;
    int total = add(firstNumber, secondNumber);
    return 0;
}

// This function takes two integers and returns their sum
int add (int a, int b) {
    return a + b;
}
```

Pointers

Variables that refer to other variables

- A pointer aims at memory (actually stores a memory address)
- That memory can be another variable already in the program
- It can also be allocated memory
- The pointer allows us to access another variable

- `*` dereferences the pointer (access the variable it's pointing at)
- `&` gives the address of a variable (like making a pointer to it)
- `->` is used with structs to allow a pointer to access a field inside

Simple Pointers Code

```
int main (void) {
    int i = 100;
    // the pointer ip will aim at the integer i
    int *ip = &i;
    printf("The value of the variable at address %p is %d\n", ip, *ip);

    // this second print statement will show the same address
    // but a value one higher than the previous
    increment(ip);
    printf("The value of the variable at address %p is %d\n", ip, *ip);
}

void increment (int *i) {
     *i = *i + 1;
}
```

What did we learn today?

Exam

- The rough format
- What to study

The first half of the course

- The technical parts of the first half of the course
- Basic C programming up to pointers