COMP1511 - Programming Fundamentals

Term 3, 2019 - Lecture 19

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?

The Exam

29th November in one of two 3 hour sessions

- Last week, you chose between a morning and afternoon session
- If you are allocated a session and have a serious reason why you can't attend that session, email **`cs1511@cse.unsw.edu.au`** after allocations

- Completed on a lab computer under exam conditions
- No external materials allowed
- Your Week 10 labs will show you what the exam environment is like

The Exam Format

The following details might change, but only slightly

- 30 minutes of **theory** questions
- **20** theory questions
- During the first 35 minutes, you will not have access to a code editor or compiler
- *Once you switch to the practical part of the exam, you will not be able to return to the theory questions!*
- 2.5 hours of **practical** coding questions
- **8** practical questions
- Practical questions will involve actual programming

Exams - Marc's tips

How to survive an exam

- Bring pens/pencils with multiple colours to draw diagrams
- You'll be provided with paper and can't bring your own
- Bring a (transparent) water bottle . . . dehydration affects your brain
- Eat a decent meal before the exam. Blood sugar also affects your brain, especially in a stressful situation
- Remember the C Reference Sheet is available in the exam

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

Theory Questions

Quick Questions, mostly in the same format:

- Here's some code
- It compiles like this
- Here's the command to run it
- What is the output?

- 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

Theory Questions - Marc's tips

How to maximise marks in a high speed theory test

- Read through them all fast before answering
- Skim quickly and answer the ones you definitely know
- Then go back to the ones that take some time to think about
- Don't get stuck . . . If something is going to take you some serious time to work out, then move on
- Prioritise your time! Get the easy marks, then spend time on the ones you're reasonably sure of. If you're not sure of something then don't let it eat your time!

Practical Questions

Less questions, more time

- 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

Practical Questions - Marc's tips

Solving Problems under pressure

- Read all the questions before starting
- Pick the easy ones as you read. Most likely the earlier questions
- Don't rush! A couple of minutes thinking and writing a diagram might be much faster than smashing out code that doesn't answer the question
- Remember your lab exercises! 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. No marks for partial autotests!

Questions 1-2

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

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

Questions 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 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 while putting yourself under a stressful time limit
- 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 will not be tested in the exam!

Exam Marking

Most of the marking will be automated

- Make sure your input/output format matches the specification
- Answers 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 approximately the 20th December

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, ask the exam supervisor for assistance and talk to the Lecturer
- A supplementary exam will be held between the 13th and 17th January 2020. If you think you will need to sit this exam, make sure you are available.

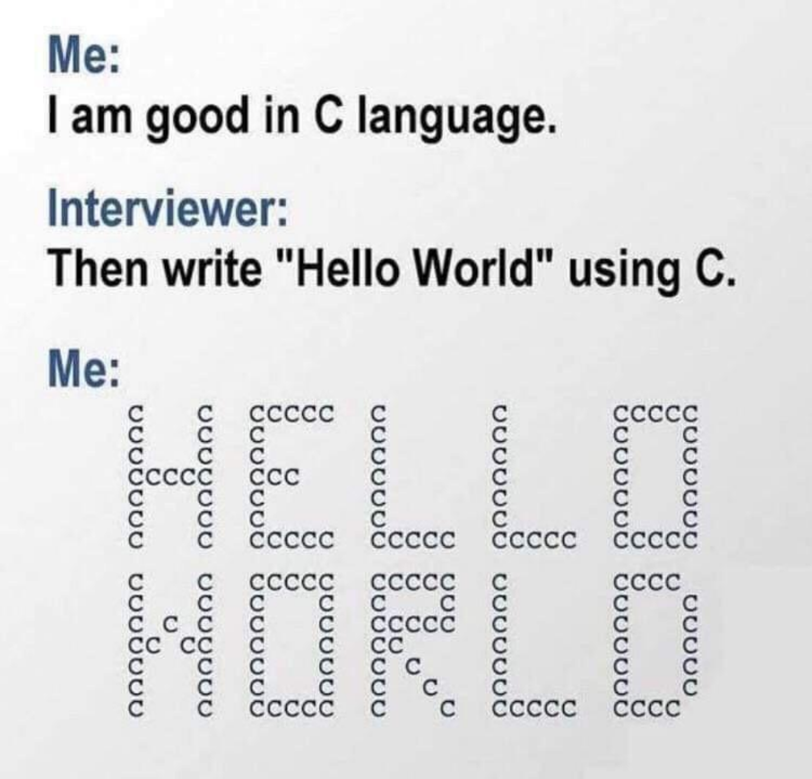
Break Time

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
- Characters and Strings
- Pointers
- Structures
- Memory
- 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
// BATMAN will be treated as if it's 100 in our code
#define BATMAN 100

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

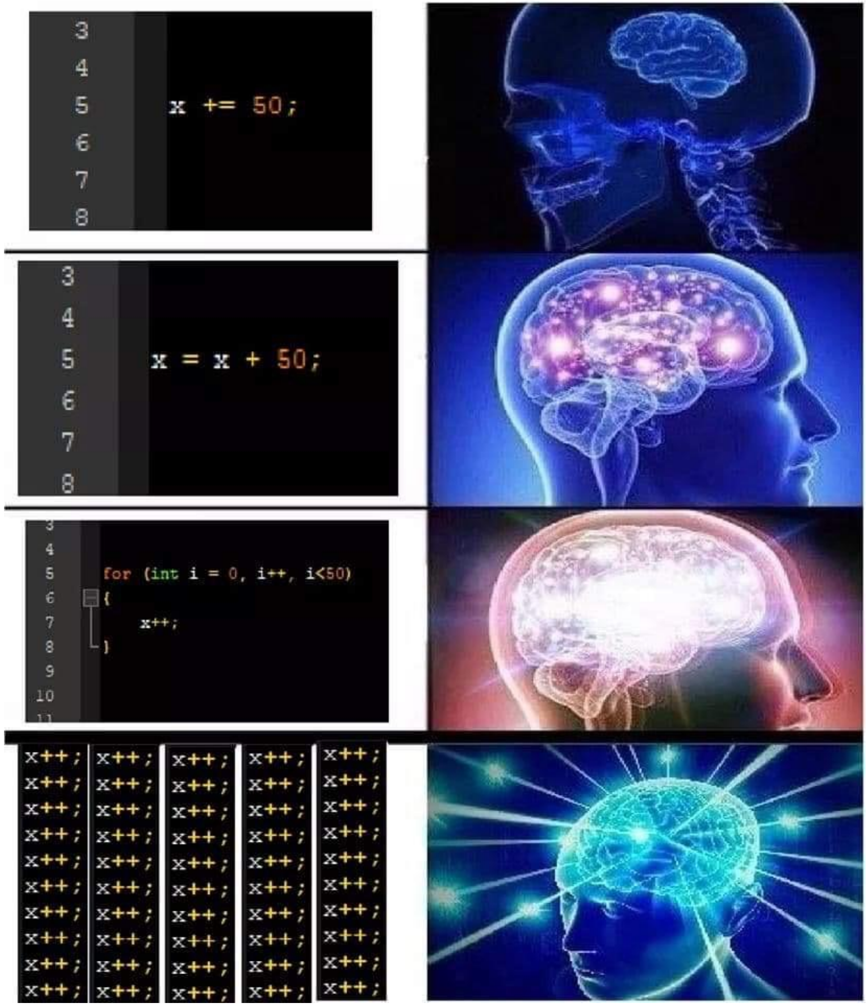
Looping Code

- 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};
int counter = 0;

// set array to the numbers 0-9 sequential
while (counter < 10) {
    // code in here will run 10 times
    numbers[counter] = counter;
    // increment the counter
    counter = counter + 1;
}
// When counter hits 10 and the loop’s test fails
// the program will exit the loop
```

While loop code - Linked Lists

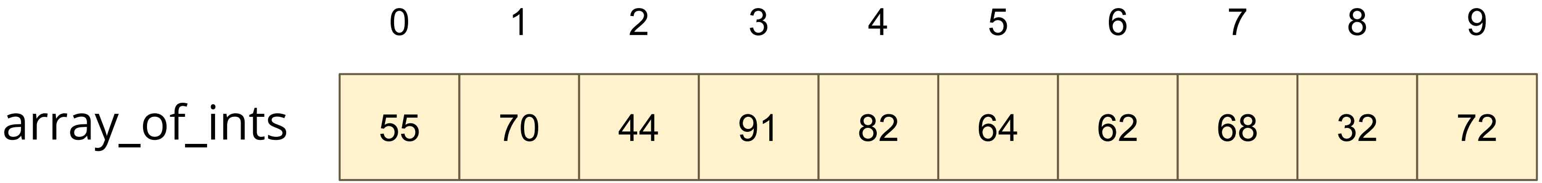
Looping through Linked Lists is also very common

```
// 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



	0	1	2	3	4	5	6	7	8	9
array_of_ints	55	70	44	91	82	64	62	68	32	72

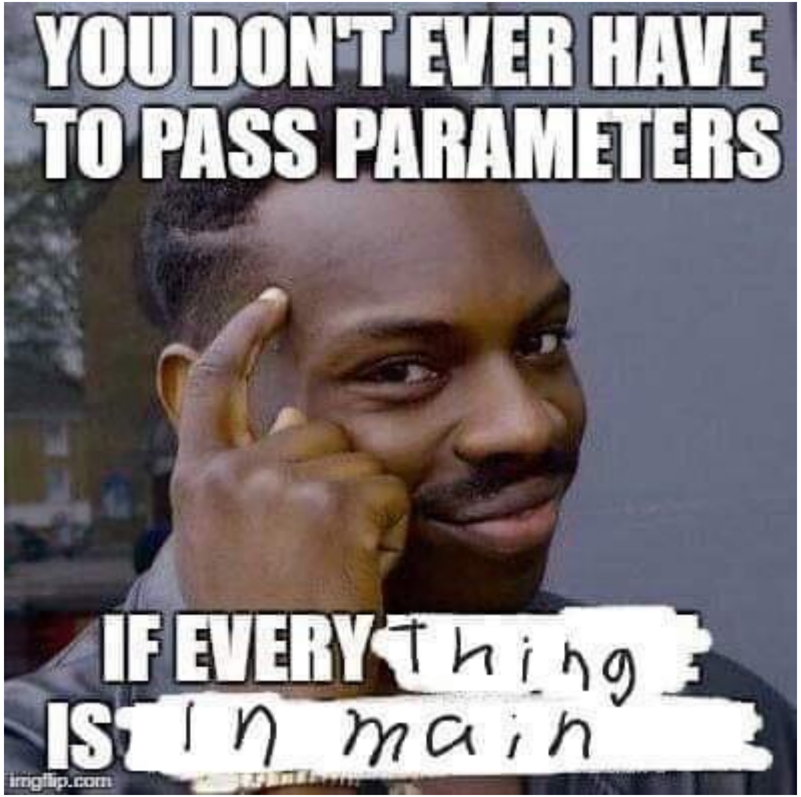
Array Code

```c
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;
}
```

Characters and Strings

Used to represent letters and words

- **char** is an 8 bit integer that allows us to encode characters
- Uses ASCII encoding (but we don't need to know ASCII to use them)

- Strings are arrays of characters
- The array is usually declared larger than it needs to be
- The word inside is ended by a Null Terminator **`'\0'`**
- Using C library functions can make working with strings easier

Characters and Strings in code

```
// read user input
char input[MAX_LENGTH];
fgets(input, MAX_LENGTH, stdin);
printf("%s\n", input);

// print string vertically
int i = 0;
while (input[i] != '\0') {
    printf("%c\n", input[i]);
    i++;
}
```

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 arrays and strings