COMP1511 - Programming Fundamentals

Week 10 - Lecture 17

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

The Exam - Timing and details

9am (Australian Eastern Standard Time) Friday 14th August for 24 hours

- Completed on your own computer at home
- Files will be provided to you when you run certain terminal commands
- Autotest and Submission will also be done with terminal commands
- Expected to take at most 3-5 hours
- 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
- Before and during the exam, we will contact you via your UNSW email if/when we need to

The Exam - Technical Details

- You will receive an email at your UNSW email address a few days before the exam
- **Make sure your UNSW email address is working!**
- This email will contain a link to the exam website
- The link will start working at 9am on the day of the exam
- The commands available in the practice exam will all be available during the final exam
- Possibly with different names: `1511 fetch-pracexam` will be `1511 fetch-exam`

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

Short Answer Questions

Quick Questions (1 mark per question)

- 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 - How To

- You will fetch a file called `exam_mc.txt` to answer them in
- This file is in a special format
- Type your answers within the `{{{` triple curly brackets `}}}`
- Only answers in the `{{{` triple curly brackets `}}}` will be accepted
- Some questions will have validation (so, you might only be able to answer with the letters ‘A’, ‘B’, ‘C’, ‘D’ for example)
- It will have the same structure as the file `prac_mc.txt` in the week 10 lab.

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 at least one question 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

Questions 1-2 First Hurdles

Basic C Programming - similar to Weekly Test question 1 (15 marks each)

- Create C programs
- Use variables (ints and doubles)
- scanf and printf
- if statements and loops
- Basic use of arrays of ints/doubles (q1)
- Basic use of linked lists of ints/doubles (q2)

Example Question 1

Loop through a 1D or 2D 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 side_length, int numbers[SIZE][SIZE])`

```
% ./evens
13 14 15
16 17 19
21 23 25
[Ctrl + D]
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 (15 marks each)

- 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
- Some simple insertion/removal
- Working with Arrays (q3)
- Working with Linked Lists (q4)

Questions 5-6

Even Harder C - similar to Weekly Test question 3 (10 marks each)

- 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
- Might use an Abstract Data Type
- Again, more complex combinations, and some questions requiring interesting problem solving

Question 7

Challenge Questions for people chasing HDs (10 marks)

- 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 the difficulty level
- Partial completion of this question will award some marks

How important are different topics?

Important

- Variables, If, Looping, Functions, Arrays, Linked Lists

Things that you will need to understand the important topics

- Characters and Strings, 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 错别字
- Results should be ready by around the 1st September

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; or you are unable to continue due to circumstances out of your control, let us know via the provided email address (**cs1511.exam@cse.unsw.edu.au**).
- A supplementary exam will be held between the 7th and 11th September. If you think you will need to sit this exam, make sure you are available.

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 Allocation
- 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
  - and others . . .
- 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

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 power;
    // Initialising the variable
    power = 7;
    // Assign the variable a different value
    power = GOKU;

    // we can also Declare and Initialise together
     int powerTwo = 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

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

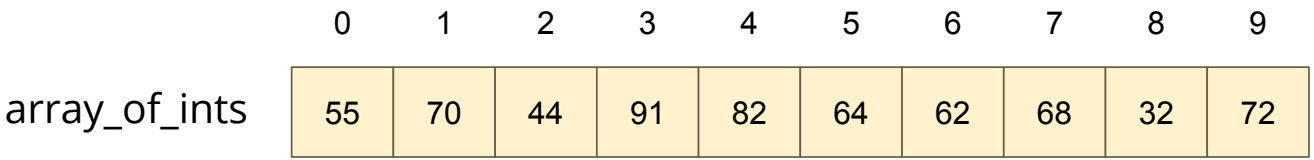
Looping through Linked Lists is also very common

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

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

    // increment the current pointer
    current = current->next;
}
// When current 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

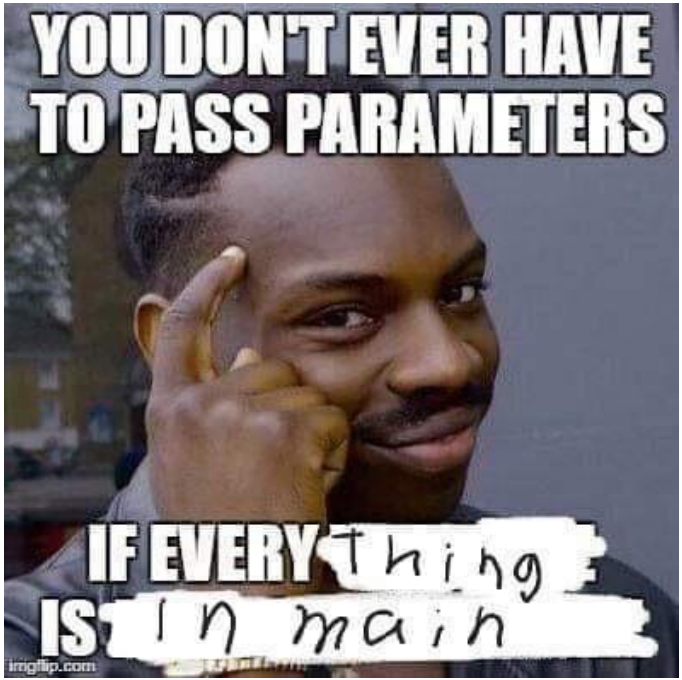
```
int main (void) {
    // declare an array, all zeroes
    int marks[10] = {0};

    // 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;
```

0-9 没有 10.

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



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;
}
```

address

100

101

address 不變

101