COMP1521
WEEK 1

intro to the course!

Siyu (Annie) Qiu

Introductions

4th year Comp Eng.

4 term year teaching COMP1521

Introductions

Let’s get to know each other!

- Name (preferred name)
- Degree
- Year
- Share something about yourself!

Admin

- Tutor contact: siyu.qiu1@student.unsw.edu.au
- Course contact: cs1521@cse.unsw.edu.au

- Tute/lab attendance is optional but highly recommended.

- You can complete the lab in your own time

- Please join **EdForum**!!

- Help session!!!

Assessment

- Weekly Tests (from wk3): 10%
  - Each test is worth 1.7 marks.
  - best 6/8 test marks
- Labs (from wk1): 15%
  - 9 labs totally
  - DUE:12:00 (midday) Monday
  - each week are worth in total 2 marks.
  - Can get 95% by doing all regular exercises
- Assignments
- Final Exam

Announcements 

- Lab start this week!
  - Lab1 has been released, due on Week3 Monday midday
- No weekly quiz this week!
- Feel free to email me!

For this tutorial!!

- 15 - 20 minutes of content revision first

- 30 - 40 minutes of working through questions
  - Try to cover all relevant content, especially exam/ assignment relevant ones.
  - Lots of useful tips and tricks!

Pointers revision

- **&** is the **reference** operator - it gives you the **reference** to an object
- ***** is the **dereference** operator - it takes a pointer, **dereferences** it and gives you the value of the object

& -> value to pointer

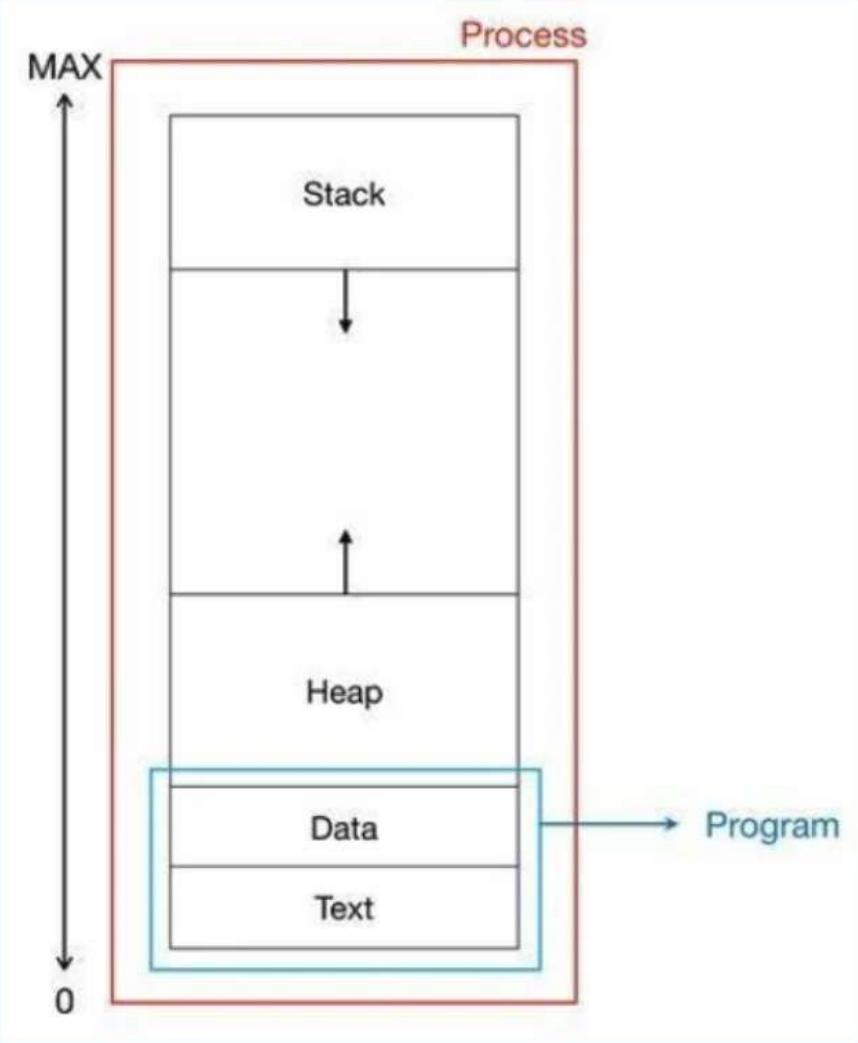
lets analyse this code!

```c
#include <stdio.h>
void test_function(int *pointer_to_number);

int main() {
    int number = 36;
    // nmonic: & = address!
    int *pointer_to_number = &number;
    // think of * as an operator: so (*pointer_to_number) is of type int.
    printf("pointer_to_number: %d\n", pointer_to_number);
    printf("*pointer_to_number: %d\n", *pointer_to_number);
    test_function(pointer_to_number);
    return 0;
}

void test_function(int *pointer_to_number) {
    printf("function pointer_to_number: %d\n", pointer_to_number);
    printf("function *pointer_to_number: %d\n", *pointer_to_number);
}
```

Output:

pointer_to_number: -144215296
* pointer_to_number: 36

Function pointer_to_number: -144215296
Function * pointer_to_number: 36



Memory

- The code (text) segment contains the program instructions e.g. functions.
- The data segment contains global and static variables, and string literals.
- The heap contains dynamically allocated memory (think malloc()).
- The stack contains local variables.

example:

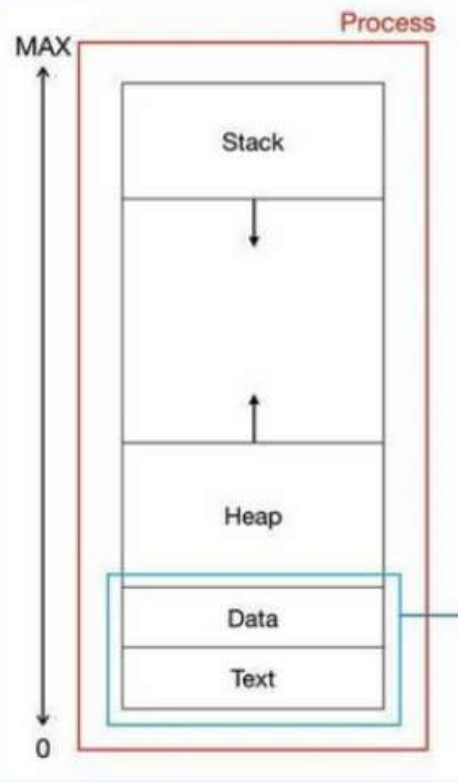
```c
#include <stdlib.h>

int globalCounter = 0;

int main(void)
{
        int i = 0;
        char *message = "Hello world!\n";
        float floatArray[10];
        int *pointer = malloc(400);
}
```

Credit to Dong Zhu Huang for content on this slide and next slide

Answers to previous slide

```c
#include <stdlib.h>

int globalCounter = 0;

int main(void)
{
        int i = 0;
        char *message = "Hello world!\n";
        float floatArray[10];
        int *pointer = malloc(400);
}
```

- globalCounter is a global variable, so it's stored in data segment.
- main() is a function, and its instructions are stored in the code segment.
- i is a local variable so it's in the stack



- char *message = "Hello world!\n" has 2 allocations
  - "Hello world" is a string literal, so it's in the data segment.
  - message is a pointer. It points to "Hello world" in the data segment, but the pointer itself is a local variable, and hence it's in the stack.
- int *pointer = malloc(400) has 2 allocations:
  - The 400 bytes allocated by malloc() is in the heap.
  - pointer is a local variable that points to the 400 byte memory region returned by malloc(), but the pointer itself is in the stack.

getchar

- Q5
- Use `man 3 getchar` to look at the manual entry.

RECURSION

- Recursion is when a function calls itself to solve a smaller part of a problem.
- **Key Concepts:**
  - ✅ **Base Case** – a condition where the function stops calling itself
  - 🔁 **Recursive Case** – where the function calls itself with a smaller input

- Tut question 7

- For, while loop -> tut question 6

What Are argc and argv?

(Q8)

- **argc:** Number of command-line arguments, including the program's name.

- **argv:** Array of pointers to the arguments.
  - argv[0]: Program name.
  - argv[1] to argv[argc-1]: Command-line arguments.

```c
#include <stdio.h>

int main(int argc, char *argv[]) {
    printf("argc=%d\n", argc);
    for (int i = 0; i < argc; i++) {
        printf("argv[%d]=%s\n", i, argv[i]);
    }
    return 0;
}
```

```
$ dcc -o print_arguments print_arguments.c
$ ./print_arguments I love MIPS
```

Output:

```
argc=4
argv[0]=./print_arguments
argv[1]=I
argv[2]=love
argv[3]=MIPS
```

Quick linux commands

- compile: dcc -o hello hello.c
- run: ./hello
- make folder: mkdir week1
- access folder: cd week1
- access previous folder: cd .. / cd ~
- make a file: code hello.c
- open a folder: code .
- remove a file: rm hello.c
- remove a directory: rm -r week1
- use tab to autofill file names!

If have time -> Extra questions

talking about mipsy -> challenge exercise

Questions and Answers