COMP1521 Week 1

Welcome!!!

Feel free to chat in the chat/open mic/turn on cams, we shall start at 5:05pm

Icebreaker questions to prep for

- Highlight of the day?
- Courses you are doing
- Most recently watched youtube video 👀
- Interesting fact

Admin stuff

- Course website (https://cgi.cse.unsw.edu.au/~cs1521/23T2/)
- HELPLINES:
  - course forum
  - course admins cs1521@cse.unsw.edu.au
  - me (z5362344@unsw.edu.au)
- Tutorial files (https://github.com/abbylxt/COMP1521-23T2-Tut)
- Email is the best form of contact to me (response rate is within 24hrs normally, max 48hrs)

Overview

- Ice breaker
- How C programs store data in memory
- Revise C

Ice breaker

Ice breaker

1. Introduce yourself to everyone in the class
    a. preferred name
    b. Choose one of the below
        i. Worst/best tutorial experience you had
        ii. Highlight of the day?
        iii. Most recently watched youtube video 👀
        iv. Interesting fact 😄😎
2. Add to our drawing

Little self intro

Name: Abby

Degree: Food Science/CompSci

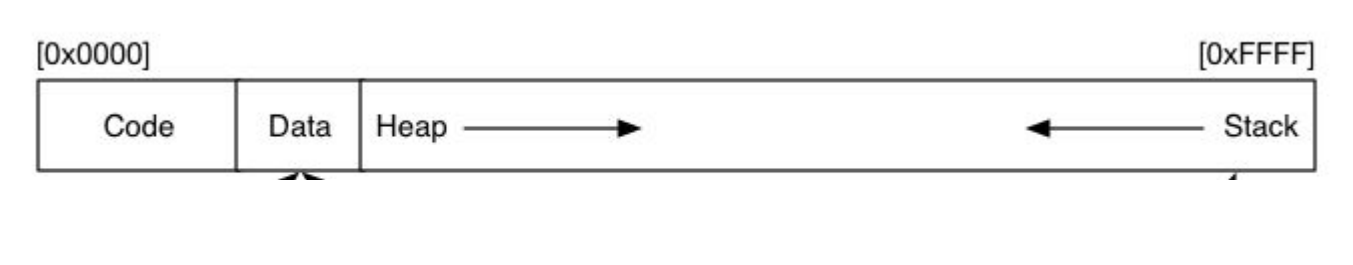
How C programs store data in memory

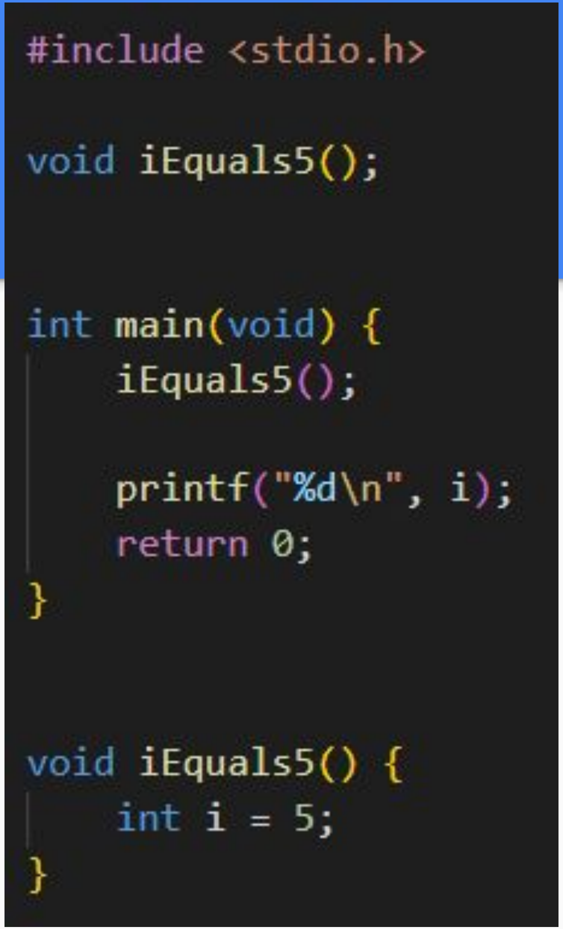
C vs mips/assembly memory

- C manages memory allocation for us, in the sense that it decides where to store data used for the programs in the computer
- In mips you have to do it yourself (manually)
- Hypothetically you can store it anywhere you like, but in the context of this course will be following a similar structure to C programs

Memory in C

- Most programs we write first year uni courses the data is short-lived
- Therefore most of the data will be stored in the RAM (Random-access memory)
- RAM is essential a huge array which is divided into 4 segments: Text, Data, Heap, Stack





```c
#include <stdio.h>

void iEquals5();


int main(void) {
    iEquals5();

    printf("%d\n", i);
    return 0;
}


void iEquals5() {
    int i = 5;
}
```

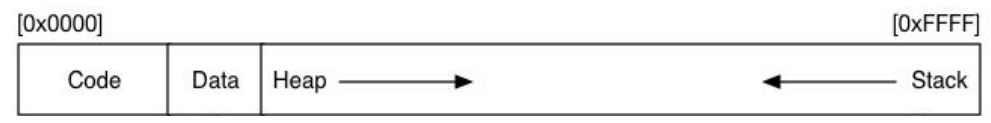
stack
{

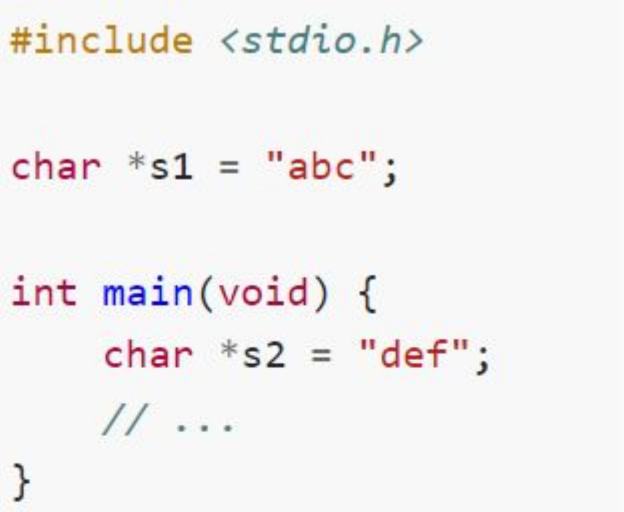
Revision C

Q1

What is the difference between s1 and s2 in the following program fragment? (1pt)

Where is each variable and the strings located in memory? (1pt)





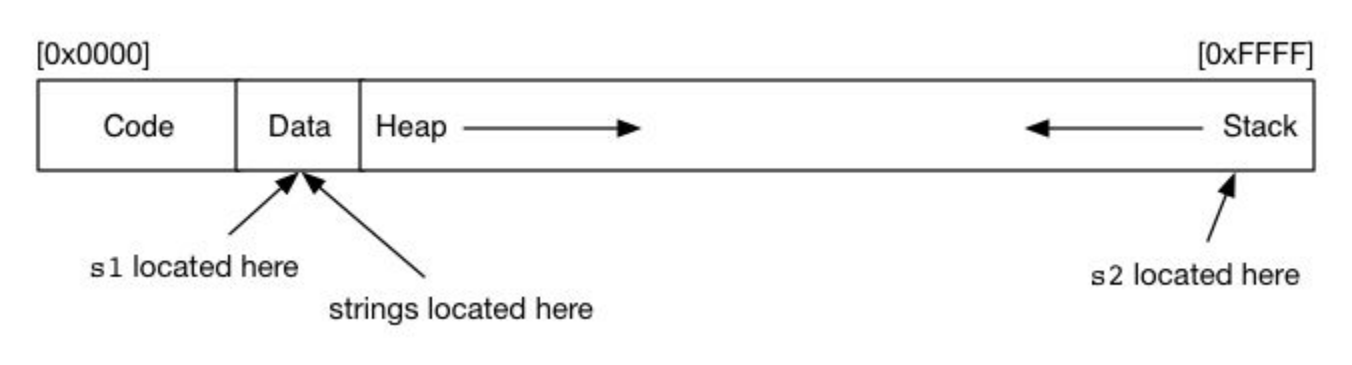
```
#include <stdio.h>

char *s1 = "abc";

int main(void) {
    char *s2 = "def";
    // ...
}
```

Global Variable

- The s1 variable is a global variable
- Accessible from any function in this .c file
- Accessible from other .c files that referenced it as an extern'd variable.
- C implementations typically store global variables in the data segment (region of memory).

Local Variable

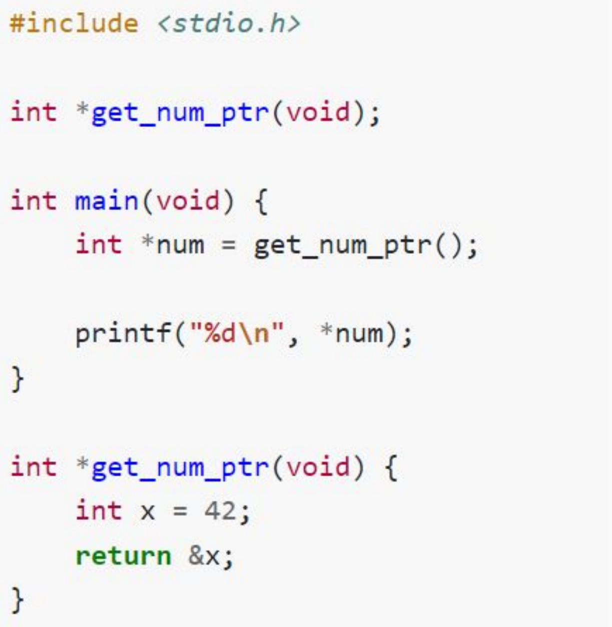
- The s2 variable is a local variable
- Only accessible within the main() function.
- C implementations typically store local variables on the stack, in a stack frame created for function — in this case, for main().



Q2

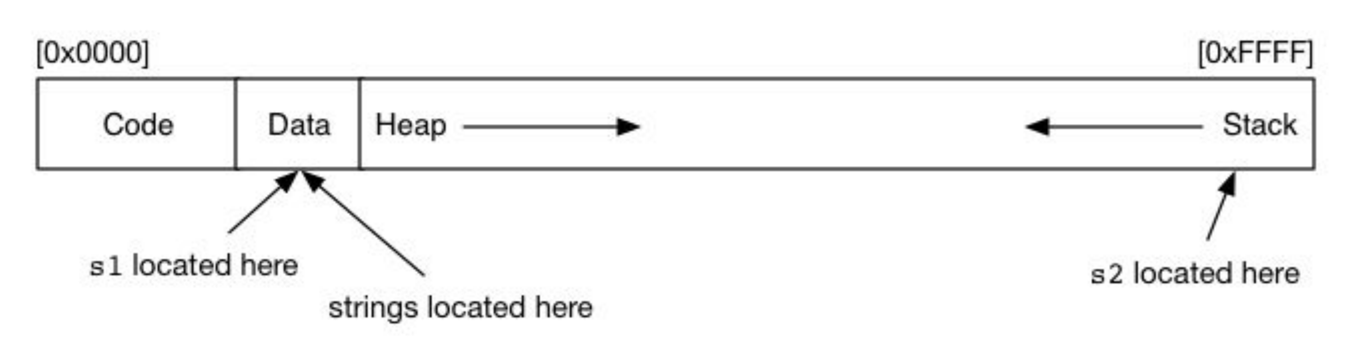
What is wrong with the following code? (1pt)

If we still want get_num_ptr to return a pointer, how can we fix this code? (1pt)



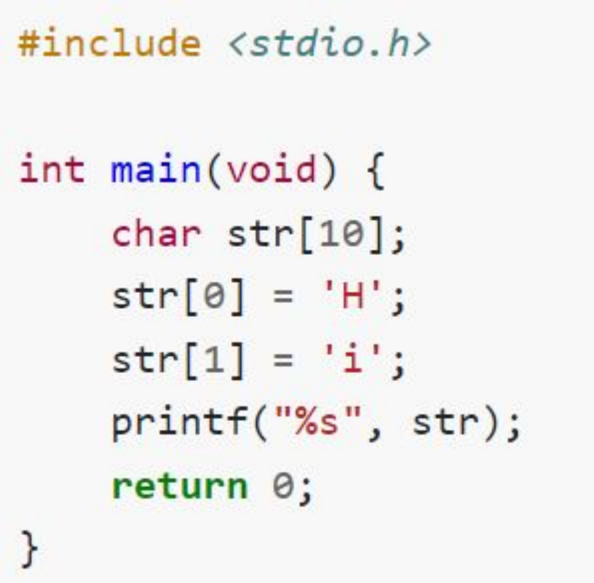
```
#include <stdio.h>

int *get_num_ptr(void);

int main(void) {
    int *num = get_num_ptr();

    printf("%d\n", *num);
}

int *get_num_ptr(void) {
    int x = 42;
    return &x;
}
```

In context of memory



Q3

How can you fix this program?
(1pt)



```c
#include <stdio.h>

int main(void) {
    char str[10];
    str[0] = 'H';
    str[1] = 'i';
    printf("%s", str);
    return 0;
}
```

What would have happened if you ran the code?

Many C library functions like printf expects strings to be null-terminated (indicates the end of the string). Therefore it will try and read the string until it reaches ‘\0’

DCC: Code produced by dcc will then stop with an error because str[2] is uninitialized.

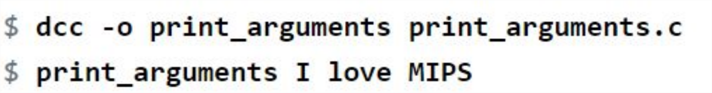
GCC: The code with gcc will keep executing and printing element from str until it encounters one containing '\0'. Often str[2] will by chance contain '\0' and the program will work correctly.

Another common behaviour will be that the program prints some extra "random" characters.

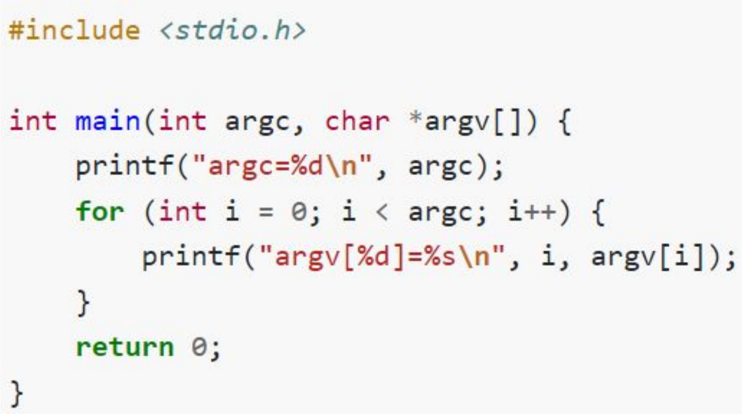
Q4

In the following program, what are argc and argv? (1pt)

What will be the output of the following commands?



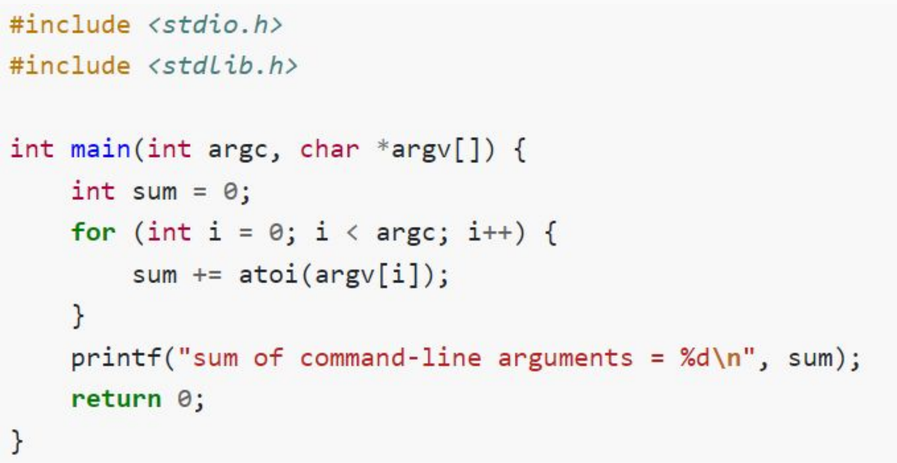
```
$ dcc -o print_arguments print_arguments.c
$ print_arguments I love MIPS
```



```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;
}
```

Q5

Why do we need the function atoi in the following program?



```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
    int sum = 0;
    for (int i = 0; i < argc; i++) {
        sum += atoi(argv[i]);
    }
    printf("sum of command-line arguments = %d\n", sum);
    return 0;
}
```

Q6

For each of the following commands, describe what kind of output would be produced:

```
clang -E x.c

clang -S x.c

clang -c x.c

clang x.c
```

Clang commands

1. `clang -E x.c`
   Executes the C pre-processor, and writes modified C code to `stdout` containing the contents of all `#include`'d files and replacing all `#define`'d symbols.
2. `clang -S x.c`
   Produces a file `x.s` containing the assembly code generated by the compiler for the C code in `x.c`. Clearly, architecture dependent.
3. `clang -c x.c`
   Produces a file `x.o` containing relocatable machine code for the C code in `x.c`. Also architecture dependent. This is not a complete program, even if it has a `main()` function: it needs to be combined with the code for the library functions (by the linker ld).
4. `clang x.c`
   Produces an executable file called `a.out`, containing all of the machine code needed to run the code from `x.c` on the target machine architecture. The name `a.out` can be overridden by specifying a flag `-o filename`

Intro to MIPS

mipsy and mipsy_web fun facts

mipsy

- Is a mipsy emulator which simulates the execution of a MIPS CPU and lets you run MIPS assembler on any computer (regardless of native architecture).
- developed at CSE by Zac Kologlu
- You can run mipsy on CSE with the command: `1521 mipsy`

mipsy_web (https://cs1521.web.cse.unsw.edu.au/mipsy/)

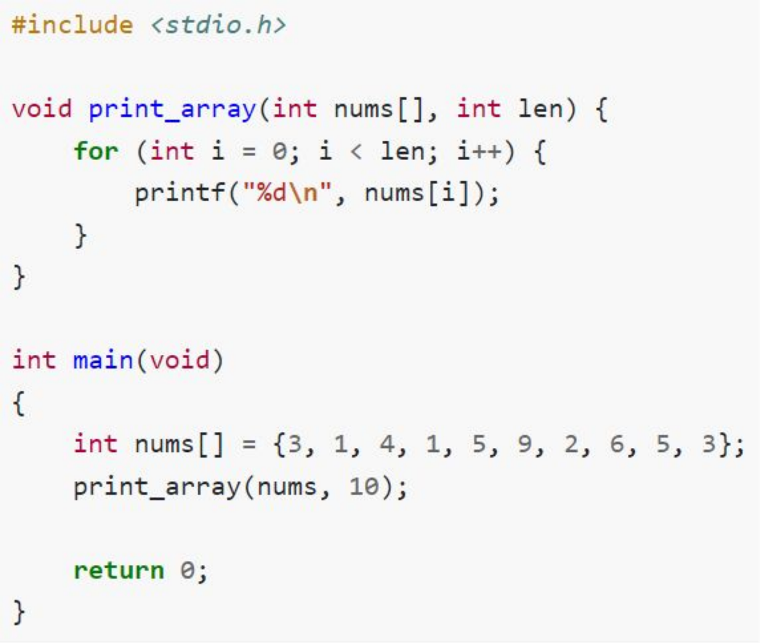
- A web-based version of mipsy that is still in very early stages
- Able to run a full MIPS simulator visually in any web browser

Translate a simple c program into mips 👀

https://cgi.cse.unsw.edu.au/~cs1521/23T2/tut/01/questions

Extra Q1

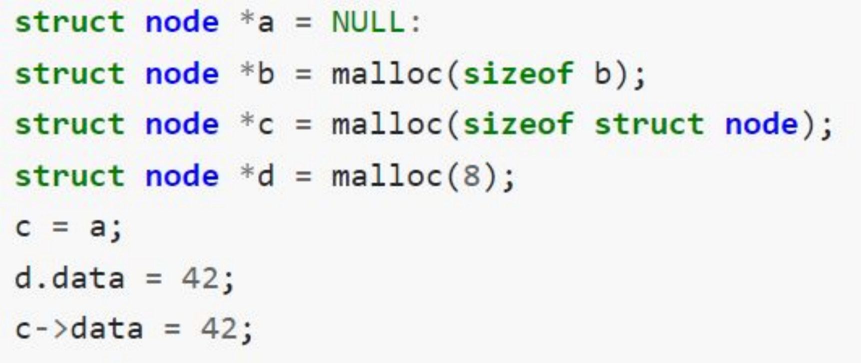
(Code) Rewrite this program using a recursive function (4pts)



```c
#include <stdio.h>

void print_array(int nums[], int len) {
    for (int i = 0; i < len; i++) {
        printf("%d\n", nums[i]);
    }
}

int main(void)
{
    int nums[] = {3, 1, 4, 1, 5, 9, 2, 6, 5, 3};
    print_array(nums, 10);

    return 0;
}
```

Extra Q1

Find the errors in this code (each error is 1pt)



```
struct node *a = NULL:
struct node *b = malloc(sizeof b);
struct node *c = malloc(sizeof struct node);
struct node *d = malloc(8);
c = a;
d.data = 42;
c->data = 42;
```

Extra Q2

(Code)

Define a struct that might store information about a pet.

- The information should include the pet's name, type of animal, age and weight. (2pts)
- Create a variable of this type and assign information to it to represent an axolotl named "Fluffy" of age 7 that weighs 300grams. (2pts)

Code: Pineapples