COMP1521 Week 1

Welcome!!!

Please sit down we shall start at 1:05pm

Try and disperse yourselves amongst the tables

Icebreaker questions

- Highlight of the day?
- What is the favourite course you have taken?
- Something interesting that happened over the holidays 👀

Admin stuff

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

Overview

- admin
- Ice breaker
- bingo
- c memory + revision

Ice breaker

Ice breaker

1. Introduce yourself to everyone in the class
    a. name + pronouns(optional)
    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. fun fact

Little self intro

Name: Abby

Degree: Food Science/CompSci

Worst/best tutorial experience you had:

best: Not when I was a student but as an lab assist one of tutors spent an hour explaining in depth how unicode worked to a student to make sure they knew how it worked

worst: Every week they would take ½ hour to get bb to work and start the tutorial

Intro bingo

rules

- You can only write down one person once on the bingo
- Once you get 4 people in a row shout out bingo

have/has applied to subcom this year	prefers matcha over coffee	first and last name starts with the same letter as you	went overseas last holidays
has cried when their favourite character died in a series	has their p's	uses google calender	has an uniqlo airism tshirt or a cresent bag
has taken the light rail to the wrong end of unsw	has tripped/fallen over on public transport	owns a pair sony headphones	has tried upper campus pho
takes them less than 45 mins to get to uni	uses notion	can speak more than one language	gone to the wrong room for class

How today’s Revision Trivia will work

As today is mostly revision of C and transferring the understanding of memory in C to mips, I’ve decided to create trivia is so you can meet your classmates and revise at the same time :DDD

- I will show a question on the board
- On the count of three I will unlock the buzzer
- If you get the question wrong the next group on the list will get to answer
- Questions starting with (Code) so having a code editor up and ready will be convenient

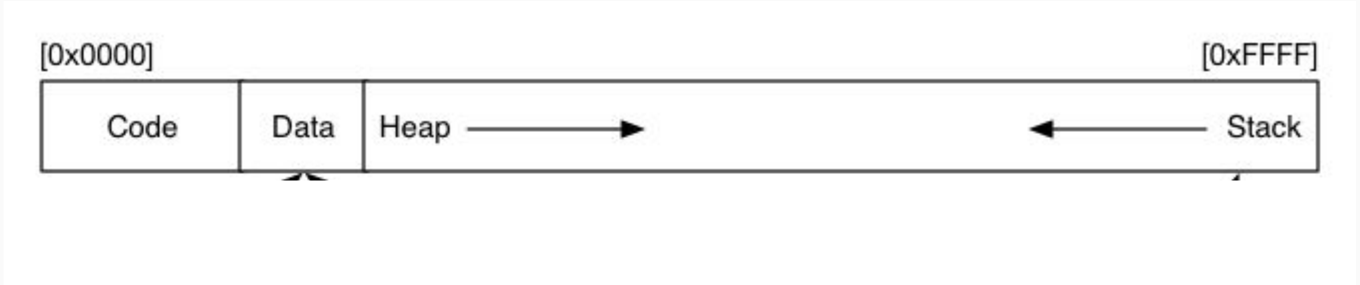
How C programs store data in memory

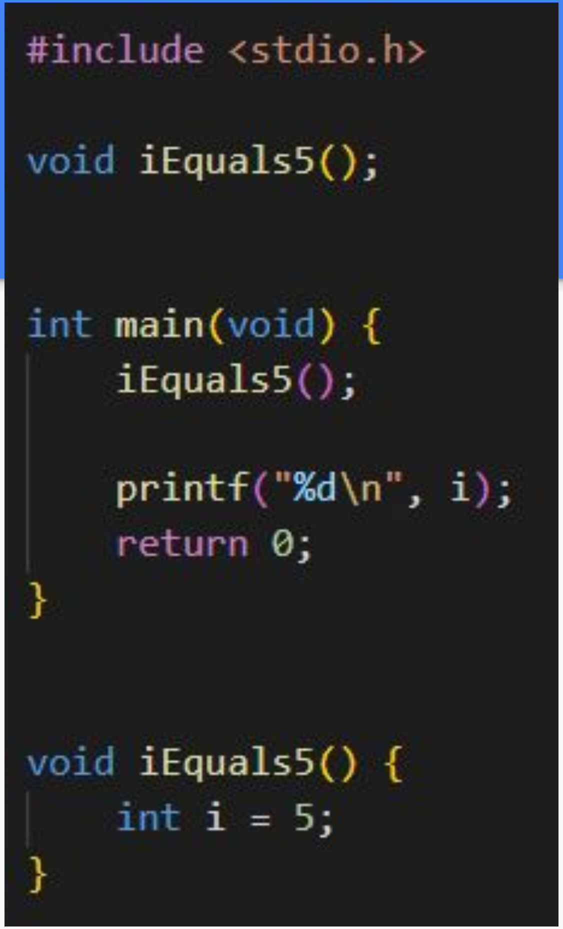
C vs mips/assembly memory

- 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





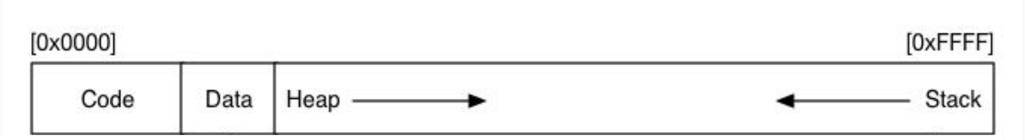
```c
#include <stdio.h>

void iEquals5();


int main(void) {
    iEquals5();

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


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

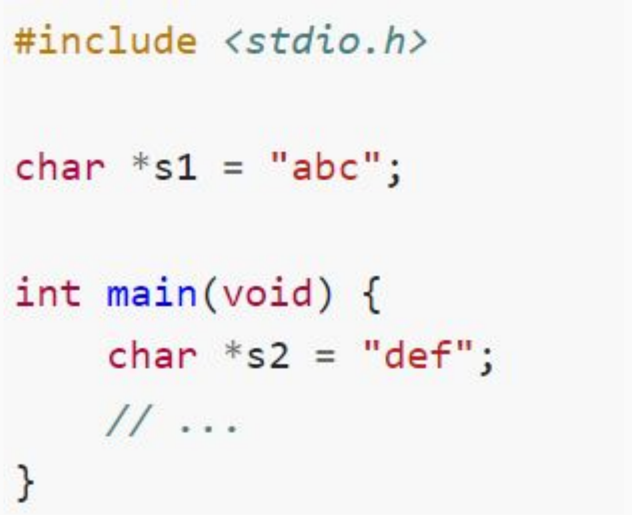
Revision C

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)





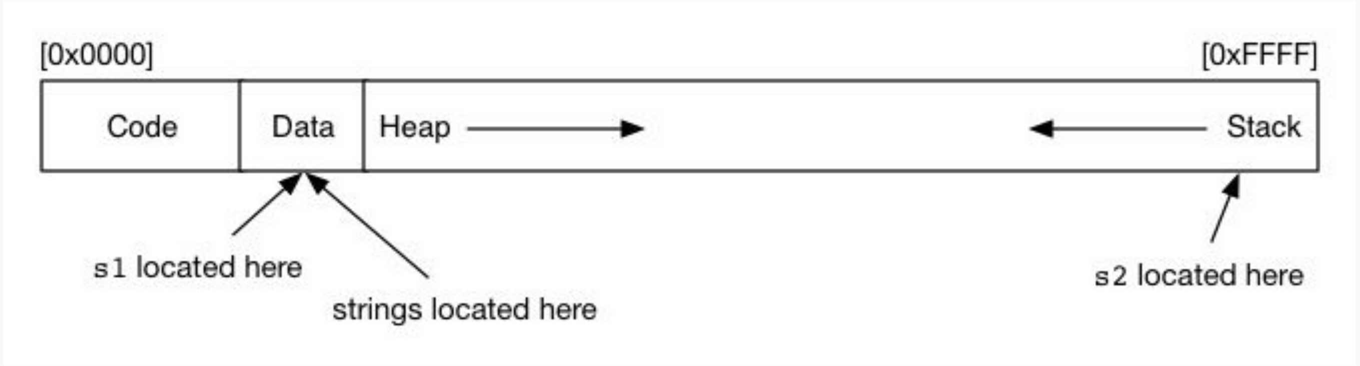
```
#include <stdio.h>

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

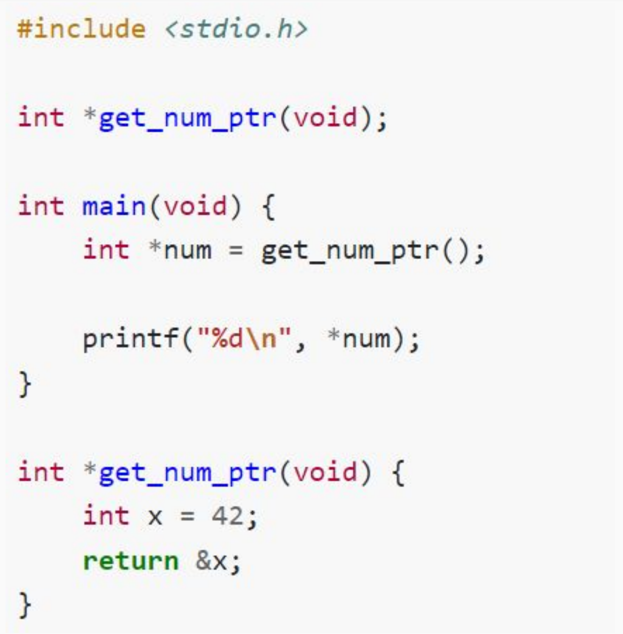
- The s2 variable is a local variable
- Only accessible within the main() function.
- C implementations typically store local variables on the stack, in a stack frame created for function — in this case, for main().



Q2

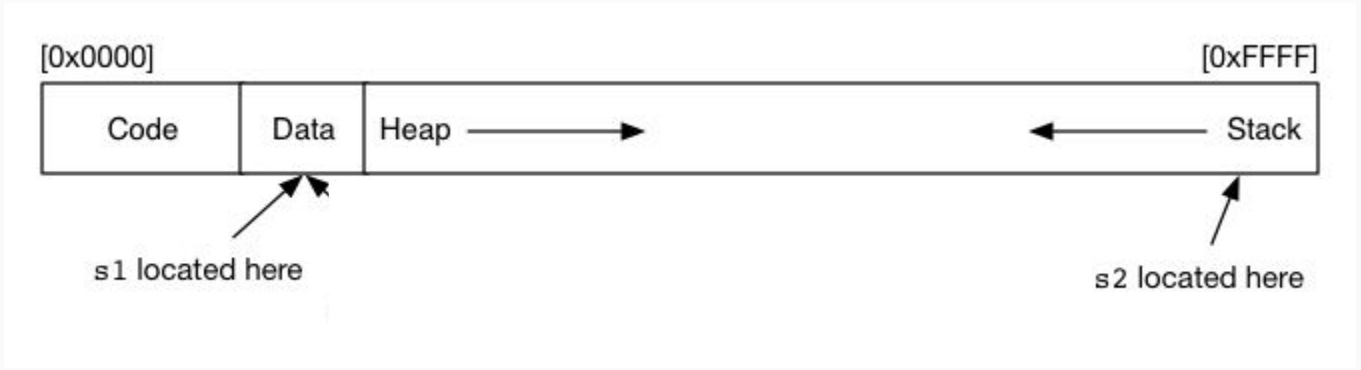
What is wrong with the following code? (1pt)

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



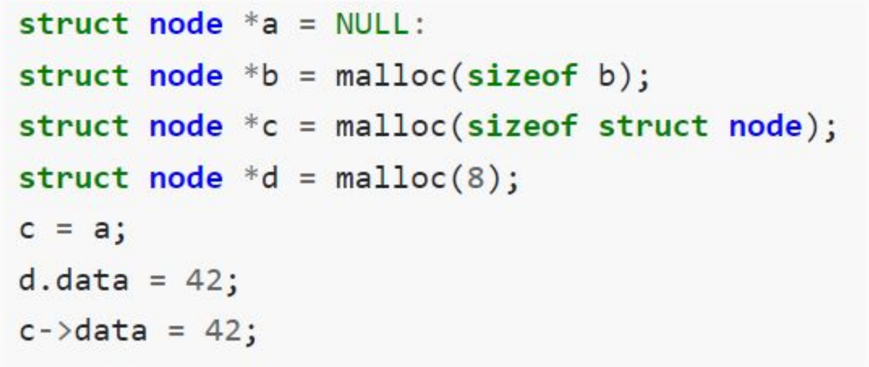
```c
#include <stdio.h>

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

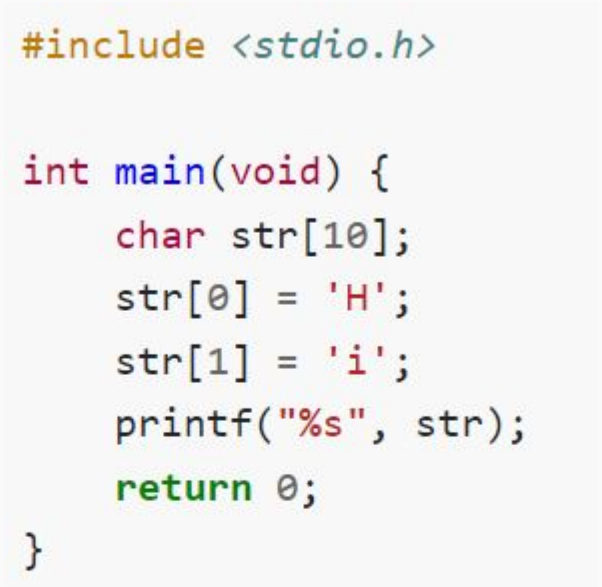
Find the errors in this code (each error is 1pt)



```c
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;
```

Q4

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.

Q5

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

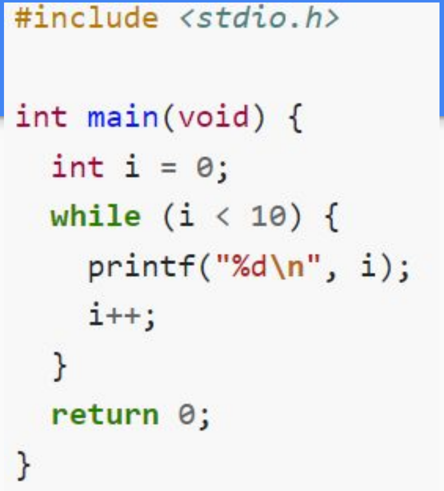
What will be the output of the following commands (1pt)?

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

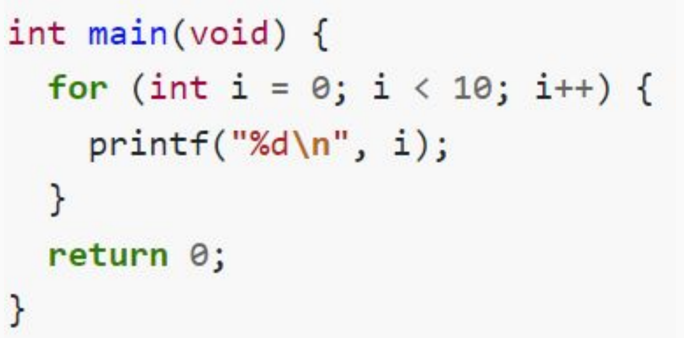
```c
#include <stdio.h>

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;
}
```

Q6

Here I have rewritten a while loop using a for loop. What subtle difference would there be between the two programs?



```c
#include <stdio.h>

int main(void) {
  int i = 0;
  while (i < 10) {
    printf("%d\n", i);
    i++;
  }
  return 0;
}
```



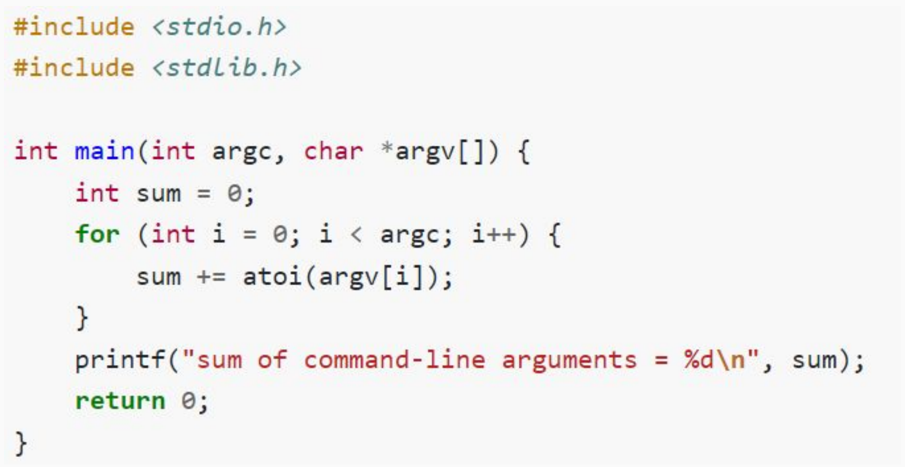
```c
int main(void) {
  for (int i = 0; i < 10; i++) {
    printf("%d\n", i);
  }
  return 0;
}
```

Q6

The subtle difference between the two programs is that the variable, i, is only accessible within the for loop, while in the while loop, i, is accessible outside the loop. This is a result of scope. In the while loop, i, is declared outside the loop, so it is accessible outside the loop. In the for loop, i, is declared within the loop, so it is only accessible within the loop.

Q7

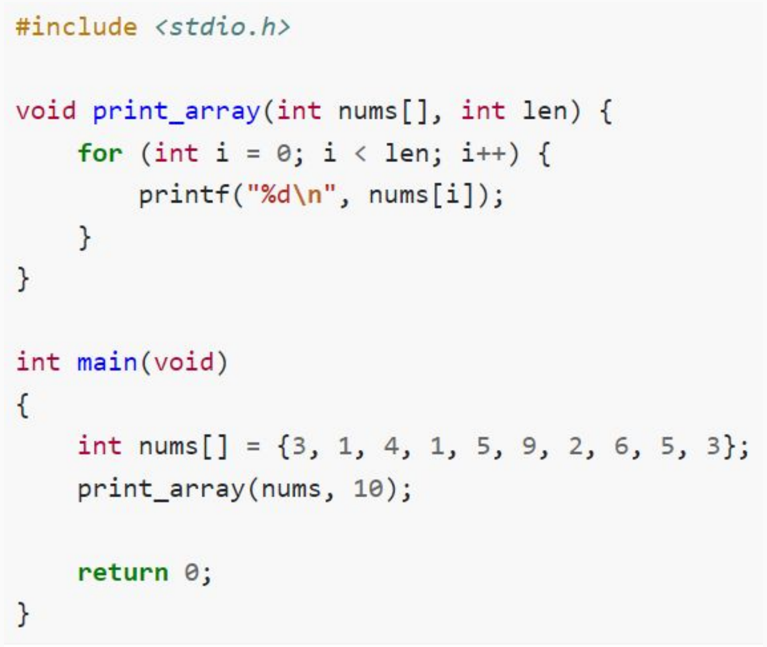
Why do we need the function atoi in the following program?



```c
#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;
}
```

Q8

(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;
}
```

Q9

clang is also an compiler similar to dcc

What output will be produced by these commands? (1pt for each)

- clang -E x.c
- clang -S x.c
- clang -c x.c
- clang x.c

Q9

```
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.

```
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.

Q9

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).

clang x.c

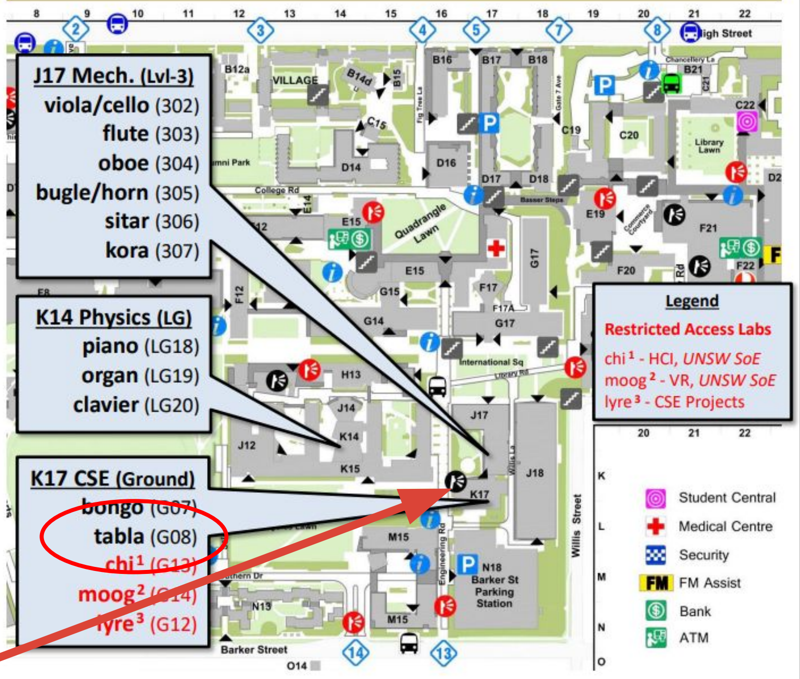
Produces an executable file called a.out, containing all of the machine code needed to run the code from x.c on the target machine architecture. The name a.out can be overridden by specifying a flag -o filename.

UNSW SYDNEY | Australia's Global University

School of Computer Science and Engineering

CSE Teaching Lab Locations

Kensington Campus



J17 Mech. (Lvl-3)
viola/cello (302)
flute (303)
oboe (304)
bugle/horn (305)
sitar (306)
kora (307)

K14 Physics (LG)
piano (LG18)
organ (LG19)
clavier (LG20)

K17 CSE (Ground)
bongo (G07)
tabla (G08)
chi[1] (G13)
moog[2] (G14)
lyre[3] (G12)

Legend
Restricted Access Labs
chi[1] - HCI, *UNSW SoE*
moog[2] - VR, *UNSW SoE*
lyre[3] - CSE Projects

🟪	Student Central
✚	Medical Centre
▨	Security
FM	FM Assist
$	Bank
▣	ATM

Source: http://www.facilities.unsw.edu.au/sites/all/files/Campus%20Map.pdf

LAB ON GROUND LEVEL

LEFT TO THE LIFTS

Code: Pineapples