Welcome!

COMP2521 25T2
Data Structures and Algorithms

COMP2521 25T2
Introduction

Sim Mautner
cs2521@cse.unsw.edu.au

course introduction
tools of the trade

Course Goals

to get you thinking like a *computer scientist*
not just a programmer

- know and understand *fundamental* data structures, algorithms
- reason about *applicability* + *effectiveness*
- analyse behaviour/correctness of programs

Assumed Knowledge

We assume that you can:

- Produce a correct C program from a specification
- Use fundamental control structures
  (sequence, selection (`if`), iteration (`while`))
- Use fundamental C data types and data structures
  (`char`, `int`, `double`, arrays, `structs`, pointers, linked lists)

Acquired Knowledge

In this course, you will learn:

- data structures: trees, graphs, hash tables, tries
- data structure/algorithm analysis: time/space complexity
- sorting and searching techniques
- graph algorithms

Acquired Skills

By the end of this course, you should be able to:

- Implement solutions to a wider range of problems
- Analyse performance characteristics of algorithms
- Analyse performance characteristics of data structures
- Make decisions about appropriate data structures and algorithms

Course Staff

Convenor	John Shepherd
Lecturer	Sim Mautner
Admins	Kevin Luxa, Ethan Brown and Ryan Berlee

Contact

Website `https://webcms3.cse.unsw.edu.au/COMP2521/25T2/`

Email cs2521@cse.unsw.edu.au

Teaching/Assessment Methods

Lectures
Tutorials
Labs
Quizzes
Assignments
Exam

Lectures

Four hours of lectures per week

- Monday 11:00–13:00; Wednesday 14:00–16:00
  - In person in Keith Burrows Theatre
  - Also livestreamed via YouTube
  - Link to livestream on the lectures page
  - Feel free to ask questions in the chat
  - Recordings will be on YouTube
- present a brief overview of theory
- demonstrate problem-solving methods
- give practical demonstrations

Tutorials

Weekly one-hour tutorials

- tutorials start in week 1
  - run every week, except flex week
  - online classes are via Blackboard Collaborate
- tutorials clarify lecture material
- work through problems related to lecture topics
- questions available (usually) the week before
- answers available Friday evening

To get the best out of tutorials

- read and attempt the problems yourself beforehand
- don't keep quiet in tutorials... talk, discuss, ...
- ask if you don't understand something

Labs

Each tutorial is followed by a two-hour lab class

- several exercises, mostly small implementation/analysis tasks
- aim to improve your coding and analysis skills
- give you experience applying algorithms and techniques
- done individually, unless specified
- submitted via `give`, before Monday ~~17:00~~ 09:00 the following week
- many labs have a handmarking component (see spec for details)
  - handmarking completed by showing your work to your tutor in the lab **within two weeks of the lab**
- worth 15% of your final mark, best 7 of 8 labs used to calculate the 15%

Quizzes

Weekly quizzes

- on WebCMS
- questions about previous week's lectures
- different kinds of questions
  - multiple choice, multiple select, fill-in-the-blank...
- aim to test your knowledge and understanding of the theory
- done individually
- due Monday ~~17:00~~ 09:00 the following week
- can submit multiple times, only last submission is marked
- worth 10% of your final mark, best 7 of 8 quizzes used to calculate the 10%

Assignments

Two assignments

- each worth 15% of your final mark
- give you experience applying algorithms to larger problems
- done individually
- help sessions will be available to assist with assignments
  - will be very busy in the last days before an assignment is due

Late Penalties

Labs, quizzes and assignments all have the same late penalty

- UNSW standard late penalty
- **0.2%** of the maximum mark taken from your raw mark for each hour late
  - equivalent to 4.8% per day
- submissions later than 5 days not allowed (automatically enforced)

Sample Solutions and Marking

Due to the UNSW standard late penalty allowing late submissions up to 5 days after the deadline, along with extensions for special consideration:

- sample solutions for labs will be released 12 days after the due date
- marks for labs will be released a week after the due date
- answers and marks for quizzes will be released 5 days after the due date
- sample solutions for assignments are not released
- marks for assignments are released in two parts
  - automarking will be released a week after the due date
  - handmarking (style, automarking adjustments) takes longer and will be released 2 weeks after the automarking

Final Exam

- 3 hour in-person exam, during exam period
- Limited resources provided
- Two sections:
  - Theory - short-answer questions
  - Programming - programming questions
- To pass the exam hurdle, you must:
  - score at least 18/45 (40%) on the final exam
  - score at least 25% in the theory section of the exam
  - score at least 25% in the programming section of the exam

Special Consideration

- Have you been impacted by unforeseen adverse circumstances?
- Has it affected your ability to complete coursework?
- You can apply for special consideration via myUNSW
- Find out how to apply here:
  `https://student.unsw.edu.au/special-consideration`

Assessment

Summary:

15% labs
10% quizzes
15% assignment 1
15% assignment 2
45% final exam

Assessment

To pass COMP2521, you must:

- score at least 50/100 overall
- pass the exam hurdle

Academic Integrity

- Labs, quizzes and assignments must be entirely your own work
- Plagiarism will be checked for and penalised
- Plagiarism may result in suspension from UNSW
- Scholarship students may lose their scholarship
- International students may lose their visa
- Supplying your work to any other person may result in loss of all your marks for the lab/assignment

Academic Integrity

Generative AI Tools

- Use of generative AI tools, e.g., GitHub Copilot, ChatGPT, with the intention of generating answers/solutions for assessment tasks is not permitted
- Use of generative AI tools for learning is permitted
  - You must still be critical of any response you get from these tools
- Generative AI tools have great potential to assist coders, but use of them requires good understanding of the language/system

Resources

Additional Help

- Discourse forum
- Weekly consultations
  - Starting from week 3
  - Monday 1pm-2pm
  - In person at K17 (room to be confirmed)
  - For clarification of course content
- Help sessions
  - Starting from week 2
  - Schedule on course website
  - For help with labs and assignments

Resources

Additional Exercises

- Attempt lecture excercises by yourself using the `starter` code provided
- Practice exercises by topic: here
- Extra lab exercises
- LeetCode/HackerRank problems for more practice: here

Expectations

What we expect from you

- Check your email regularly
  - Announcements will be sent to your email
  - Your tutor will send you emails
  - Reminders of unsubmitted work will be sent to your email
- Read the spec before asking questions
  - Don't ask questions that are already answered in the spec
- Attempt to debug your program yourself before asking for help
  - Debugging may involve adding print statements or using gdb to check the state of the program at various points, or drawing diagrams to visualise the program's execution

Expectations

What you can expect from us

- Regular announcements/updates
- Lecture slides released before lectures
- Minimal typos/mistakes in lecture slides
- Tutorial questions/lab exercises released on time (by the weekend before)
- Assignments released on time
- Assignments marked on time

Advice

- Keep up with lectures
  Labs and quizzes require you to know content from recent lectures
- Attend tutorials, *especially* if you are falling behind
  Tutors will not judge you for falling behind
- Always try to *understand*, instead of just memorise
  Understanding something makes it easier to remember
  Exam questions will be different from what you've seen
- Programming is a skill that improves with practice
  The more you practice, the easier labs, assignments and the exam will be

Advice

Engage, ask questions, go to consults, do practice exercises...

You can improve if you put in the effort!

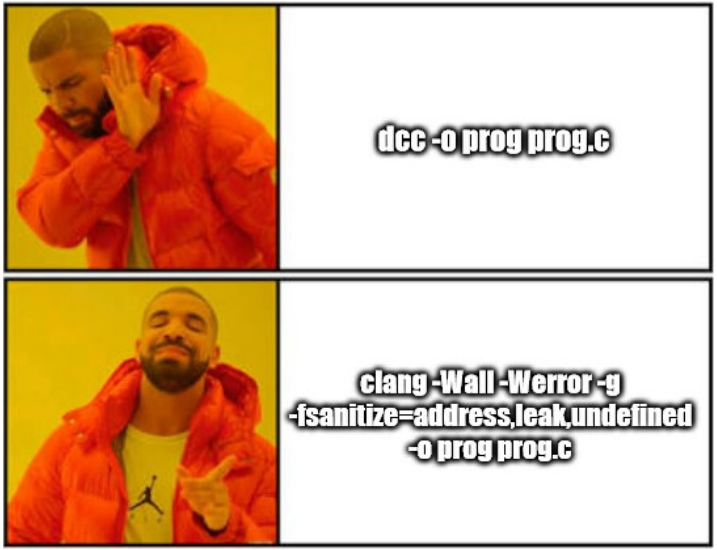
Acknowledgements

COMP2521 material drawn from...

- slides by Jashank Jeremy (COMP2521 19T0)
- slides by Angela Finlayson (COMP2521 18x1)
- slides by John Shepherd (COMP1927 16s2)
- slides by Gabriele Keller (COMP1927 12s2)
- lectures by Richard Buckland (COMP1927 09s2)
- slides by Manuel Chakravarty (COMP1927 08s1)
- notes by Aleks Ignjatovic (COMP2011 '05)
- slides and books by Robert Sedgewick
- Book Cormen, Thomas H.; Lieserson, Charles E.; Rivest, Ronald L.; Stein, Clifford, Introduction to Algorithms (4th ed.). MIT Press and McGraw-Hill, 2022
- slides by Kevin Luxa (COMP2521 25T1)

The Tools of the Trade

Compilation



Compilation

clang

COMP2521 uses the clang compiler. Basic compilation command:

```
clang -Wall -Werror -g -o prog prog.c
```

Compilation

clang

COMP2521 uses the clang compiler. Basic compilation command:

```
clang -Wall -Werror -g -o prog prog.c
```

- `-Wall` enables (almost) all warnings
  - Catches many possible syntax errors

Compilation

clang

COMP2521 uses the clang compiler. Basic compilation command:

```
clang -Wall -Werror -g -o prog prog.c
```

- `-Wall` enables (almost) all warnings
  - Catches many possible syntax errors
- `-Werror` turns warnings into errors
  - Prevents compilation if there are warnings

Compilation

clang

COMP2521 uses the clang compiler. Basic compilation command:

```
clang -Wall -Werror -g -o prog prog.c
```

- `-Wall` enables (almost) all warnings
  - Catches many possible syntax errors
- `-Werror` turns warnings into errors
  - Prevents compilation if there are warnings
- `-g` preserves information useful for debugging
  - Line numbers, function and variable names, etc.

Sanitizers

{Address, Leak, Memory, Thread, DataFlow, UndefinedBehavior}Sanitizer

a family of compiler plugins, developed by Google
which instrument executing code with sanity checks
use-after-free, array overruns, value overflows, uninitialised values, and more

you've been using ASan+UBSan already: *dcc* uses them!
usable on your own *nix systems (Linuxes, BSDs, 'macOS') too!

Sanitizers

AddressSanitizer

- Detects invalid memory accesses, such as:
  - Out-of-bounds array accesses
  - Use-after-free errors
  - Double-free errors
  - ...and many others
- To use AddressSanitizer, compile with `-fsanitize=address`
  - Our Makefiles compile with AddressSanitizer by default

Sanitizers

AddressSanitizer - Example

```c
#include <stdio.h>

#define SIZE 5

int main(void) {
    int arr[SIZE];
    int i = 0;
    while (scanf("%d", &arr[i]) == 1) {
        i++;
    }
    ...
}
```

Sanitizers

AddressSanitizer - Example


```
=================================================================
==2848814==ERROR: AddressSanitizer: stack-buffer-overflow on address 0x7ffc9a6b8b74
at pc 0x00000043ab36 bp 0x7ffc9a6b8a00 sp 0x7ffc9a6b8180
WRITE of size 4 at 0x7ffc9a6b8b74 thread T0
    #0 0x43ab35 in scanf_common(void*, int, bool, char const*, __va_list_tag*) (/imp
ort/glass/2/.../asan+0x43ab35)
    #1 0x43b98b in __isoc99_scanf (/import/glass/2/.../asan+0x43b98b)
    #2 0x4c805f in main /import/glass/2/.../asan.c:9:12
    #3 0x7f0c20c7ed09 in __libc_start_main csu/../csu/libc-start.c:308:16
    #4 0x41e2b9 in _start (/import/glass/2/.../asan+0x41e2b9)

Address 0x7ffc9a6b8b74 is located in stack of thread T0 at offset 52 in frame
    #0 0x4c7f5f in main /import/glass/2/.../asan.c:6

  This frame has 1 object(s):
    [32, 52) 'arr' (line 7) <== Memory access at offset 52 overflows this variable
HINT: this may be a false positive if your program uses some custom stack unwind mec
hanism, swapcontext or vfork
      (longjmp and C++ exceptions *are* supported)
SUMMARY: AddressSanitizer: stack-buffer-overflow (/import/glass/2/.../asan+0x43ab35)
 in scanf_common(void*, int, bool, char const*, __va_list_tag*)
```

Sanitizers

LeakSanitizer

- Detects memory leaks
- To use LeakSanitizer, compile with `-fsanitize=leak`
- Example of error that would be caught by LeakSanitizer:

```c
#include <stdlib.h>

int main(void) {
    int *a = malloc(sizeof(int));
    *a = 42;
    // free(a);
}
```

Sanitizers

MemorySanitizer

- Detects uninitialized memory access
- To use MemorySanitizer, compile with `-fsanitize=memory`
- Example of error that would be caught by MemorySanitizer:

```c
#include <stdio.h>

int main(void) {
    int arr[10];
    arr[0] = 42;
    if (arr[1] == 0) {
        printf("zero\n");
    }
}
```

Sanitizers

UndefinedBehaviorSanitizer

- Detects wide range of undefined behaviours
- To use UndefinedBehaviorSanitizer, compile with `-fsanitize=undefined`
- Example of error that would be caught by UndefinedBehaviorSanitizer:

```c
#include <limits.h>
#include <stdio.h>

int main(void) {
    int a = INT_MAX;
    printf("%d\n", a + 1);
}
```

valgrind

- finding memory leaks
  ... not `free`'ing memory that you `malloc`'d
- finding memory errors
  ... illegally trying access memory

```
$ valgrind ./prog
...
==29601== HEAP SUMMARY:
==29601==     in use at exit: 64 bytes in 1 blocks
==29601==   total heap usage: 1 allocs, 0 frees, 64 bytes allocated
==29601==
==29601== LEAK SUMMARY:
==29601==    definitely lost: 64 bytes in 1 blocks
```

Valgrind doesn't play well with ASan. Compile without ASan if you want to use it.

make

Can't be bothered typing long compilation commands?

make lets you specify
rules, *dependencies*, *variables*
in a Makefile
to define what a program needs to be compiled

With a Makefile, all you need to do to compile is to type
`make`