Welcome!

COMP2521 24T2
Data Structures and Algorithms

COMP2521 24T2
Introduction

Sim Mautner
cs2521@cse.unsw.edu.au

course introduction
tools of the trade

Slides adapted from those by Kevin Luxa 2521 24T1

Course Goals

Introduction
Outline
People
Teaching
Assessment
Resources
Expectations
Advice
Advice
Feedback
Acknowledgements
Tools

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

Introduction
Outline
People
Teaching
Assessment
Resources
Expectations
Advice
Advice
Feedback
Acknowledgements
Tools

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

Introduction
Outline
People
Teaching
Assessment
Resources
Expectations
Advice
Advice
Feedback
Acknowledgements
Tools

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

Acquired Skills

Introduction
Outline
People
Teaching
Assessment
Resources
Expectations
Advice
Advice
Feedback
Acknowledgements
Tools

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

Introduction
Outline
People
Teaching
Assessment
Resources
Expectations
Advice
Advice
Feedback
Acknowledgements
Tools

Lecturer	Sim Mautner
Admin	Ethan Brown, Kevin Luxa
Tutors	`https://cgi.cse.unsw.edu.au/~cs2521/24T2/team`

Contact

Introduction
Outline
People
Teaching
Assessment
Resources
Expectations
Advice
Advice
Feedback
Acknowledgements
Tools

Website `https://webcms3.cse.unsw.edu.au/COMP2521/24T2/`

Email cs2521@cse.unsw.edu.au

Teaching/Assessment Methods

Introduction
Outline
People
Teaching
Assessment
Resources
Expectations
Advice
Advice
Feedback
Acknowledgements
Tools

Lectures
Tutorials
Labs
Quizzes
Assignments
Exam

Lectures

Introduction
Outline
People
Teaching
Assessment
Resources
Expectations
Advice
Advice
Feedback
Acknowledgements
Tools

Four hours of lectures per week

- Monday 11:00–13:00
  - In person in Clancy Auditorium
  - If you're enrolled online, but would like to attend in person, please do, there's plenty of space.
  - Also livestreamed via YouTube
  - Link to livestream on the lectures page
  - Feel free to ask questions in the chat
  - Recordings will be on YouTube
- Thursday 11:00–13:00
  - Online only
  - Otherwise the same as Monday
- present a brief overview of theory
- demonstrate problem-solving methods
- give practical demonstrations

Introduction
Outline
People
Teaching
Assessment
Resources
Expectations
Advice
Advice
Feedback
Acknowledgements
Tools

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

Introduction
Outline
People
Teaching
Assessment
Resources
Expectations
Advice
Advice
Feedback
Acknowledgements
Tools

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 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

Introduction
Outline
People
Teaching
Assessment
Resources
Expectations
Advice
Advice
Feedback
Acknowledgements
Tools

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 the following week
- worth 10% of your final mark, best 7 of 8 quizzes used to calculate the 10%

Assignments

Introduction
Outline
People
Teaching
Assessment
Resources
Expectations
Advice
Advice
Feedback
Acknowledgements
Tools

Two assignments

- each worth 15% of your final mark
- give you experience applying algorithms to larger problems
- done individually
- will *always* take longer than you expect
- don't leave them to the last minute
- help sessions will be available to assist with assignments
  - will be very busy in the last days before an assignment is due

Late Penalties

Introduction
Outline
People
Teaching
Assessment
Resources
Expectations
Advice
Advice
Feedback
Acknowledgements
Tools

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

Introduction
Outline
People
Teaching
Assessment
Resources
Expectations
Advice
Advice
Feedback
Acknowledgements
Tools

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

Introduction
Outline
People
Teaching
Assessment
Resources
Expectations
Advice
Advice
Feedback
Acknowledgements
Tools

- 3 hour in-person exam, during exam period
- Made up of two components:
  - theory : you must demonstrate understanding of the topics taught in the course
  - practical : you must be able to produce C programs to a specification
- You must score at least 18/45 (40%) on the final exam to pass the course
- You must score at least 25% on each of the two components of the exam to pass the course

Final Exam - Hurdle Requirement

Introduction
Outline
People
Teaching
Assessment
Resources
Expectations
Advice
Advice
Feedback
Acknowledgements
Tools

- Hurdle requirements:
  - You must score at least 18/45 (40%) on the final exam to pass the course
  - You must score at least 25% on each of the two components of the exam to pass the course
- Rationale:
  - Demonstrates that you've achieved the purpose of the course
  - The 40% hurdle alone, enables students to pass the course with only a strong understanding of theory *or* strong practical skills
  - *Both* these aspects are required in order to complete COMP2521 as well as to have an adequate foundation for the courses for which COMP2521 is a prerequisite
  - What I understand the content, and usually program well, but simply can't get my code to work during the exam?

Special Consideration

Introduction
Outline
People
Teaching
Assessment
Resources
Expectations
Advice
Advice
Feedback
Acknowledgements
Tools

- 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

Introduction
Outline
People
Teaching
Assessment
Resources
Expectations
Advice
Advice
Feedback
Acknowledgements
Tools

Summary:

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

Assessment

Introduction
Outline
People
Teaching
Assessment
Resources
Expectations
Advice
Advice
Feedback
Acknowledgements
Tools

To pass you must:

- score at least 50/100 overall
- score at least 18/45 on the final exam
- score at least 25% on each of the two components (theory and practical) of the final exam

Academic Integrity

Introduction
Outline
People
Teaching
Assessment
Resources
Expectations
Advice
Advice
Feedback
Acknowledgements
Tools

- 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

Introduction
Outline
People
Teaching
Assessment
Resources
Expectations
Advice
Advice
Feedback
Acknowledgements
Tools

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

Introduction
Outline
People
Teaching
Assessment
Resources
Expectations
Advice
Advice
Feedback
Acknowledgements
Tools

Resources

Additional Help

- Ed forum
- Weekly consultations
  - Starting from week 2
  - Day and time to be announced on Thursday
  - For clarification of course content
- Help sessions
  - Starting from week 2 or week 3 (details to come on Thursday)
  - Schedule on course website
  - For help with labs and assignments

Introduction
Outline
People
Teaching
Assessment
Resources
Expectations
Advice
Advice
Feedback
Acknowledgements
Tools

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

Introduction
Outline
People
Teaching
Assessment
Resources
Expectations
Advice
Advice
Feedback
Acknowledgements
Tools

Expectations

What you can expect from us

- Regular announcements/updates
- Lecture slides released before lectures... some of the time...
  - Some lectures will closely follow lecture slides, others will not. Some lectures will have more detailed slides, others will not.
- 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

Introduction
Outline
People
Teaching
Assessment
Resources
Expectations
Advice
Advice
Feedback
Acknowledgements
Tools

- 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

Introduction
Outline
People
Teaching
Assessment
Resources
Expectations
Advice
Advice
Feedback
Acknowledgements
Tools

Programming is like swimming.

- It doesn't matter how many hours you spend watching someone swim and watching someone teach swimming, at some point you need to get into the water.
- Effort over time:
  - Approach 1: 2 hours a day, 5 days a week, for 10 weeks
  - Approach 2: 10 hours a day for the 10 days leading up to the exam
  - These two are not equal!!

Advice

Introduction
Outline
People
Teaching
Assessment
Resources
Expectations
Advice
Advice
Feedback
Acknowledgements
Tools

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

You can improve if you put in the effort!

Feedback

Introduction
Outline
People
Teaching
Assessment
Resources
Expectations
Advice
Advice
Feedback
Acknowledgements
Tools

We'd love to get your feedback throughout the term!
https://forms.office.com/r/riGKCze1cQ

Feedback is also collected via myExperience at the end of the term.

Acknowledgements

Introduction
Outline
People
Teaching
Assessment
Resources
Expectations
Advice
Advice
Feedback
Acknowledgements
Tools

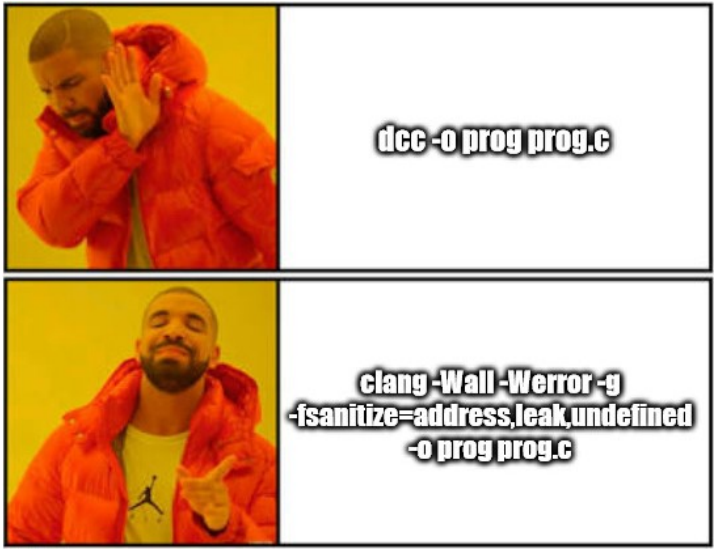
COMP2521 material drawn from...

- slides and memes by Kevin Luxa (COMP2521 24T1)
- 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

The Tools of the Trade

Compilation

Introduction
Tools
Compilation
Sanitizers
valgrind
make



Introduction
Tools
Compilation
Sanitizers
valgrind
make

Compilation

clang

COMP2521 uses the clang compiler. Basic compilation command:

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

Introduction
Tools
Compilation
Sanitizers
valgrind
make

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

Introduction
Tools
Compilation
Sanitizers
valgrind
make

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

Introduction
Tools
Compilation
Sanitizers
valgrind
make

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

Introduction
Tools
Compilation
Sanitizers
valgrind
make

{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!

Introduction
Tools
Compilation
Sanitizers
valgrind
make

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

Introduction
Tools
Compilation
Sanitizers
valgrind
make

```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

Introduction
Tools
Compilation
Sanitizers
valgrind
make

```
=================================================================
==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");
    }
}
```

Introduction
Tools
Compilation
Sanitizers
valgrind
make

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

Introduction
Tools
Compilation
Sanitizers
valgrind
make

- 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.

Introduction
Tools
Compilation
Sanitizers
valgrind
make

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`

Feedback

Introduction
Tools
Compilation
Sanitizers
valgrind
make

https://forms.office.com/r/riGKCze1cQ