Week 13: Revision

Couse review, Exam, future

- Postconditions of the course
- Preparing for the Exam
- Examples
- Examination format

Key postconditions

- Ability to read and write correct, clean code in C that allocates, deallocates and manages memory
- Ability to correctly implement standard linked list data structures
- Evaluate common memory-related errors (such as memory leaks, dangling pointers) and how to avoid these
- NULL != ‘\0’
- No variable length arrays!

Key postconditions

- Ability to read and write code that correctly uses the main standard library functions, especially for I/O, file handling, and string handling.

- Ability to use code quality strategies appropriate for C, including preprocessor techniques, and use of common idioms

- Apply a thorough automated testing regime using tools such as make, diff, scripts to present the outcomes, and a tool to manage regression testing.

Key postconditions

- Understanding of the approach and concepts of Unix, including its tools philosophy, processes (including pipes and redirection), the file system, and the shell.

- Ability to learn to use Unix & C commands and system calls (including usage of flags etc) from online manual system.

Key postconditions

- Understand the limitations of sequential computing performance and concepts in parallel programming
  - task-parallelism, data-parallelism
  - Synchronisation
  - Deadlock, livelock, starvation
  - Locking hierarchy
  - Scheduling
  - …
- Ability to design a parallel solution to a given problem. Use a parallel thread library such as Pthreads to derive the code.

- Ability to measure performance through profiling and identify load balancing issues or bottlenecks in either software and hardware.

Essentials for the pass

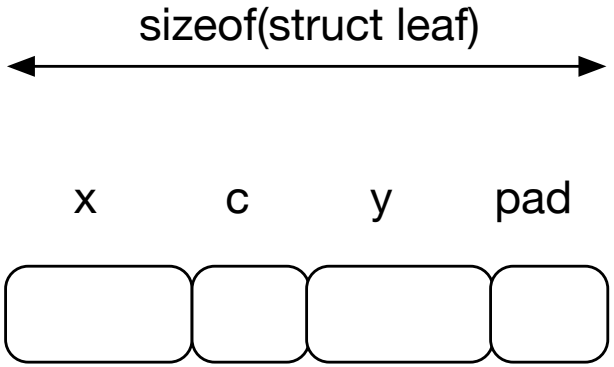
- C aspects that are conceptually like Java
- C pointers basics
- C pointers with arrays and strings
- C pointers with lists and other data structures
- C memory model
- Applying synchronization primitives

Preparing for core

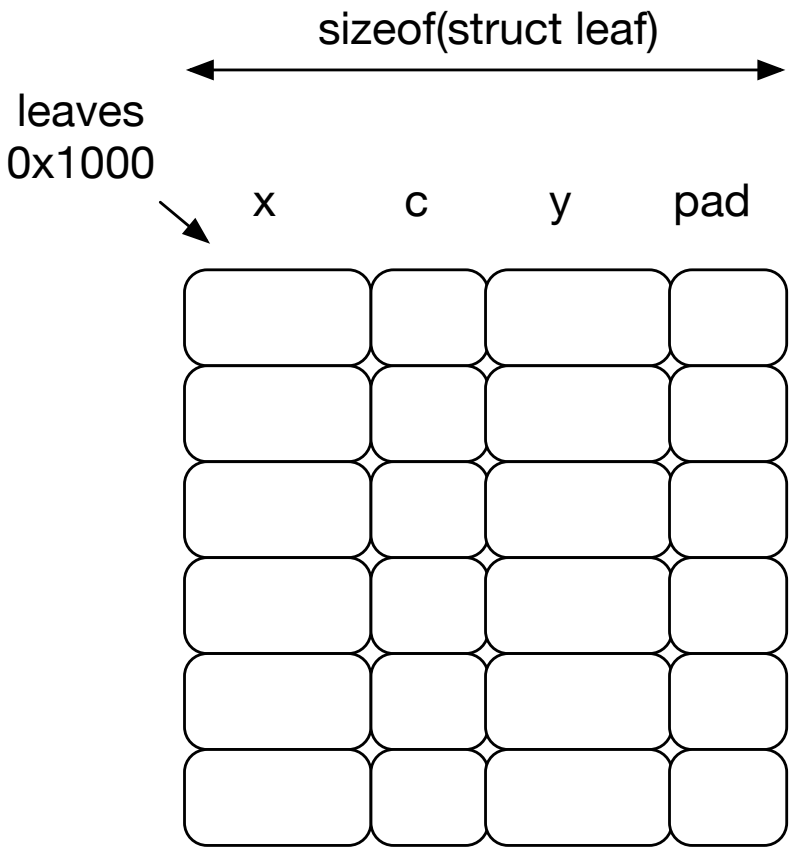
- Review the lab material, lectures
- Especially review C aspects
- Practice all the exercises from labs and lectures
- Write small code examples
  - Get them to work
  - Test them (write small test scripts)
  - So you know they are right
  - Draw pictures of memory

```
struct leaf {
    int x;
    char c[2];
    float y;
};

struct leaf leaves[6];
```

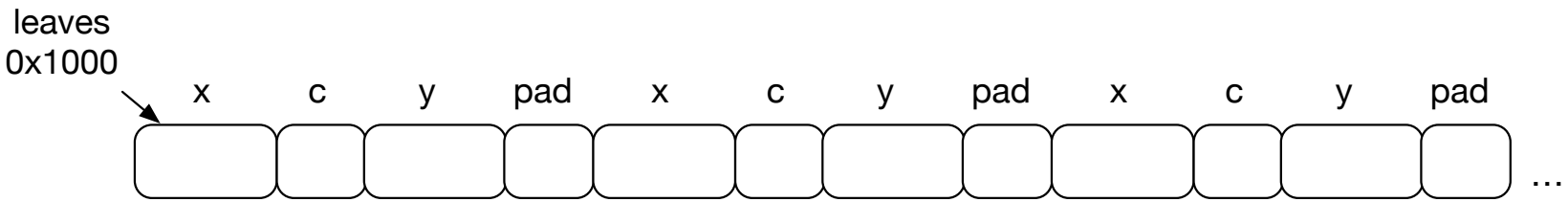
```
struct leaf {
    int x;
    char c[2];
    float y;
};

struct leaf leaves[6];
```



```
struct leaf {
    int x;
    char c[2];
    float y;
};

struct leaf leaves[6];
```

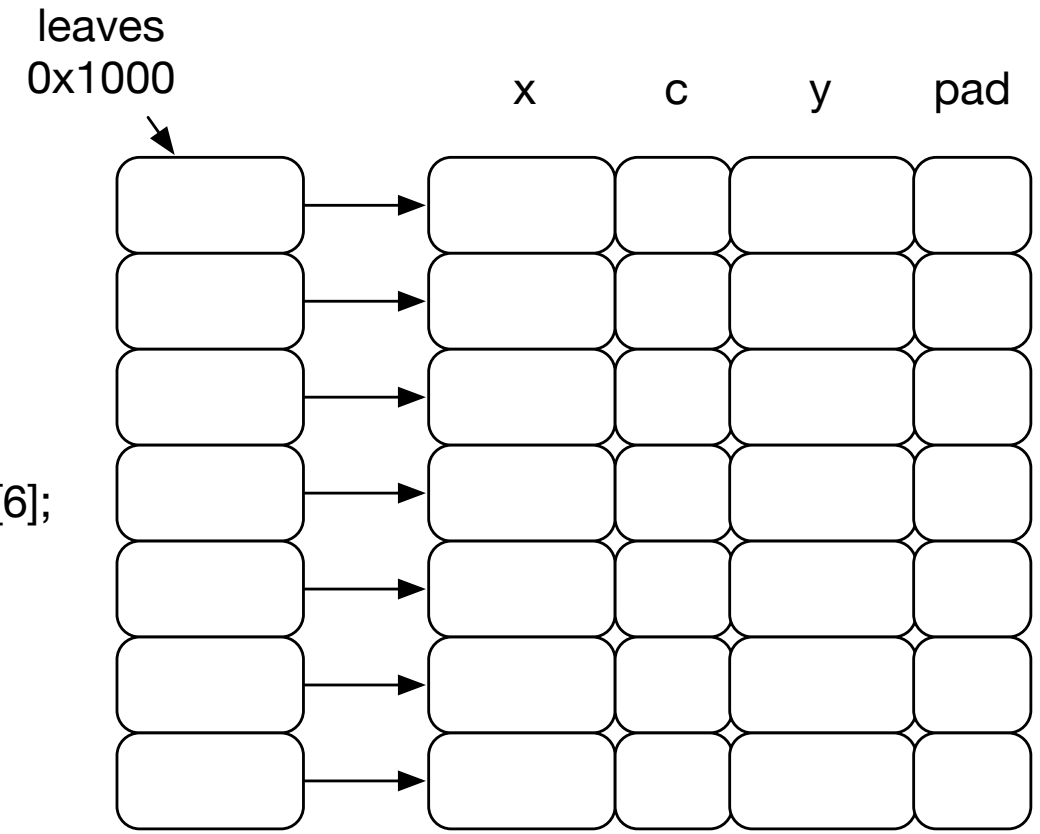


```
struct leaf {
    int x;
    char c[2];
    float y;
};

struct leaf leaves[6];
```



What has changed?

```
struct leaf {
    int x;
    char c[2];
    float y;
};

struct leaf *leaves[6];
```



Preparing for core

- Aim to write clear code
- Write comments to make your thinking clear
  - Eg. Initialise Arr to values 1 .. N.
- Draw pictures
  - They really do help
  - Good programmers draw them – So why not you?
- Map out solution at high level
  - Write pseudo-code if it is easier first
  - Write comments for each main step
  - Write actual code or function call
  - Later, write within each of the function

Example

- Context: there is a linked list of ints:
  - Task: add 1 to the value of each element
- Steps:
  - Needs to traverse a list
  - Needs to have a list that has been built
- Examples you know you need to test
  - Empty list
  - Single element list
  - Problems at head
  - Problems at end
  - “Normal “case

Example

- Start with diagram/pictures, whatever helps you
- Write an outline for special cases
- Write code for special cases

- Think about top level steps
- Write a main() to reflect this
  - Comments
  - Function calls
  - Flesh out details

- Aim for clarity, simplicity, correctness
- Once high level is done, write each function

What is the goal of this code?

```
int index_match(char *str,char *pattern) {
        int tarindex = 0;
        while(str[tarindex] != '\0') {
                int i;
                int tarlen = tarindex;
                for (i = 0; pattern[i] != '\0'; i++) {
                        if (str[tarlen++] != pattern[i]) {
                                break;
                        }
                }
                if (pattern[i] == '\0') {
                        return tarindex;
                }
                tarindex++;
        }
        return -1;
}
```

https://github.com/wenruo95/algorithm/blob/master/c/kmp.c

Can you comment on the following code

```
59  FileList construct_path_list(int argc, char **argv, char const *filename) {
60      FileList list;
61      list.watch_files = 0;
62      list.exclude_files = 0;
63      FILE *file = 0;
64
65      if (!filename) {
66      } else if (!strcmp(filename, "-")) {
67          file = stdin;
68      } else {
69          file = fopen(filename, "r");
70      }
```

https://github.com/rvoicilas/inotify-tools/blob/master/src/common.c

Can you comment on the following code

```
60      FileList list;
61      list.watch_files = 0;
62      list.exclude_files = 0;
```

```
72      int watch_len = LIST_CHUNK;
73      int exclude_len = LIST_CHUNK;
74      int watch_count = 0;
75      int exclude_count = 0;
76      list.watch_files = (char const **)malloc(sizeof(char *) * watch_len);
77      list.exclude_files = (char const **)malloc(sizeof(char *) * exclude_len);
```

https://github.com/rvoicilas/inotify-tools/blob/master/src/common.c

Can you comment on the following code

```
79      char name[MAXLEN];
80      while (file && fgets(name, MAXLEN, file)) {
81          if (name[strlen(name) - 1] == '\n')
82              name[strlen(name) - 1] = 0;
83          if (strlen(name) == 0)
84              continue;
85          if ('@' == name[0] && strlen(name) == 1)
86              continue;
87          if ('@' == name[0]) {
88              resize_if_necessary(exclude_count, exclude_len, list.exclude_files);
89              list.exclude_files[exclude_count++] = strdup(&name[1]);
90          } else {
91              resize_if_necessary(watch_count, watch_len, list.watch_files);
92              list.watch_files[watch_count++] = strdup(name);
93          }
94      }
95      if (file && file != stdin)
96          fclose(file);
```

https://github.com/rvoicilas/inotify-tools/blob/master/src/common.c

Quick review of sample C code

```
 1  static OSStatus
 2  SSLVerifySignedServerKeyExchange(SSLContext *ctx,
 3              bool isRsa, SSLBuffer signedParams,
 4              uint8_t *signature, UInt16 signatureLen)
 5  {
 6          OSStatus err; ...
 7          if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
 8                  goto fail;
 9          if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
10                  goto fail;
11                  goto fail;
12          if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
13                  goto fail; ...
14  fail:
15          SSLFreeBuffer(&signedHashes);
16          SSLFreeBuffer(&hashCtx);
17          return err;
18  }
```

```
1  static int next_word(const char *str, int bpos, int *cdiff,
2                       const char *delim, int direction)
3  {
4          int skip_delim = 1;
5          while ((direction > 0) ? str[bpos] : (bpos > 0)) {
6                  uchar ch;
7                  int oldp = bpos;
8
9                  if (direction > 0) {
10                         ch = u_get_char(str, &bpos);
11                 } else {
12                         u_prev_char_pos(str, &bpos);
13                         oldp = bpos;
14                         ch = u_get_char(str, &oldp);
15                 }
16
17                 if (u_strchr(delim, ch)) {
18                         if (!skip_delim) {
19                                 bpos -= bpos - oldp;
20                                 break;
21                         }
22                 } else
23                         skip_delim = 0;
24
25                 *cdiff += direction;
26         }
27         return bpos;
28 }
```

https://github.com/cmus/cmus/blob/master/cmdline.c

Next?

2nd year	COMP2017 Systems Programming	SOFT2201 Software Construction and Design 1	INFO2150 Health System Data Standards & Analysis	ISYS2110 Analysis & Design of Web IS	
	COMP2022 Programming Languages, Logic, & Models	SOFT2412 Agile Software Development Practices	DATA2001 Data Science: Big Data and Data Diversity	ISYS2120 Data and Information Management	
	COMP2123 Data Structures & Algorithms	INFO2222 Computing 2 Usability and Security	DATA2002 Data Analytics: Learning from Data	ISYS2160 IS in the Internet Era	…
3rd Year	COMP3530 Discrete Optimization	COMP3419 Graphics and Multimedia	INFO3333 Computing 3 Management	ISYS3402 Decision Analytics & Support Systems	
	COMP3027 Algorithm Design	COMP3520 Operating Systems Internals	INFO3315 Human-Computer Interaction	COMP3615, ISYS3400, SOFT3413 Project	
	COMP3221 Distributed Systems	SOFT3410 Concurrency for Software Development	INFO3616 Principles of Security and Security Eng	DATA3404 Data Science Platforms	
	COMP3308 Introduction to Artificial Intelligence	INFO3220 Object Oriented Design	ISYS3401 Information Technology Evaluation	INFO3406 Introduction to Data Analytics	…

Examination format

- Two main parts.
- PART 1: Open book examination
- PART 2: Closed book oral examination

PART 1: Open Book Computer Examination

Open book meaning

ALLOWED

- You may refer to course materials: textbook, lecture, tutorials, task sets and assignments

- man pages (access via terminal)

- C specification (this C11 draft is more than you need: http://www.open-std.org/jtc1/sc22/WG14/www/docs/n1570.pdf)

PART 1: Open Book Computer Examination

NOT ALLOWED

- Asking another person to give you hints, dictate, communicate, or do the work for you.
- Using internet search for reference materials is forbidden (you have the allowed above). For example, searching "function for white space in C", "solution to palindrome in C", or ”how to use strcmp()”, “what is a pointer” etc.
- Using any other mechanism that defies the objective measure of this assessment.
- Academic integrity applies to all assessments including the final examination.

PART 1: Open Book Computer Examination

4 major parts

- Questions relating to C
- There are questions to require use of standard library
- There are questions that forbid anything external (show us you know how to build them!)
- Syntax is important and with the reference pages, there is little excuse for not knowing which symbol, or function parameter order is correct.
- **The program must compile and run on the environment specified**

PART 1: Open Book Computer Examination

Warnings

- Avoid using your own software development configuration against that which is sought, or provided. The fact that "it works on my machine" does not lend to the nature of a computer examination.

- If you make wild assumptions about concepts/behaviour, you will lose marks.
  e.g. I assume fgets always reads the entire file into my buffer → very wrong!. e.g. there is always enough memory → very wrong!.

- There are also no magic functions that you can assume exist to solve the problem. The scope of which functions to use is restrictive as illustrated in the question. e.g.

```
#include “solution.h”
int main() {
        return solution_run_for_full_marks();
}
```

- Don’t expect help. This as an *examination*.

PART 2: Close Book Oral Examination

- Closed book. No external materials allowed. No open notes, documents, pages, etc.
- One on one examination. Individual assessment
- sIdentification required
- Video, Voice, and Screen sharing abilities
- Time available is at most 30 minutes
- Please be mindful that when the time expires, the session will end.

PART 2: Close Book Oral Examination

- Examiner will ask different sets of questions to test your understanding of the general subject matter and further questions about your understanding of programming ability based on experience you should have accrued during the course.

- Not all questions will need to be answered correctly. You should expect the majority are needed for the examiner to confirm you.

- Code, diagrams, or text may be shared with you during the examination to aid your understanding of the questions context.

- The time limit allows for you to think and give an answer. Remember that this is an opportunity for you to explain your answer and be able to communicate your ideas to the examiner. Mistakes in word choices and statements made can be corrected.

Final Grade

To pass this course, you need to:

- Obtain at least 40% in all other assessments that are not the final examination, AND
- Obtain at least 50% in the final examination to qualify for a pass in this course, AND
  - Pass the oral examination,
  - Pass the open book examination,
- Obtain at least 50% final mark overall

The grade for the final examination will reflect the combination of the open book exam and the closed book oral examination.

Your heroes of the course

- Teaching Assistant
  - **Tyson Thomas**
  - **William Wang**
  - **Byung Hoon Cho**

- Tutors
  - Anuj Dhavalikar
  - Gregory McLellan
  - Alan Robertson
  - Yun Li
  - Shumin Kong
  - Mathew Magee
  - David Byrne
  - Sam Arch
  - Michael Paul Zammit
  - Matthew Strasiotto
  - Zijun Hui
  - Darius Zhu
  - Ngoc Mai Le
  - Dennis Chen

This course is needs feedback

- Please take a few moments to complete the end of semester survey
- http://sydney.edu.au/itl/surveys/complete
- Give us the good and the bad on lectures, tutorials, assignments, and anything else on your mind
- It is anonymous, so please name your tutors!

Thank you

Good luck