

Week 13: Revision

COMMONWEALTH OF AUSTRALIA

Copyright Regulations 1969

WARNING

This material has been reproduced and communicated to you by or on behalf of the University of Sydney pursuant to Part VB of the Copyright Act 1968 (**the Act**).

The material in this communication may be subject to copyright under the Act. Any further copying or communication of this material by you may be the subject of copyright protection under the Act.

Do not remove this notice.

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.

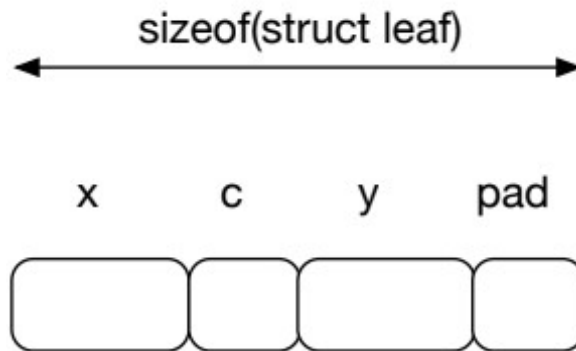
```
struct leaf {  
    int x;  
    char c[2];  
    float y;  
};
```

```
struct leaf leaves[6];
```



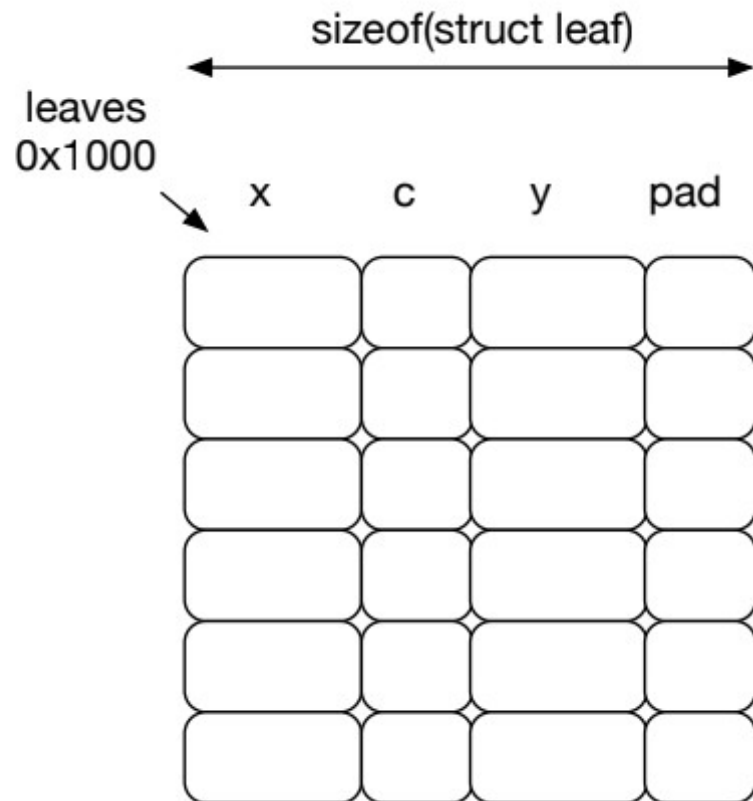
```
struct leaf {  
    int x;  
    char c[2];  
    float y;  
};
```

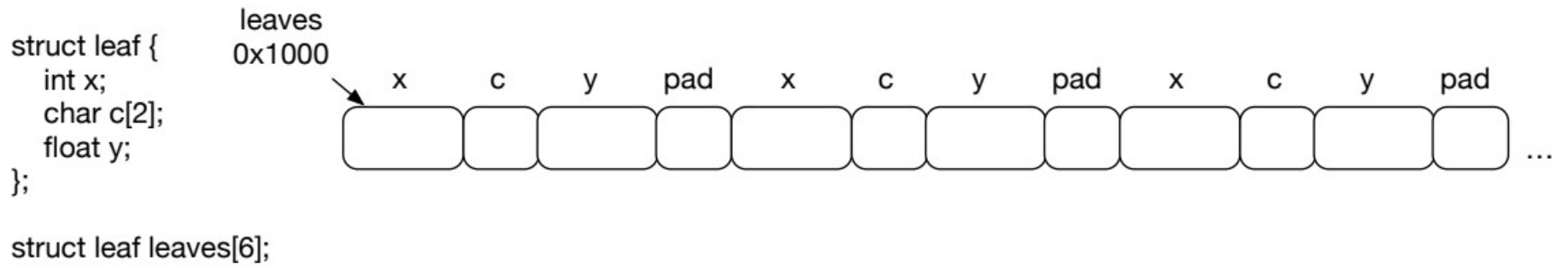
```
struct leaf leaves[6];
```



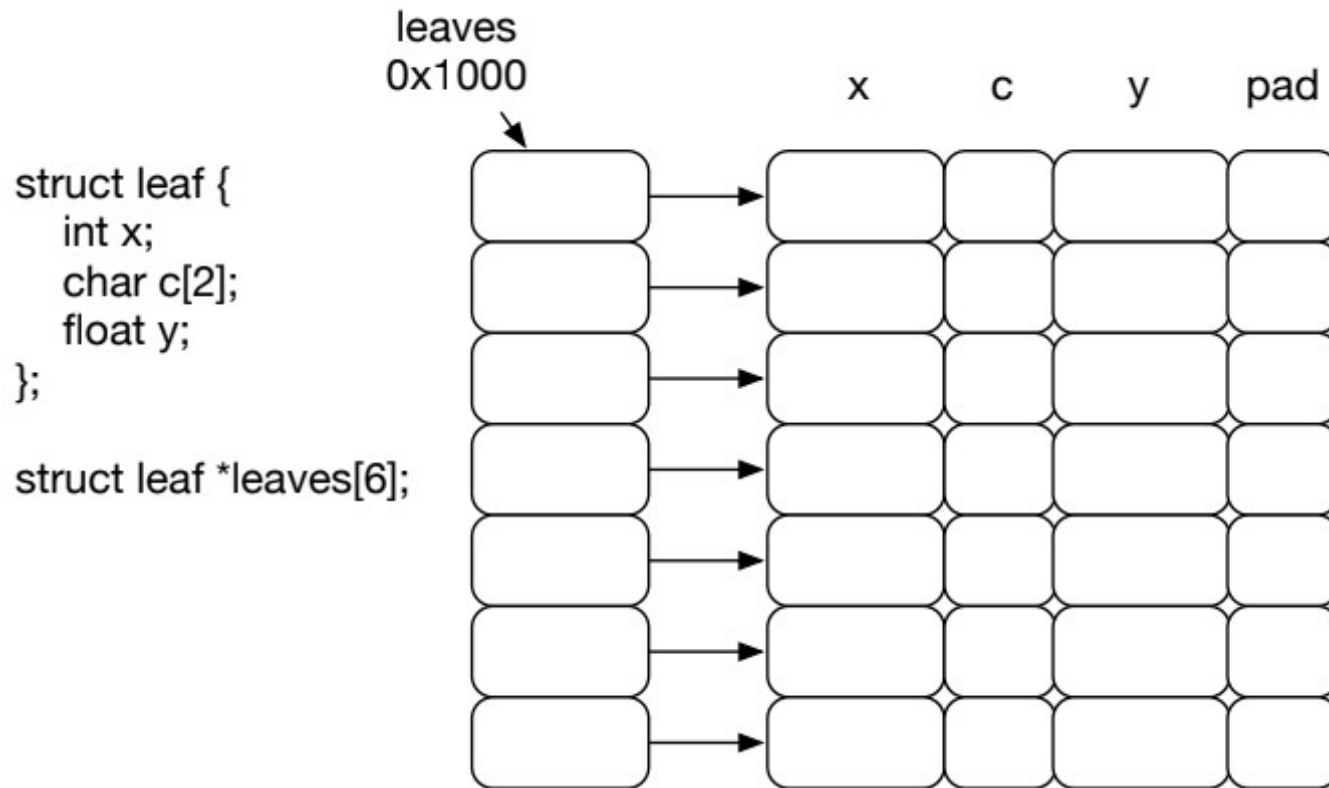
```
struct leaf {  
    int x;  
    char c[2];  
    float y;  
};
```

```
struct leaf leaves[6];
```





What has changed?



Example

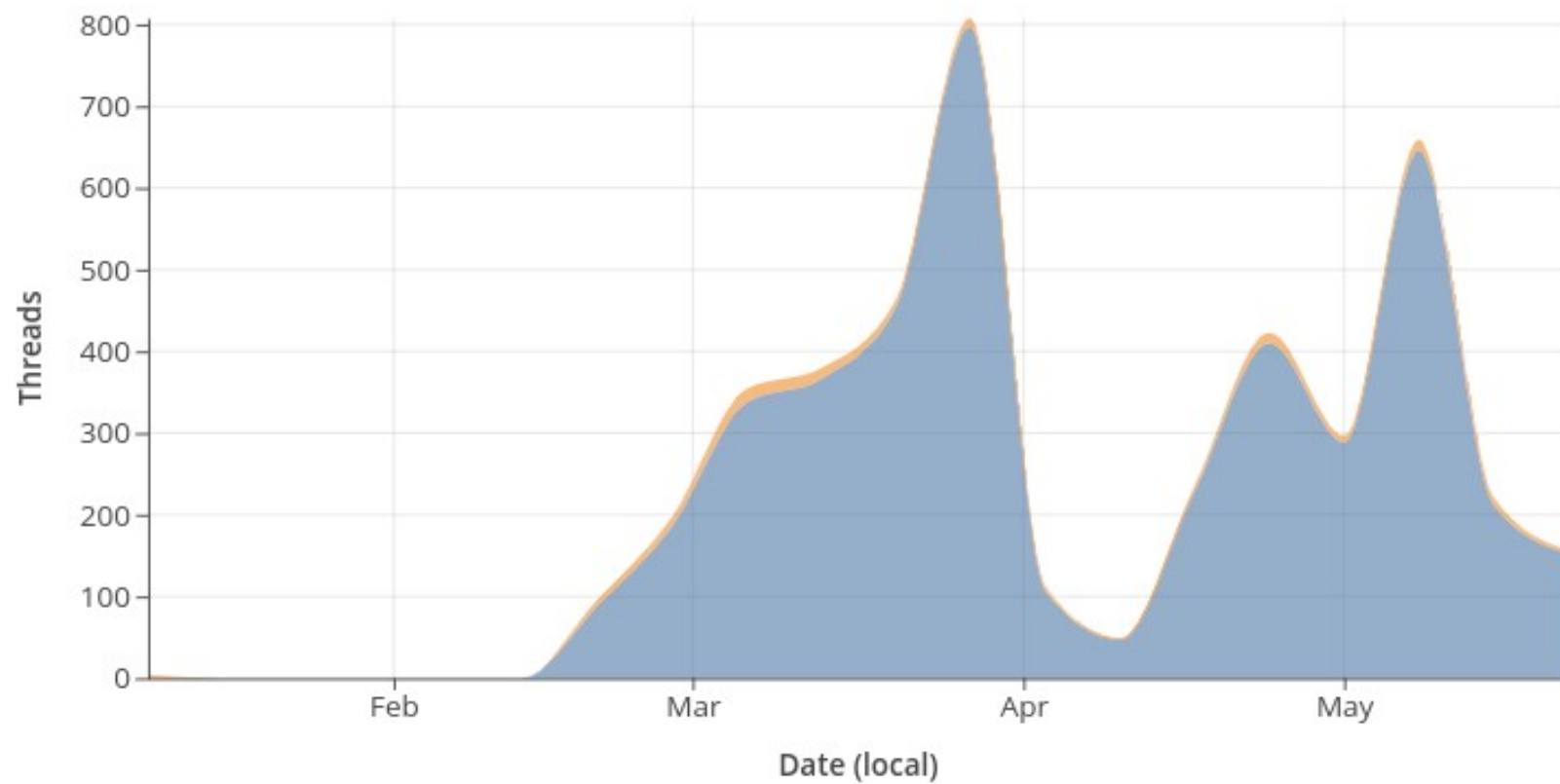
- **Context:** there is a linked list of ints:
 - **Task:** add 1 to the value of each element
- **Steps:**
 - Needs to traverse a list
 - Needs to have a list that has been built
- **Examples you know you need to test**
 - Empty list
 - Single element list
 - Problems at head
 - Problems at end
 - “Normal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

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;  
}
```

Quick review of sample C code

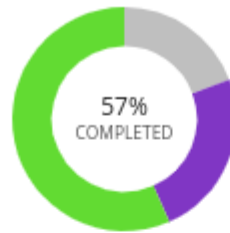
```
1 static OSStatus
2 SSLVerifySignedServerKeyExchange(SSLContext *ctx,
3     bool isRsa, SSLBuffer signedParams,
4     uint8_t *signature, UInt16 signatureLen)
5 {
6     OSStatus err; ...
7     if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
8         goto fail;
9     if ((err = SSLHashSHA1.update(&hashCtx, &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 }
```



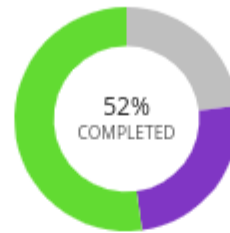
★ Challenges — Week 1 Challenges



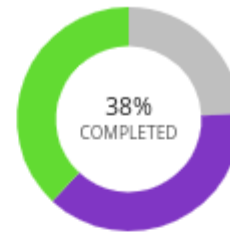
1 <> Hello from C



2 <> Greetings



3 <> Min, Max and Sum



4 <> Rot 13

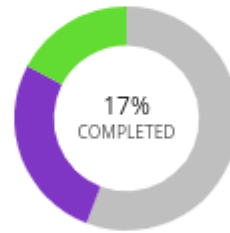
★ Challenges — Week 2 Challenges



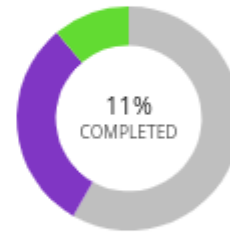
1 <> Count Files



2 <> Bouncy String



3 <> Reverse Array



4 <> Triforce

★ Challenges — Week 3 Challenges



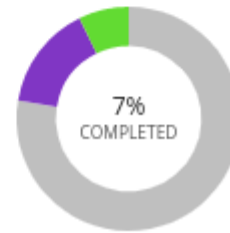
1 <> HamsterDecode



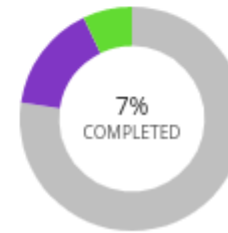
2 <> Binary



3 <> Caesar Cipher



4 <> Reverse



5 <> Only chars

Credits

- Many people contributed to the preparation and delivery of the COMP2129/COMP2017 course.
- Lecture preparation
 - Bernhard Scholz
 - Tony Greening
 - Judy Kay
 - Bob Kummerfeld
 - John Stavrakakis (course coordinator)

★ Your heroes of the course ★

- Teaching Assistants:
 - Tiancheng Mai, Haoyan Qi

Tutors:

Tyson Thomas	Xiaoxiang Wu	Keaun Wild
William Saffery	Pratham Purohit	Doris Tran
Simon Dowd	Richard McKenzie	William Gan
Lindsay Bath	Oliver Freeman	Joshua Blair
Chuang Chen	Dan Cocking	Yao Ke
Angus O'Grady	Matthew Kenny	

- Support: Hamish Croser
Byung Hoon Cho

This course needs feedback

- Please take a few moments to complete the end of semester survey
- <https://student-surveys.sydney.edu.au/students/>
- 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

