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.

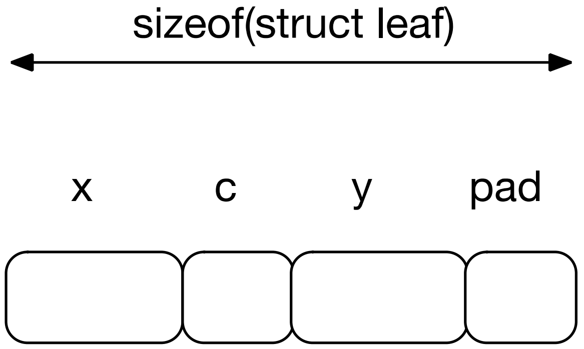
Essentials for the pass

- 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

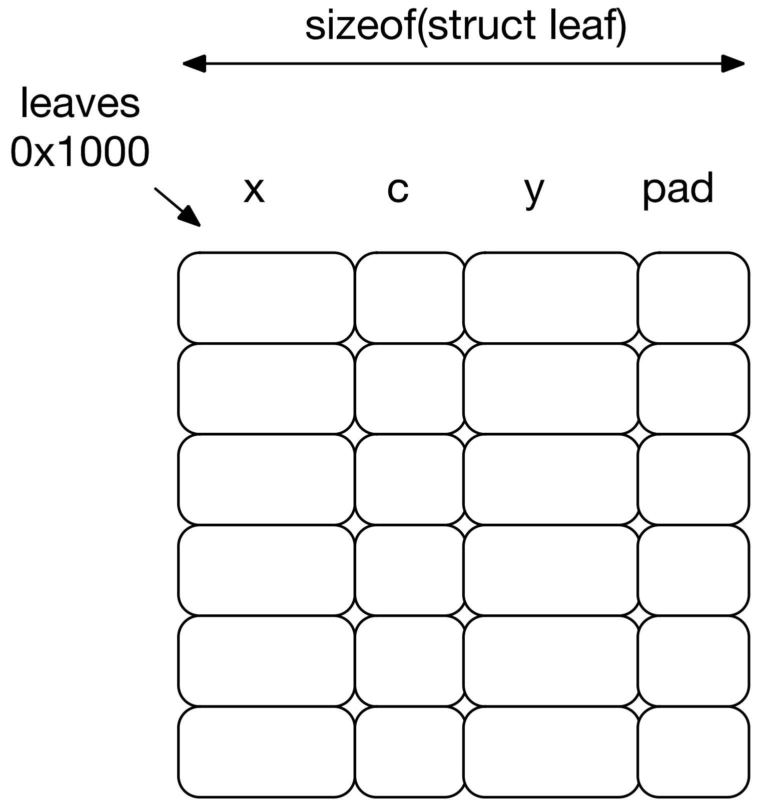
Preparing for core

- 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

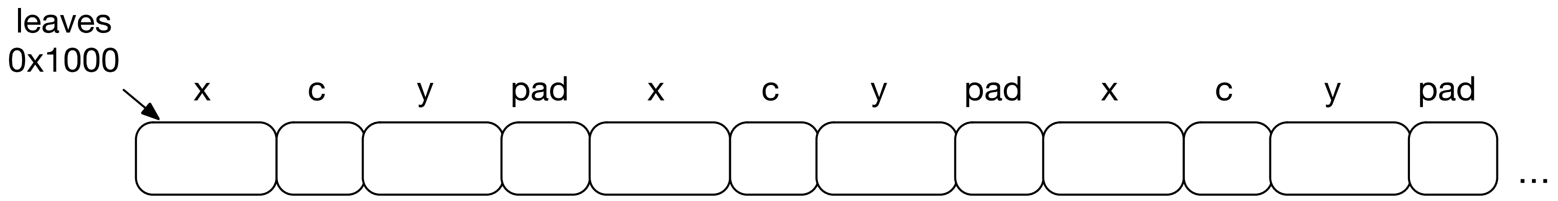
```
struct leaf {
    int x;
    char c[2];
    float y;
};

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];
```

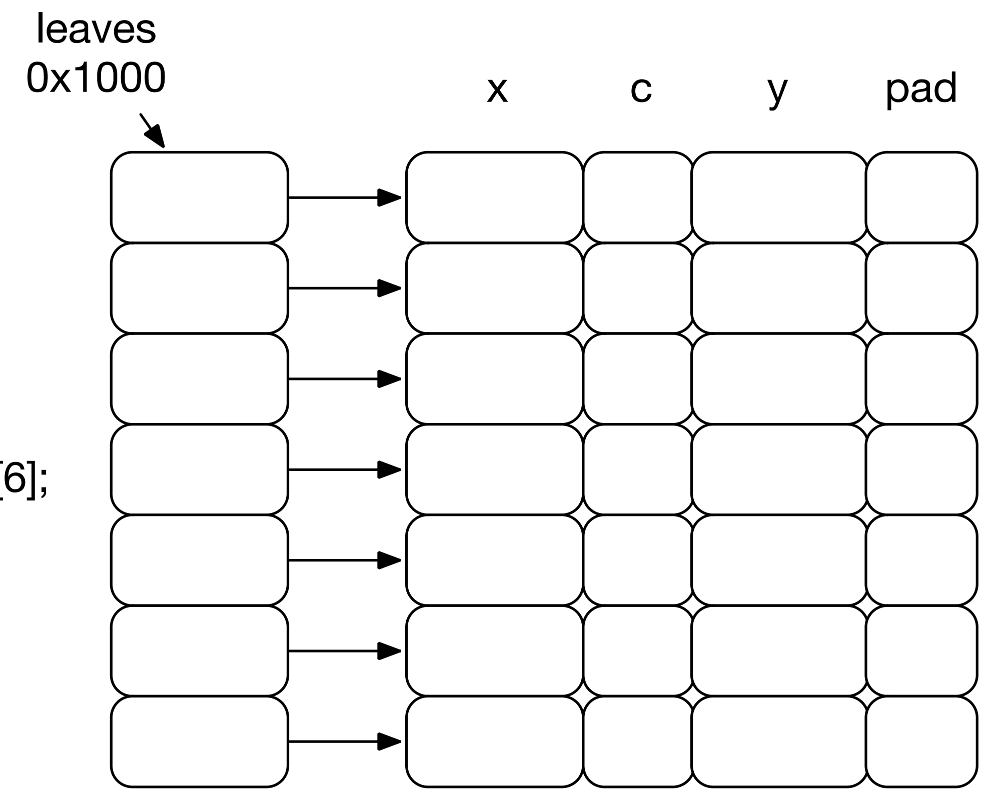


```
struct leaf {
    int x;
    char c[2];
    float y;
};

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

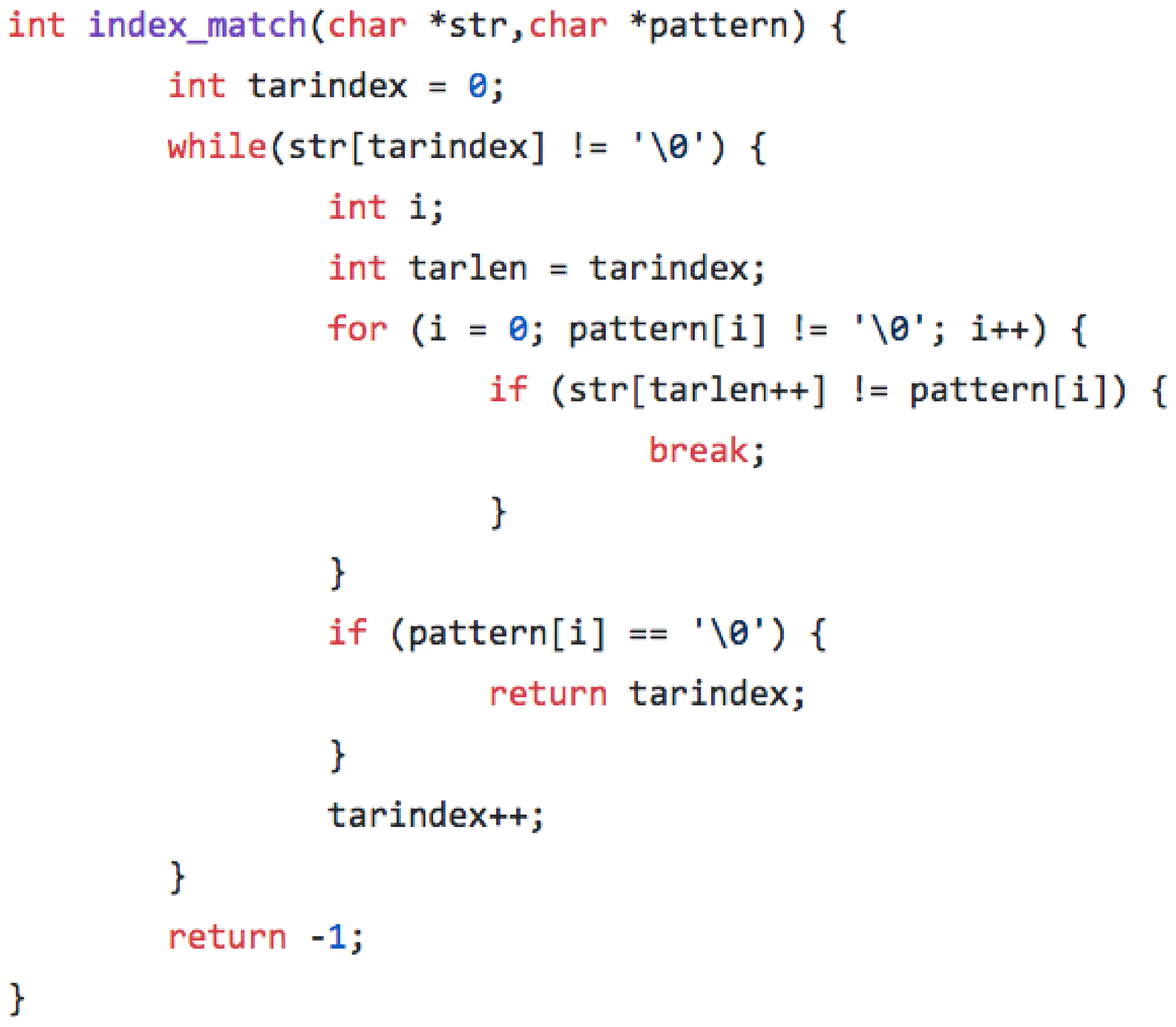
Example

- 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

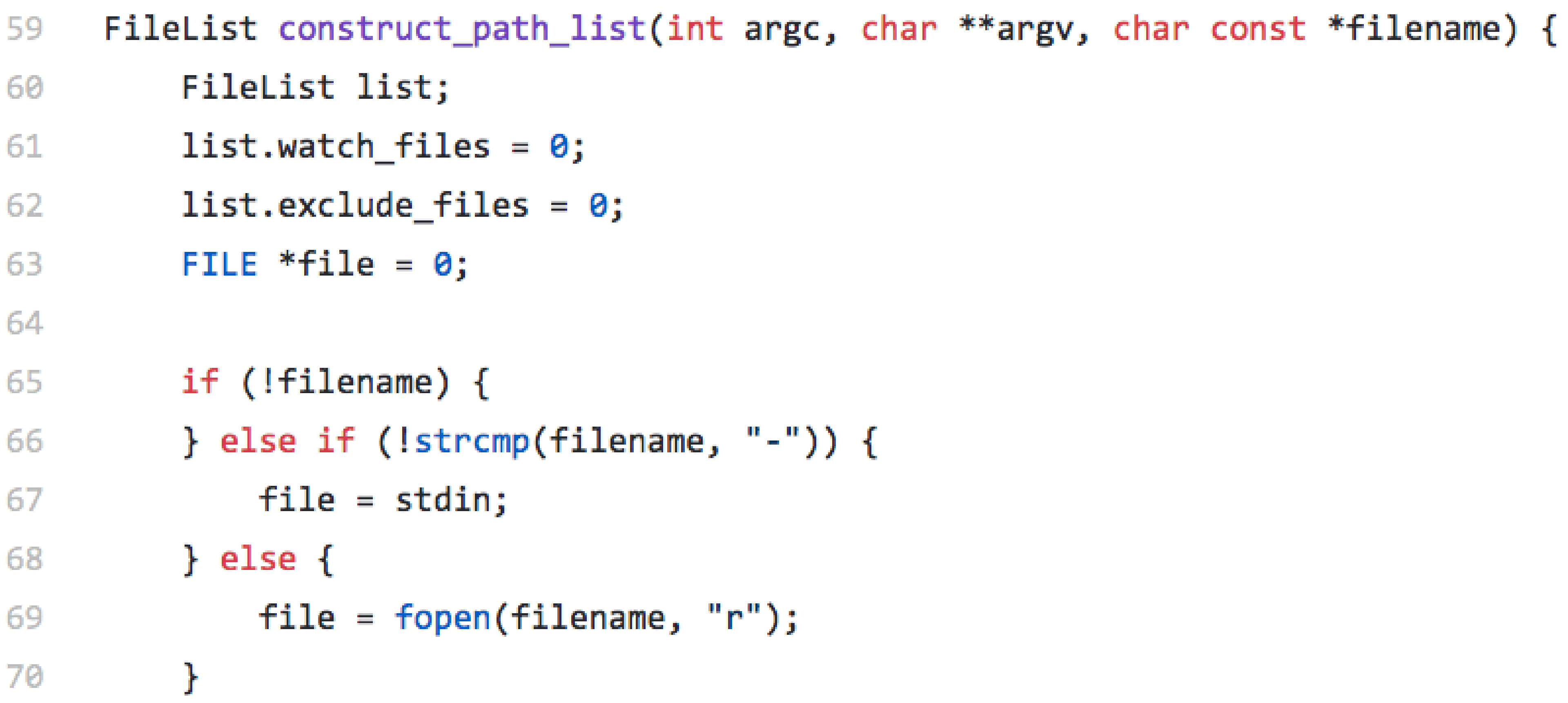
What is the goal of this code?



```c
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

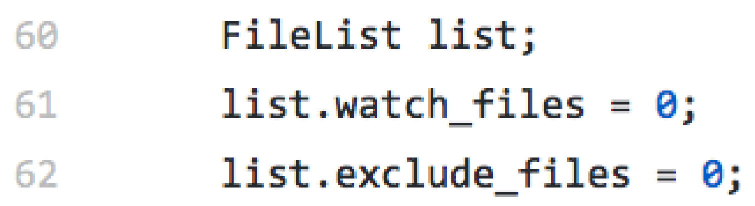
Can you comment on the following code



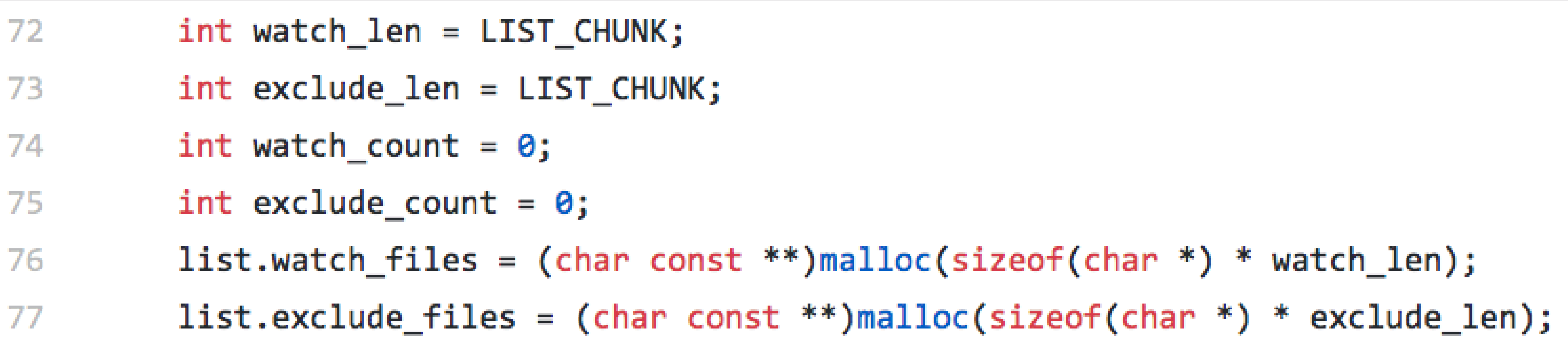
```c
FileList construct_path_list(int argc, char **argv, char const *filename) {
    FileList list;
    list.watch_files = 0;
    list.exclude_files = 0;
    FILE *file = 0;

    if (!filename) {
    } else if (!strcmp(filename, "-")) {
        file = stdin;
    } else {
        file = fopen(filename, "r");
    }
```

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

Can you comment on the following code



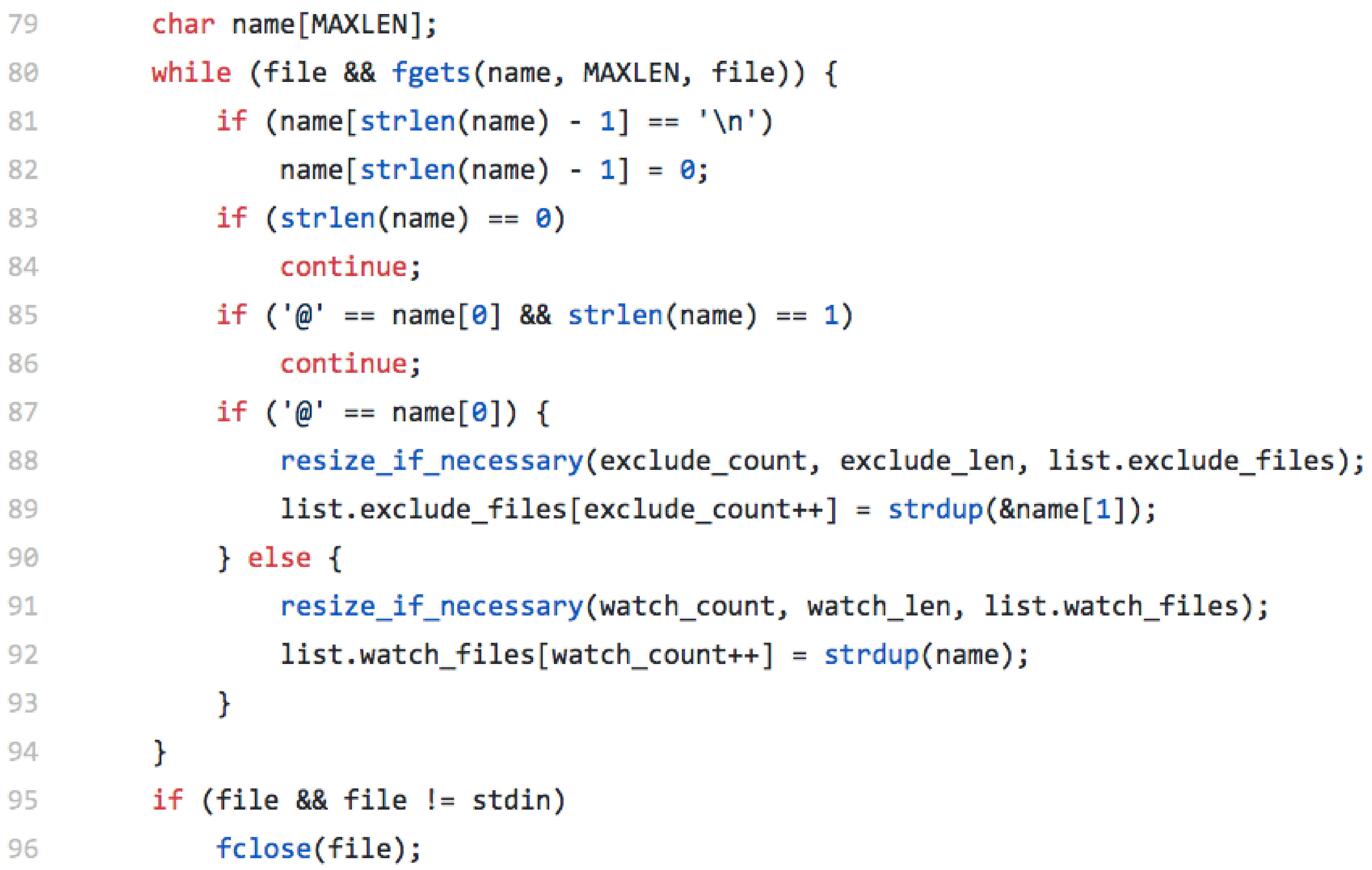
```c
    FileList list;
    list.watch_files = 0;
    list.exclude_files = 0;
```



```c
    int watch_len = LIST_CHUNK;
    int exclude_len = LIST_CHUNK;
    int watch_count = 0;
    int exclude_count = 0;
    list.watch_files = (char const **)malloc(sizeof(char *) * watch_len);
    list.exclude_files = (char const **)malloc(sizeof(char *) * exclude_len);
```

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

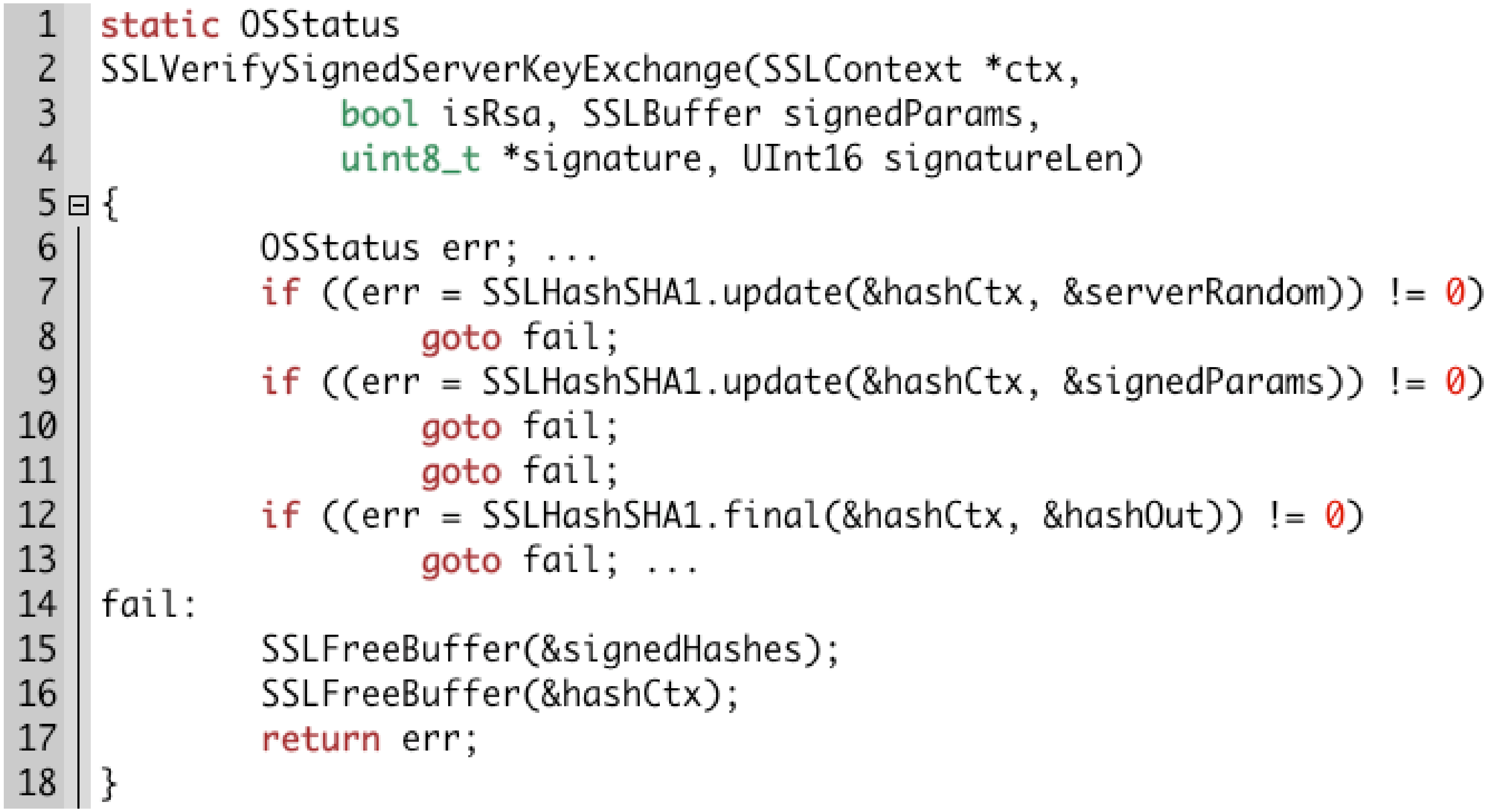
Can you comment on the following code



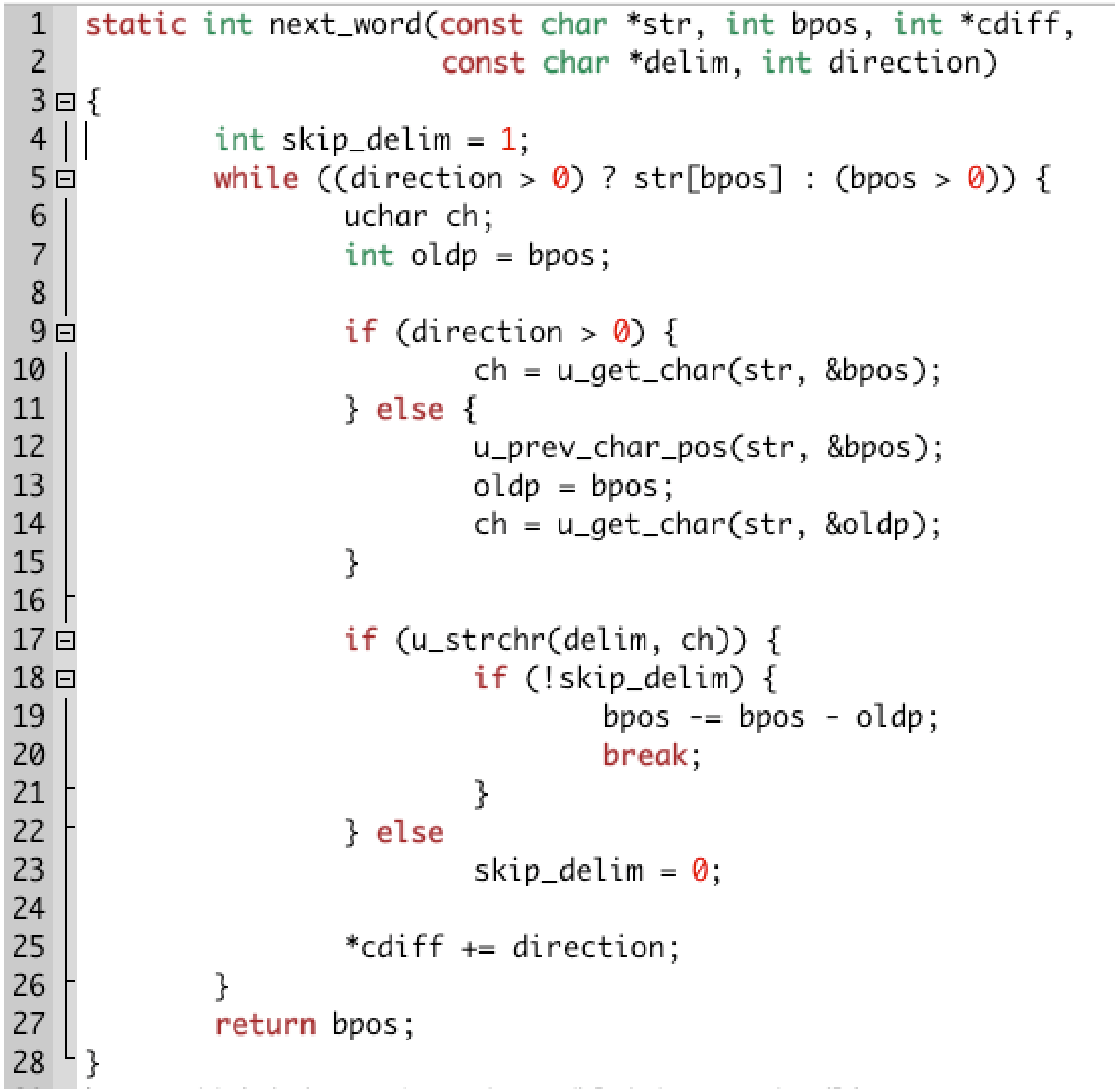
```c
char name[MAXLEN];
while (file && fgets(name, MAXLEN, file)) {
    if (name[strlen(name) - 1] == '\n')
        name[strlen(name) - 1] = 0;
    if (strlen(name) == 0)
        continue;
    if ('@' == name[0] && strlen(name) == 1)
        continue;
    if ('@' == name[0]) {
        resize_if_necessary(exclude_count, exclude_len, list.exclude_files);
        list.exclude_files[exclude_count++] = strdup(&name[1]);
    } else {
        resize_if_necessary(watch_count, watch_len, list.watch_files);
        list.watch_files[watch_count++] = strdup(name);
    }
}
if (file && file != stdin)
    fclose(file);
```

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

Quick review of sample C code



```c
static OSStatus
SSLVerifySignedServerKeyExchange(SSLContext *ctx,
            bool isRsa, SSLBuffer signedParams,
            uint8_t *signature, UInt16 signatureLen)
{
        OSStatus err; ...
        if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
                goto fail;
        if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
                goto fail;
                goto fail;
        if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
                goto fail; ...
fail:
        SSLFreeBuffer(&signedHashes);
        SSLFreeBuffer(&hashCtx);
        return err;
}
```



```c
static int next_word(const char *str, int bpos, int *cdiff,
                     const char *delim, int direction)
{
        int skip_delim = 1;
        while ((direction > 0) ? str[bpos] : (bpos > 0)) {
                uchar ch;
                int oldp = bpos;

                if (direction > 0) {
                        ch = u_get_char(str, &bpos);
                } else {
                        u_prev_char_pos(str, &bpos);
                        oldp = bpos;
                        ch = u_get_char(str, &oldp);
                }

                if (u_strchr(delim, ch)) {
                        if (!skip_delim) {
                                bpos -= bpos - oldp;
                                break;
                        }
                } else
                        skip_delim = 0;

                *cdiff += direction;
        }
        return bpos;
}
```

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

Examination format

- Closed book examination
- Canvas submissions only
- Question 1 – 40%
- Question 2 – 60 %

Examination

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)

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.

Examination & Code

- There are questions to require use of standard library
- 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**

Examination & Code

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*.

Submission of examination

- Canvas only
- Zip and upload completed source files
- Ed submission not accepted
- Code must compile
- No late submissions

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
- Obtain at least 50% final mark overall

Your heroes of the course

- Teaching Assistants: Tyson Thomas William Wang
- Tutors:

Anuj Dhavalikar
Gregory McLellan
Dennis Chen
Byung Hoon Cho
Andrew Xu

Leon Chen
Hanin Zenah
Ziyun (Erik) Chi
Xiaowen Hu
Alistair de Vroet

- Support:

Yun Li
Alan Robertson
Mathew Magee
Matthew Strasiotto
Zijun Hui

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