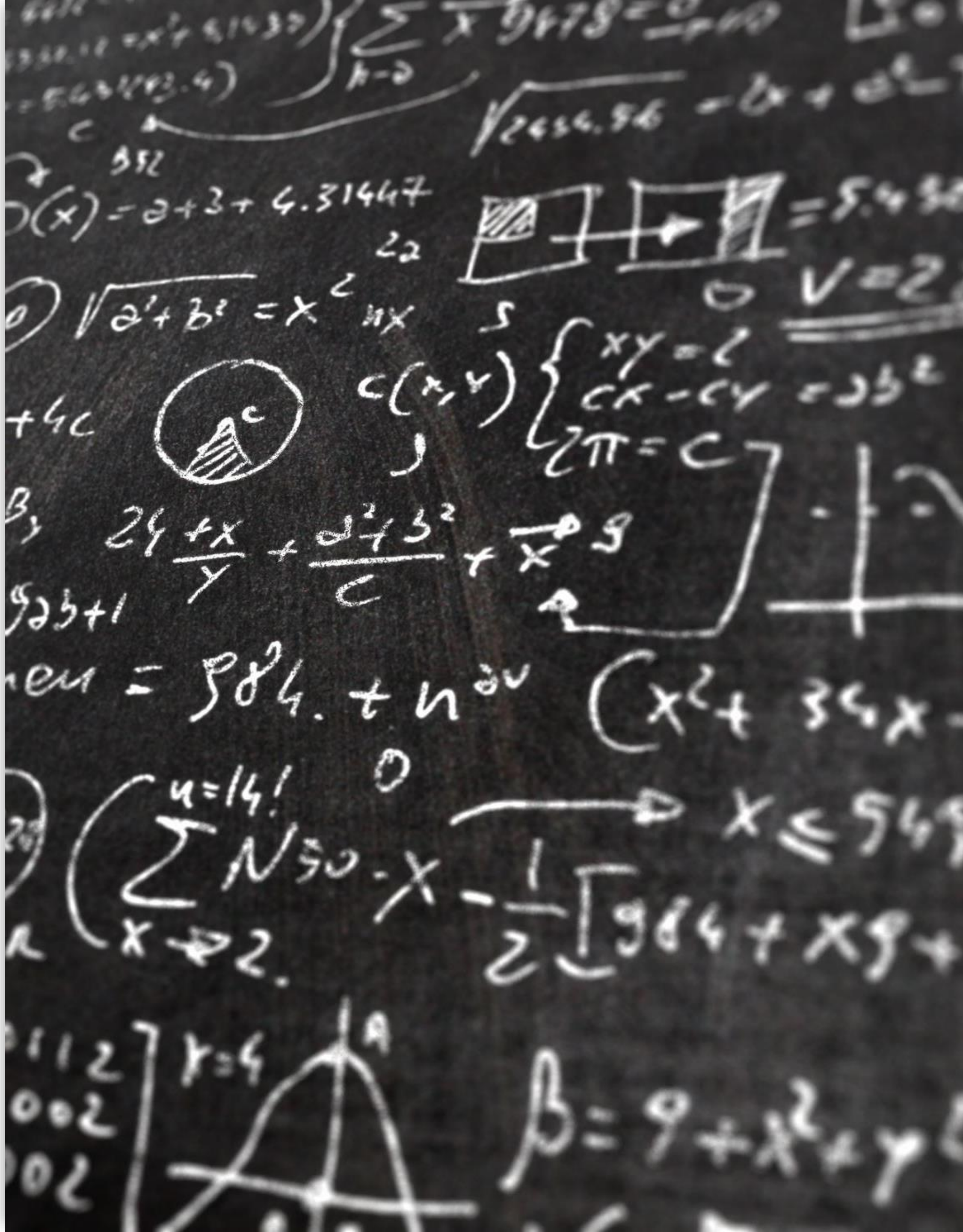
SCC.111 Software Development – Lecture 15: Files and I/O

Adrian Friday, Nigel Davies, Hansi Hettiarachchi, Saad Ezzini

This lecture

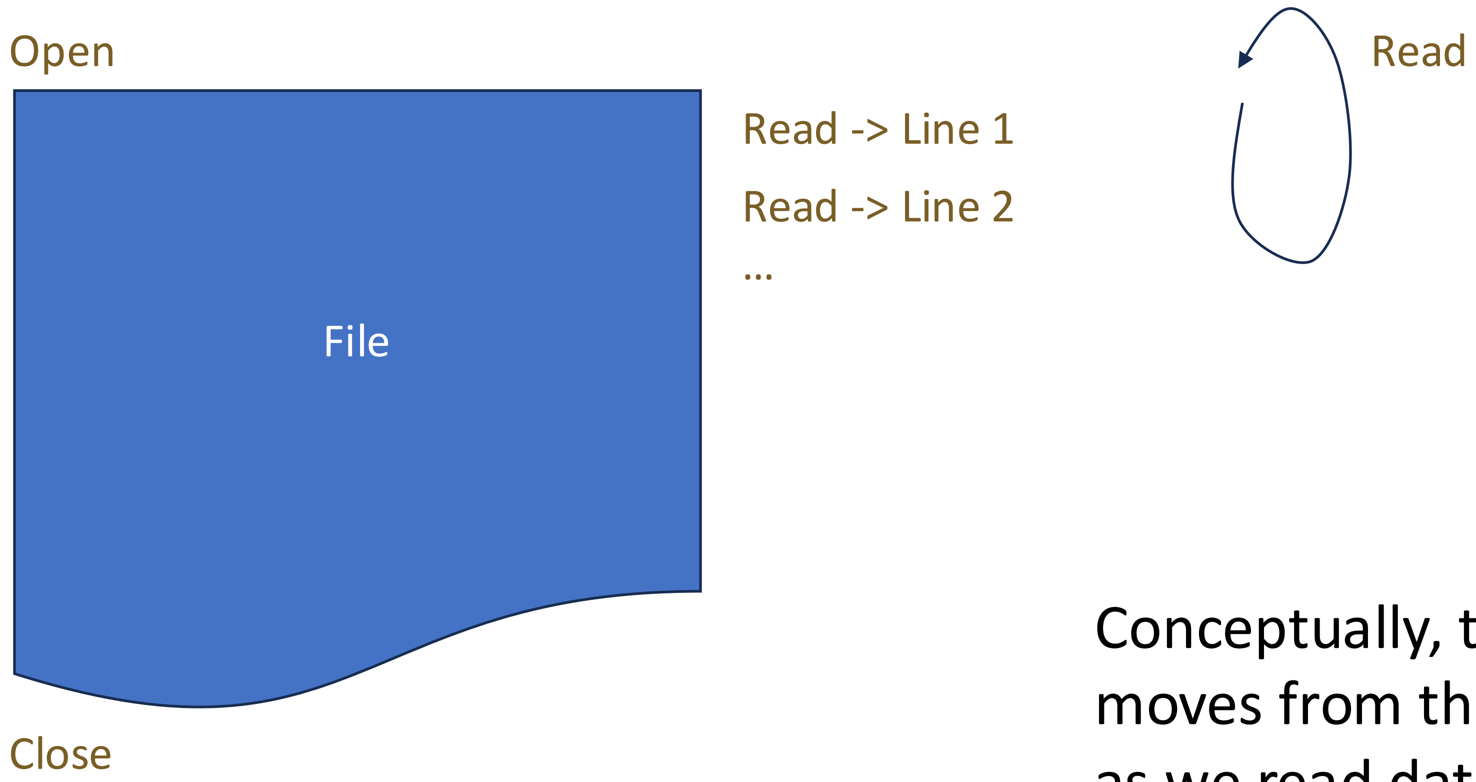
- Using file I/O with examples



A file

- Provides *ideally* persistent storage of data
- Either **text** or **binary** format (text file lines terminated with an end of line marker, e.g. ‘\n’ on UNIX)
- Can access **serially** or **random** access (jumping or ‘seeking’ to the data we want)
- We need to **open** a file to access it, and **close** it afterwards

Serial access



Conceptually, there's a 'file pointer' that moves from the start to the end of the file as we read data from it

2. Opening and closing a file

```
// Declare a pointer to some FILE structure
FILE *fp;

// The API call to open a file
fp = fopen(“input.txt”, “r”);

Iff file pointer ‘valid’.  It could be ‘NULL’

fclose(fp);
```

Challenge 1: copying a file

1. Only copy a file that exists
2. Only write to a valid file
3. Copy data from the input to the output
4. Until we reach the end of the input file

Example, copying a file:

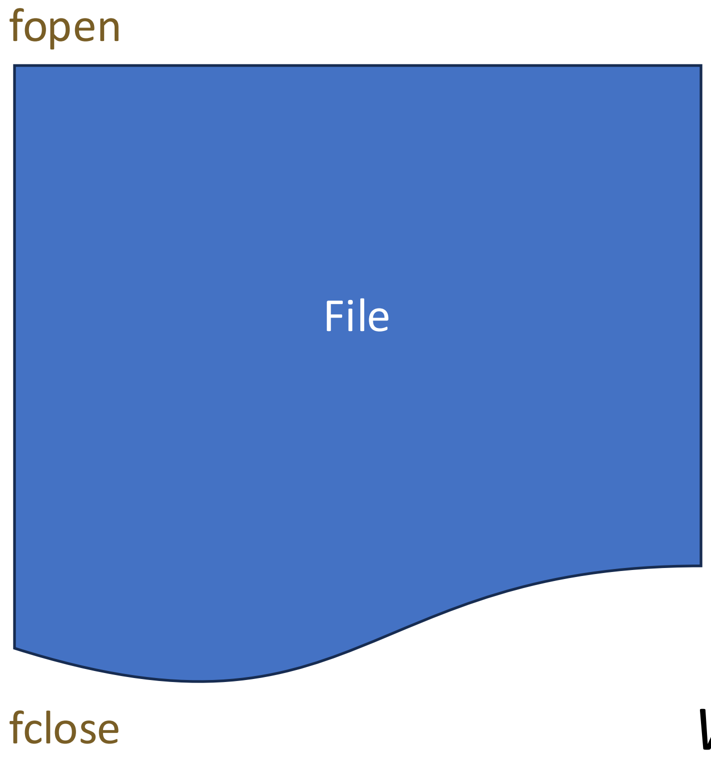
```
/* filecopy: copy file ifp to file ofp */

void filecopy(FILE *ifp, FILE *ofp)
{
  int c;

  while ((c = getc(ifp)) != EOF)
    putc(c, ofp);
}
```

```
int main()
{
  FILE *inFile = fopen("input.txt", "r"),
       *outFile = fopen("output.txt", "w");

  if (inFile && outFile) {
    filecopy(inFile, outFile);

    fclose(outFile);
    fclose(inFile);
  }

  return 0;
}
```

Same as saying if…
inFile != NULL && outFile != NULL

3. Working with lines of text (text files)

```
char *fgets(char *buf, int maxline, FILE *fp)
```



```
char line[80];
```

```
while (fgets(line, 80, fp) != NULL) {
  // Do something with line
}
```

What happens if the line of text in the file is longer than 80?

Note: others include fprintf, fscanf, fputs…

Challenge 2: Finding a pattern in a text file

- We need to find a pattern in the text file
- Read the file line by line (stdio.h)
- Search the line for the pattern (strings.h)
- Count how many times we find the pattern
- *What do we need to watch out for?*

A text file

```
46.4.84.242 - - [31/Jul/2011:16:40:46 +0100] "GET /2009/10/create-tv-episode-1/ HTTP/1.1" 200 1684578 "-" "Python-urllib/2.7"

66.249.66.26 - - [31/Jul/2011:19:15:02 +0100] "GET /carolynne/fashion-forecast/2011/06/14/going-native/?comments HTTP/1.1" 200 5791 "-" "Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)"

127.0.0.1 - - [31/Jul/2011:21:03:20 +0100] "OPTIONS * HTTP/1.0" 200 152 "-" "Apache/2.2.16 (Ubuntu) (internal dummy connection)"

127.0.0.1 - - [31/Jul/2011:21:58:42 +0100] "OPTIONS * HTTP/1.0" 200 152 "-" "Apache/2.2.16 (Ubuntu) (internal dummy connection)"

148.88.61.11 - - [01/Aug/2011:09:36:48 +0100] "GET /wp-content/themes/lusu/lusu_v3/js/jquery.hoverIntent.js HTTP/1.1" 200 1983 "http://housing.lusu.co.uk//" "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/534.30 (KHTML, like Gecko) Chrome/12.0.742.122 Safari/534.30"

148.88.61.11 - - [31/Jul/2011:15:54:38 +0100] "GET /2009/11/06/postgraduate-frequently-asked-questions/ HTTP/1.1" 200 5417 "http://housing.lusu.co.uk/2009/11/06/general-information/" "Mozilla/5.0 (Windows NT 6.1; WOW64; rv:5.0) Gecko/20100101 Firefox/5.0"
```

Worked example

Binary files and random access

Challenge 3: Record access (binary data)

- Need to read and write binary data to a file
- Use binary file IO (no text end of line markers)
- Assuming that the format of the data structure is the format of the file
- *Generally not the case… Why not?*
- N.B. Will need to validate that we comply to binary format specifications

4. Reading and writing 'records' (binary files)

Note: Need to open the file with "rb" or "wb"

```
size_t fread(void *ptr, size_t size, size_t nobj, FILE *fp);
```

- fread reads from stream into the array ptr at most nobj objects of size size
- fread returns the number of objects read; this may be less than the number requested
- feof and ferror must be used to determine status

- There's also:

```
size_t fwrite(const void *ptr, size_t size, size_t nobj, FILE *fp);
```

Binary I/O example

```c
#include <stdio.h>

struct image {
  int x,
      y;
  char data[100];
};

void read_image(struct image *img, FILE *fp)
{
  if (fp)
    fread(img, sizeof(struct image), 1, fp);
}

void write_image(struct image *img, FILE *fp)
{
  if (fp)
    fwrite(img, sizeof(struct image), 1, fp);
}
```

```c
int main()
{
  struct image testImage = { 10, 10, 0 };

  FILE *fp = fopen("test", "wb");

  if (fp) {
    write_image(&testImage, fp);

    fclose(fp);

    return 0;
  }

  return -1;
}
```

Creates…

- hexdump -C test

```
00000000  0a 00 00 00 0a 00 00 00  00 00 00 00 00 00 00 00
00000010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
*
00000060  00 00 00 00 00 00 00 00  00 00 00 00
0000006c
```

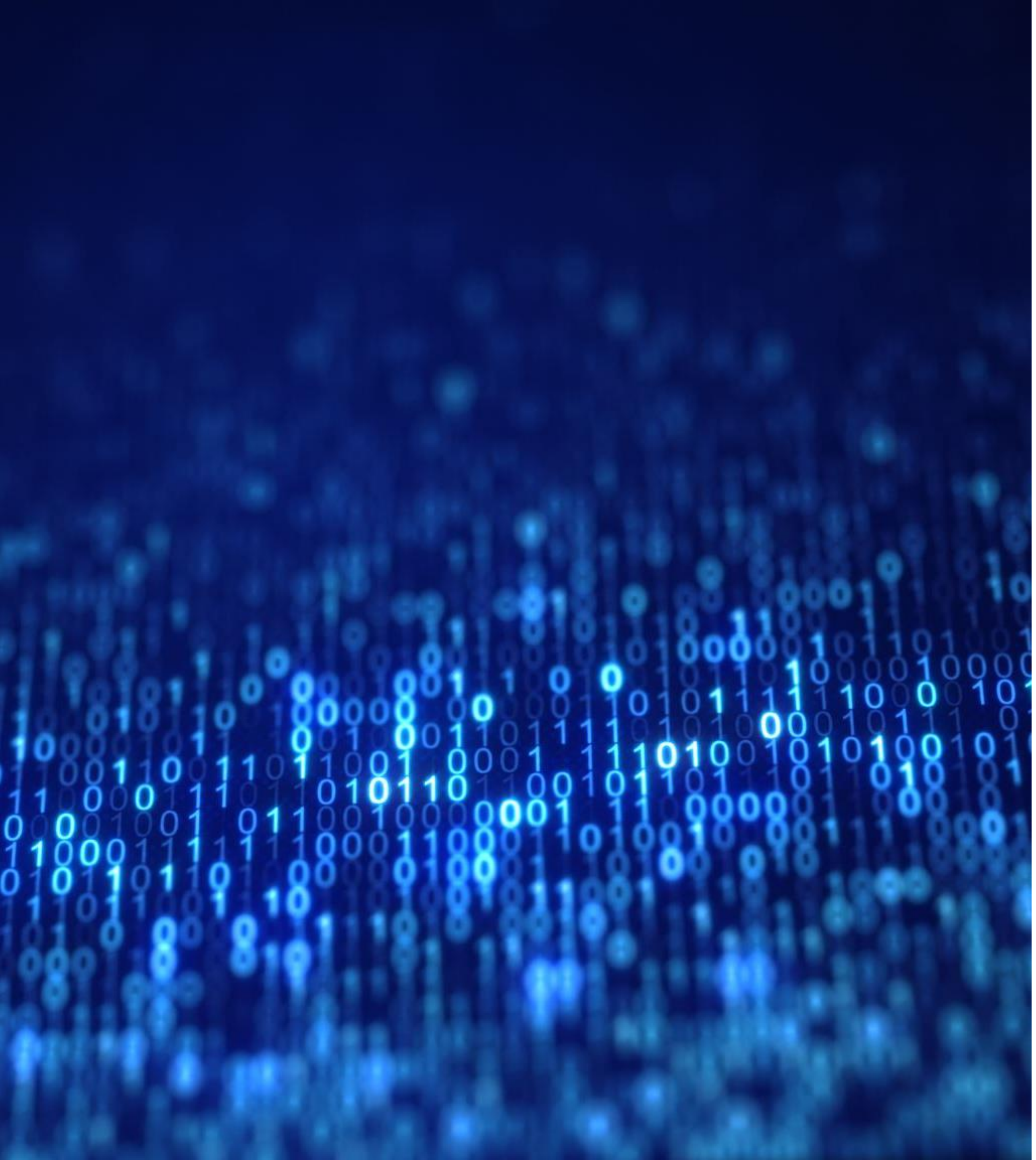
4b. Seeking to a location in a file

```
long ftell(FILE *stream);
```

- Find out where we are in ‘stream’ (bytes from start)

```
int fseek(FILE *stream, long offset, int whence)
```

- Seek to ‘offset’ from ‘whence’, generally SEEK_SET



Summary

- Presented 3 examples of file I/O (byte level, line by line, binary)
- Explained how to call library functions for file I/O using examples