Lecture 9
Files and Streams

TIC1001 Introduction to Computing and Programming I

What are files?

A file is simply a sequence of bytes

There are two types of files

- Text files
- Binary files

Text vs Binary files

Physically, both are the same

- just sequence of bytes

Just interpreted differently

- Each byte in an ASCII text file is interpreted as a character
- UTF-8 encoding is backward compatible with ASCII

Example: Text file

hello.c

```
0000000: 2369 6e63 6c75 6465 203c 7374 6469 6f2e  #include <stdio.
0000010: 683e 0d0a 0d0a 696e 7420 6d61 696e 2876  h>....int main(v
0000020: 6f69 6429 207b 0d0a 2020 2020 7072 696e  oid) {..    prin
0000030: 7466 2822 4865 6c6c 6f20 576f 726c 6421  tf("Hello World!
0000040: 5c6e 2229 3b0d 0a7d                      \n");..}
```

Example: Binary file

hello.png

```
0000000: 8950 4e47 0d0a 1a0a 0000 000d 4948 4452  .PNG........IHDR
0000010: 0000 0046 0000 000a 0803 0000 003e 6ad8  ...F.........>j.
0000020: a100 0000 6c50 4c54 45ff ffff d480 2b2b  ....lPLTE.....++
0000030: 80d4 ffd4 8000 0000 80d4 ffaa ffff ffaa  ................
0000040: 5580 2b2b ffff aa00 55aa 2b00 55d4 ffff  U.++....U.+.U...
0000050: 55aa ff55 0055 2b00 2bff ffd4 2b2b 80aa  U..U.U+.+...++..
…
0000110: 95a4 cbd6 5b57 6a56 9d37 b19b f444 ddeb  ....[WjV.7...D..
0000120: 4604 9a64 4c61 2200 1b8e e85b 4858 5805  F..dLa"....[HXX.
0000130: 3161 90f3 3d14 4689 ea63 6c0a 7f34 301e  1a..=.F..cl..40.
0000140: 0cb1 fa09 d34e 7918 caf9 26b4 df58 6295  .....Ny...&..Xb.
0000150: 3138 a750 a8d7 116f f470 c4b8 42a6 0a6e  18.P...o.p..B..n
0000160: 569f 98f0 cfee 8cb0 75fe 0031 9b0a df82  V.......u..1....
0000170: 41b4 1600 0000 0049 454e 44ae 4260 82    A......IEND.B`.
```

Same same, but different

All files have a specific encoding

- e.g. png, pdf, mp3, txt

A text file is just a file encoded in ASCII

- More universal format?
- Human readable?
- Convenience?

File I/O in C

<stdio.h>

Opening a File

```
FILE *fopen()( const char *filename,
               const char *mode );
```

- returns file pointer or `NULL` if there is error

`filename`

- The name of the file

`mode`

- `"r"` Open for reading.
- `"w"` Open and wipe (or create) for writing
- `"a"` Append. Open (or create) to write to end of file.
- `"r+"` Open for reading and writing.
- `"w+"` Open and wipe (or create) for reading and writing
- `"a+"` Open for reading (from beginning) and appending (to end)

Example: Opening a File

```
FILE *fp;
fp = fopen("hello.c", "r"); // open for reading
```

Closing a File

```
int fclose( FILE *fp );
```

- return `0` on success, or `EOF` if there is an error

Why do we need to explicitly close a file?

- Flush contents from buffer. Written content might be cached in a buffer.
- Memory leak. fp is still in use by the program. Might run out of file pointers.

Writing to a File

```
int fputc( int c, FILE *fp );
```

– writes char `c` into `fp`. Returns `c` if successful, `EOF` if error.

```
int fputs( const char *s, FILE *fp );
```

– writes string `s` into `fp`. Returns non-negative if successful, `EOF` if error.

```
int fprintf( FILE *fp, const char *format, … );
```

– same as `printf`, but writes to file `fp` instead.

Example: Writing to a File

```
int i = 42;
double pi = 3.141592654;
FILE *fp;

fp = fopen("example.txt", "w");
fputs("Hello World!", fp);
fputc('\n', fp);

fprintf(fp, "%d is the Answer\nNot %f", i, pi);

fclose(fp);
```

answer.txt

```
Hello World!
42 is the Answer
Not 3.141592654
```

Reading from a File

```
int fgetc( FILE *fp );
```

- returns the next character read from `fp`, `EOF` if error.

```
char * fgets( char *buf, int n, FILE *fp );
```

- reads until `'\n'` or n-1 chars from `fp` into `buf` and appends a `'\0'` to terminate `buf`.

```
int fscanf( FILE *fp, const char *format, … );
```

- reads and match to `format` and copy value into variables.
- returns number of matched items.
- Opposite of `fprintf`.

Format String

Tells fscanf how to interpret the input

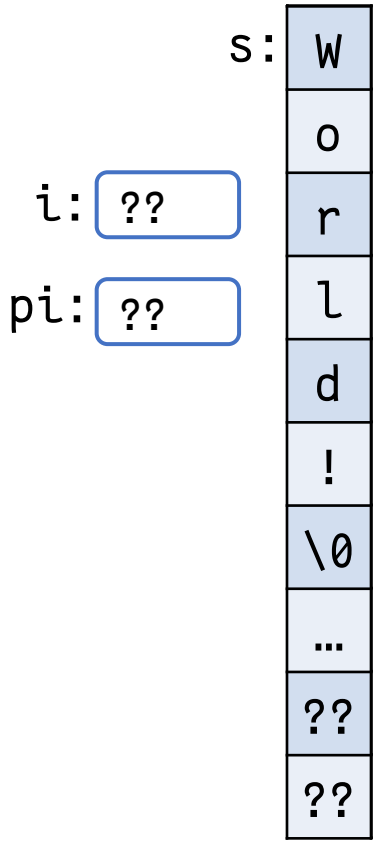
- **Whitespace characters** are ignored. Function will read and skip over whitespace characters until it reaches a non-whitespace character.
- **"Ordinary" characters** are matched and function continues reading the next character. If character does not match, function returns.
- **Format specifiers** are matched based on the type, and stored in the location provided in tadeonal arguments.

Example: fscanf

```
int i, r;
double pi;
char s[50];
FILE *fp = fopen( "example.txt", "r" );

fscanf(fp, "Hello %s", s);
r = fscanf(fp, "%d %lf", &i, &pi);

printf("%s %d %f %d\n", s, d, pi, r);

fscanf(fp, "%s", s);
printf("%s %s\n", s);

fgets(s, 49, fp);
```

answer.txt

```
Hello World!
42 is the Answer
Not 3.141592654
```

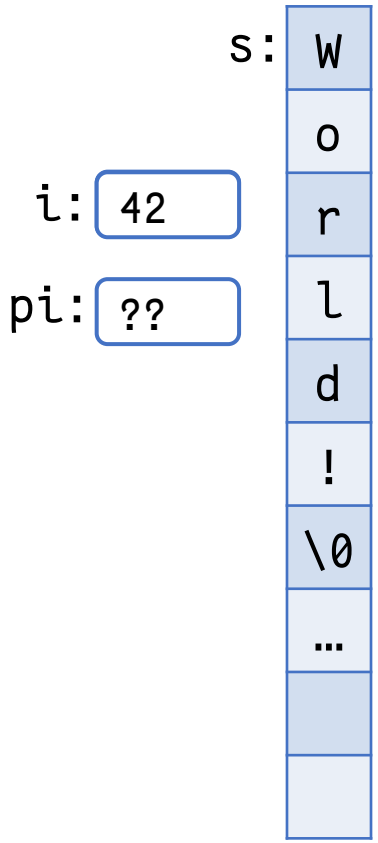


Example: fscanf

```
int i, r;
double pi;
char s[50];
FILE *fp = fopen( "example.txt", "r" );

fscanf(fp, "Hello %s", s);
r = fscanf(fp, "%d %lf", &i, &pi);

printf("%s %d %f %d\n", s, d, pi, r);

fscanf(fp, "%s", s);
printf("%s %s\n", s, &s[4]);

fgets(s, 49, fp);
```

answer.txt

```
Hello World!
42 is the Answer
Not 3.141592654
```



Example: fscanf

```
int i, r;
double pi;
char s[50];
FILE *fp = fopen( "example.txt", "r" );

fscanf(fp, "Hello %s", s);
r = fscanf(fp, "%d %lf", &i, &pi);

printf("%s %d %f %d\n", s, d, pi, r);

fscanf(fp, "%s", s);
printf("%s %s\n", s);

fgets(s, 49, fp);
```

answer.txt

```
Hello World!
42 is the Answer
Not 3.141592654
```

i: 42

pi: ??

s:

W
o
r
l
d
!
\0
…

output

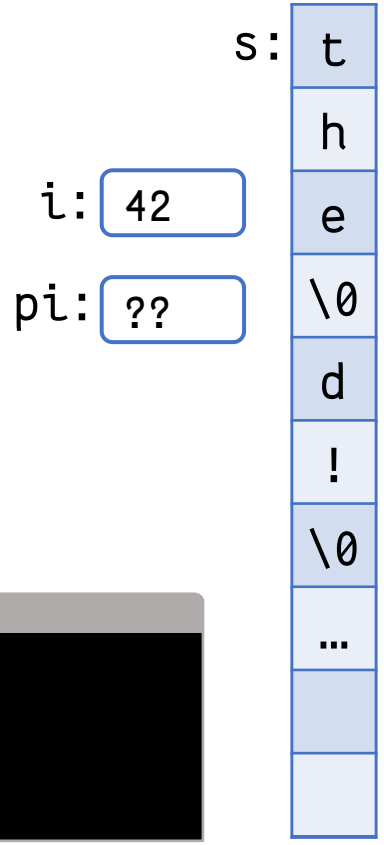
```
World! 42 0.123456 1
```

Example: fscanf

```
fscanf(fp, "%s", s);
printf("%s %s\n", s, &s[4]);

fgets(s, 49, fp);
printf("%s", s);

fscanf(fp, "%*s %lf", &pi);
printf("%f\n", pi);

fclose(fp)
```

answer.txt

```
Hello World!
42 is the Answer
Not 3.141592654
```



output

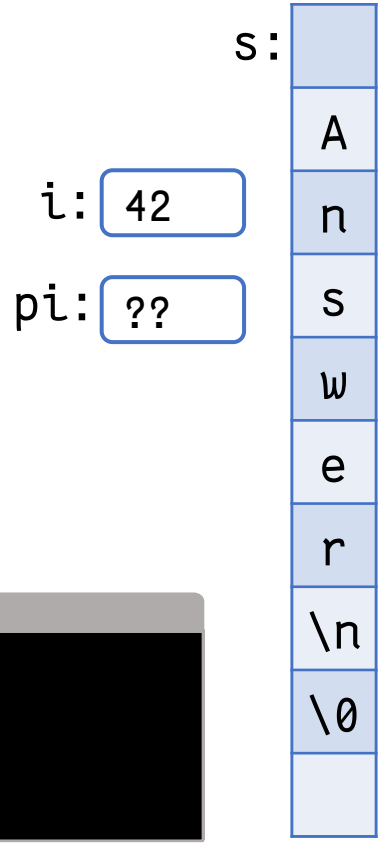
```
World! 42 0.123456 1
the d!
```

Example: fscanf

```
fscanf(fp, "%s", s);
printf("%s %s\n", &s[4]);

fgets(s, 49, fp);
printf("%s", s);

fscanf(fp, "%*s %lf", &pi);
printf("%f\n", pi);

fclose(fp)
```

answer.txt

```
Hello World!
42 is the Answer
Not 3.141592654
```



output

```
the d!
 Answer
```

Example: fscanf

```
fscanf(fp, "%s", s);
printf("%s %s\n", s, &s[4]);

fgets(s, 49, fp);
printf("%s", s);

fscanf(fp, "%*s %lf", &pi);
printf("%f\n", pi);

fclose(fp)
```

answer.txt

```
Hello World!
42 is the Answer
Not 3.141592654
```

i: 42

pi: 3.14…

s:

A
n
s
w
e
r
\n
\0

output

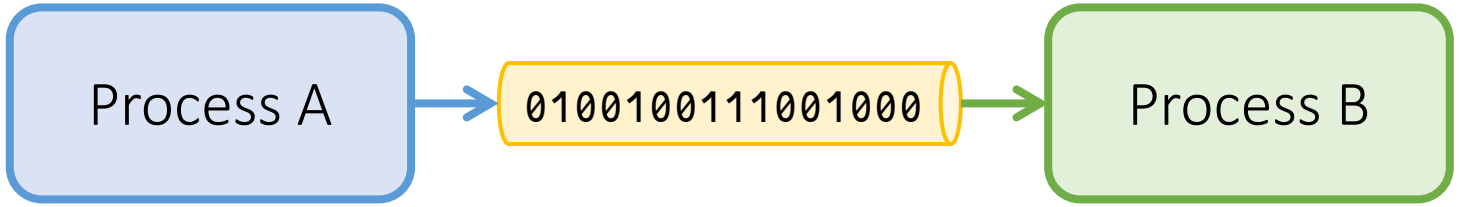
```
the d!
 Answer
3.141593
```

I/O Streams

A sequence of bytes

Streams

Used by processes to read or write data



A stream is thus a sequence of bytes

- sequence of bytes
- isn't that a file?

Streams == Files

Standard Streams

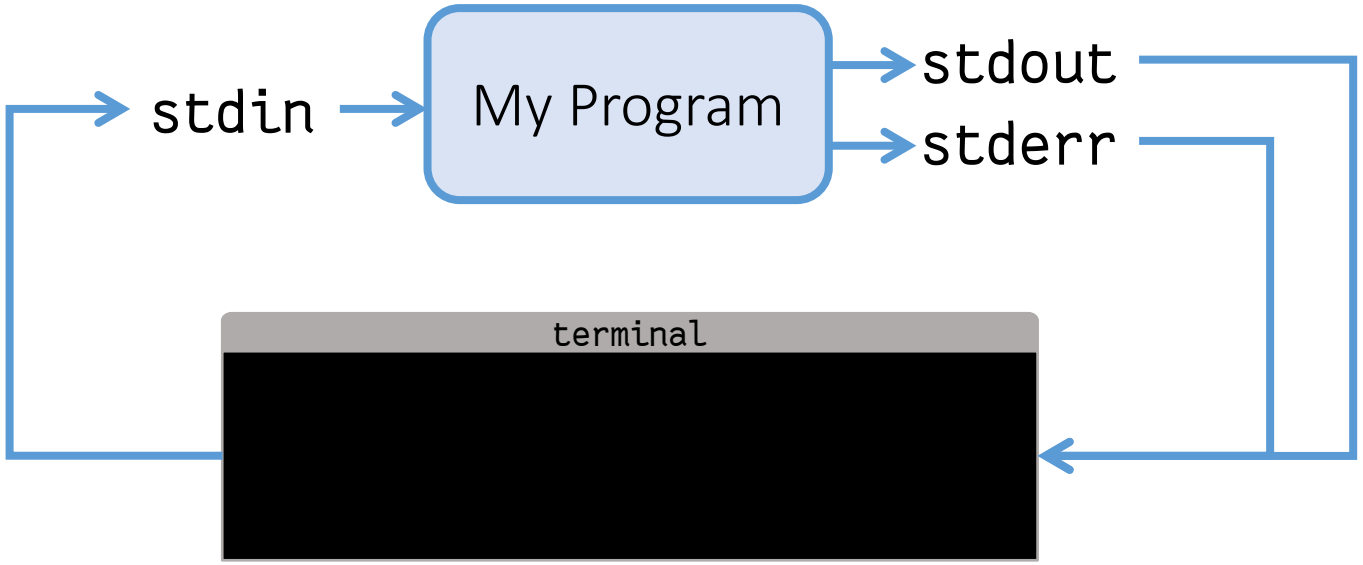
Every program has 3 standard streams

0. Standard Input (`stdin`)
1. Standard Output (`stdout`)
2. Standard Error (`stderr`)

Standard Streams

When executed in a terminal

– standard streams will be connected automatically



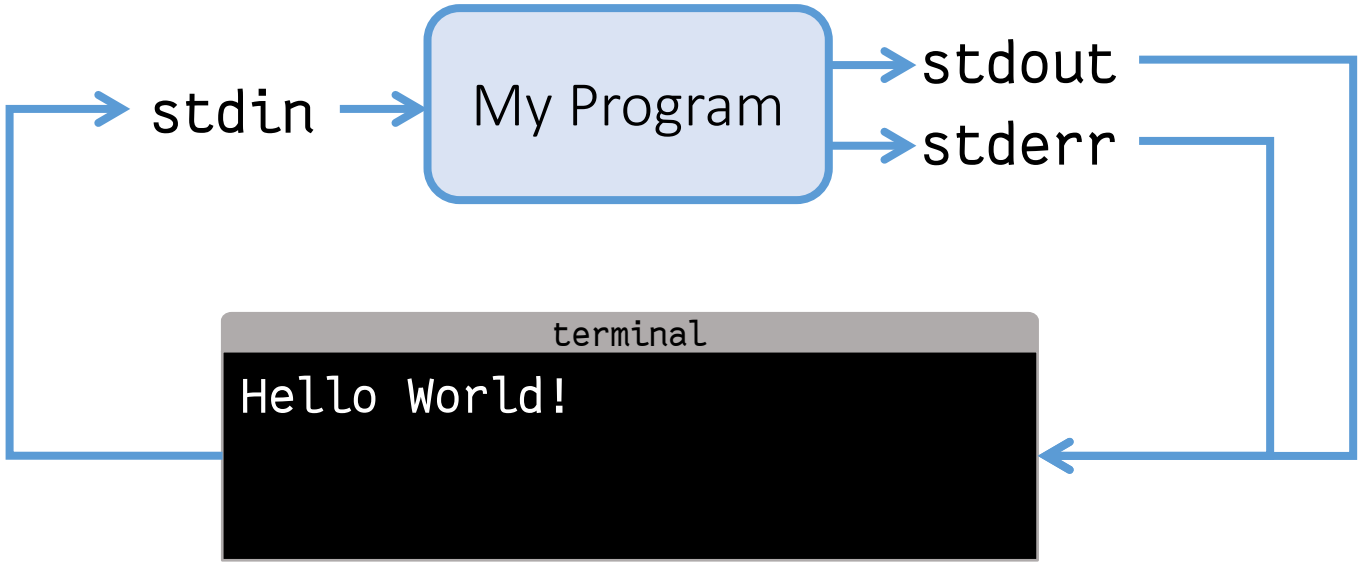
Output to Terminal

`stdin`, `stdout`, `stderr` are defined in stdio.h

```
fprintf(stdout, "Hello World!");
```

Isn't that just `printf`?!

- Yes, `printf` is `fprintf` with `stdout` as the file pointer



Input from Terminal

Then scanf just fscanf with stdin?

- Exactly. =)

```
int age;
printf("What is your age? ");
scanf("%d", &age);
```

terminal

```
What is your age? 42
```

- Note: In VSC, the output window cannot accept input. Use the terminal instead.

settings.json — Visual Studio Code

File Edit Selection View Go Debug Tasks Help

User Settings

coderunner terminal

2 Settings matched

USER SETTINGS

DEFAULT SETTINGS

Run Code configuration (2)

```
// Whether to run code in Integrated Terminal.
"code-runner.runInTerminal": false,

// For Windows system, replaces the Windows style drive
letter in the command with a Unix style root when using a
custom shell as the terminal, like Bash or Cgywin. Example:
Setting this to '/mnt/' will replace 'C:\path' with
'/mnt/c/path'
"code-runner.terminalRoot": "",
```

Place your settings here to overwrite the Default Settings.

```
// Place your settings in this file to overwrite the d
{
    "code-runner.saveAllFilesBeforeRun": true,
    "code-runner.runInTerminal": true,
    }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

1: powershell

```
Windows PowerShell
Copyright (C) 2016 Microsoft Corporation. All rights reserved.

Loading personal and system profiles took 922ms.
C:\Users\Wai Kay$
```

1 0

Ln 1, Col 1 (163 selected) Spaces: 4 UTF-8 LF JSON

Meet the Family

```
int printf( const *char format, … );
int fprintf( FILE *fp, const *char format, … );
int sprint( char *buff, const *char format, … );
int snprintf( char *buff, int buff_size,
              const *char format, … );

int scanf( const *char format, … );
int fscanf( FILE *fp, const *char format, … );
int sscanf( const char *buff,
            const *char format, … );
```

Why use stderr?

Why not just use stdout?

Because stderr is not buffered

- data is written immediately into the stream

The terminal or executing process can redirect the streams

- stderr to handle special output different from stdout

Redirecting Streams

When starting a program

- Attach the standard streams to other processes
- Or redirect to/from files

```
$ input.txt > ./my_prog > output.txt
```

- Use contents of input.txt as stdin
- Write contents of stdout to output.txt

```
$ ls | grep lecture | wc
```

- | character means pipe the stdout of one program into stdin of another

The C++ Way

with file streams

Stream Classes

C++ provide stream abstraction of files

`ofstream`

- (output) to write to files

`ifstream`

- (input) to read from files

`fstream`

- to both read and write from/to files

Opening a file

Using member functions

- open(filename, [mode])

```
ifstream myfile;
myfile.open("example.txt");
```

- opens a text file for reading

```
myfile.open("example.jpb", ios::binary);
```

- opens a binary file for reading

Closing a file

```
myfile.close();
```

Writing to text files

Same as writing to cout

```
myfile << "Hello World!" << endl;
myfile << my_variable;
```

Reading for text files

Reverse the direction

```
int i;
myfile >> i;
```

- To read primitive types

To read a line of text

```
string line;
getline(myfile, line);
```

- excludes the newline character

Summary

Files/Streams are just sequence of bytes

Open and close file

- `fopen` and `fclose`

To read and write

- Text mode: `fprintf` and `fscanf`
- Binary mode: `fputc` and `fgetc`