HIGH-LEVEL PROGRAMMING I

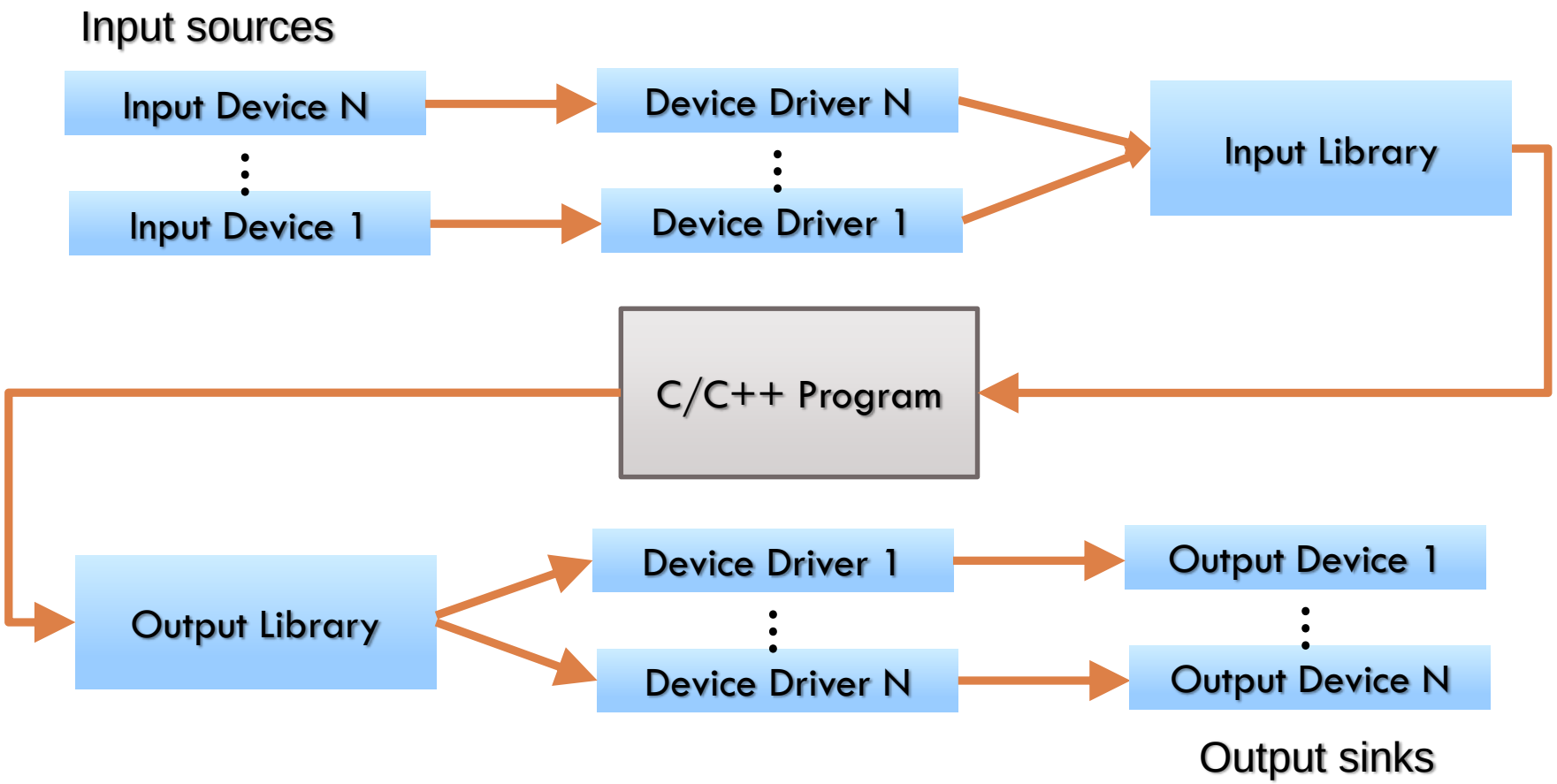
File Input/Output

by Prasanna Ghali

Topics

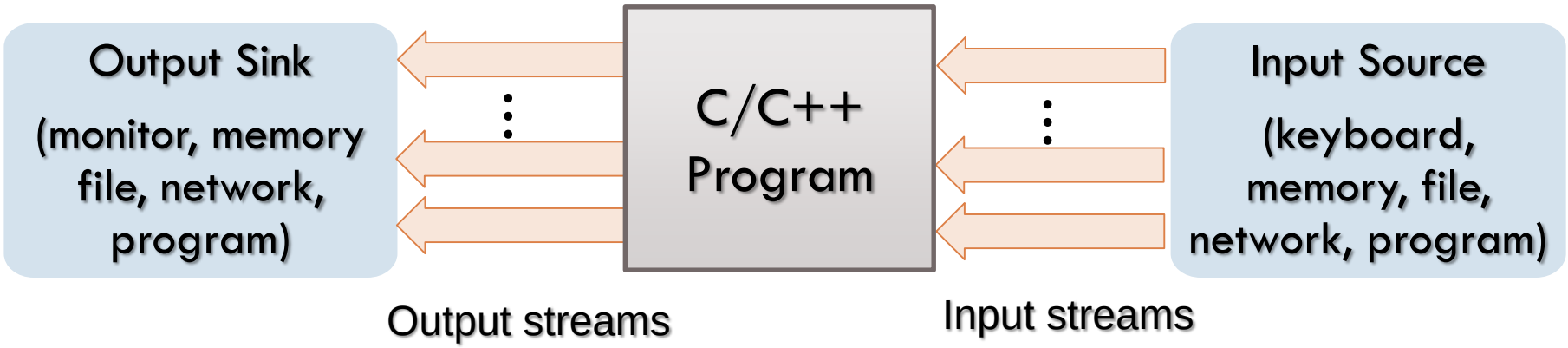
- Introduction to I/O: Stream model of I/O, File pointers, Standard streams, Text vs. binary streams
- Unformatted (character) I/O
- Redirection of I/O streams
- Formatted I/O
- I/O for binary streams

Input and Output



Stream Model

- *Stream* is abstraction for sequence of bytes consumed by program as input and generated by program as output



Header file

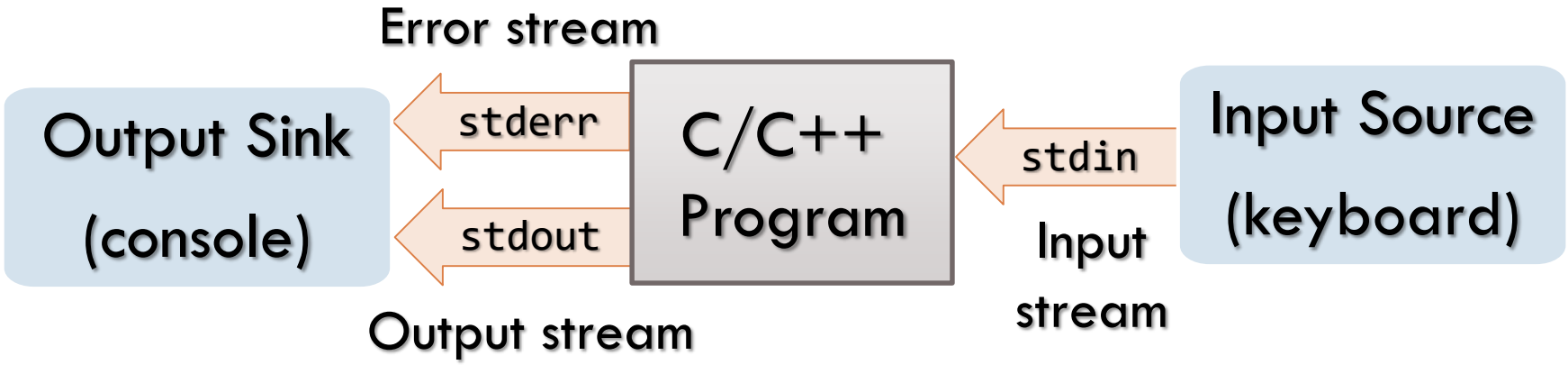
- Require `<stdio.h>` to be included to access C standard library input/output functions
- All these functions are compatible with 7-bit ASCII byte and UTF-8 encoding

File Pointers

- Streams accessed thro' objects of pointer type `FILE*`
  - `FILE*` referred to as *stream* or *file pointer* type
  - `<stdio.h>` declares structure type `FILE` – however programmers don't care about implementation details
  - File pointer points to structure that contains information about file: buffer location, current character position in buffer, whether file is being read from or written to, whether errors or end of file have occurred

Standard Streams (1/3)

- I/O library defines three streams ready to use by C/C++ programs - they've *static extent* (lifetime) and *external linkage* (public)



Standard Streams (2/3)

File pointer	Stream	Default meaning
`stdin`	Standard input	Keyboard
`stdout`	Standard output	Console
`stderr`	Standard error	Console

Standard Streams (3/3)


```c
// using standard streams
FILE *out = stdout, *in = stdin, *err = stderr;
char str[256];

fputs("Enter a string: ", out);
fgets(str, 256, in);
// remove newline from str
str[strlen(str)-1] = '\0';
fprintf(out, "You entered: %s\n", str);
fputs("This is a diagnostic message\n", err);
```

stdin is a pointer of type FILE *. The standard does not restrict the implementation beyond this, the details of what FILE is is entirely up to your compiler. It could even be an incomplete type (opaque).

Text vs. Binary Streams (1/2)

- `<stdio.h>` supports two kinds of streams: *text* and *binary*
- Consider number 23456 generated by program
  - Text stream stores sequence of five ASCII characters

  - Binary stream stores 16-bit value

Text vs. Binary Streams (2/2)

<table>
<tr><th>Text streams</th><th>Binary streams</th></tr>
<tr><td>Characters represented as bytes</td><td rowspan="2">Binary stream is anything that is not text stream: groups of bytes might represent other types of data, such as integers and floating-point numbers</td></tr>
<tr><td>Sequence of characters divided into lines</td></tr>
<tr><td>Each line consists of zero or more characters followed by newline character</td><td rowspan="2">Non-portable between platforms because of little- and big-endianness of processors</td></tr>
<tr><td>Newline character
Windows: <code>'\x0d''\x0a'</code>
UNIX & Mac OS: <code>'\x0a'</code></td></tr>
</table>

File Streams

- Just as with standard streams, reading from or writing to file has following general outline:
  - Open file stream using `fopen`
  - Read/write to file using `fgetc`, `fputc`, `fgets`, `fputs`, `fprintf`, `fscanf`
  - Close file stream using `fclose`

File Streams: Writing to File


```
// open an output file stream
FILE *out = fopen("duck", "w");

  // write text to output file stream
  fputs("Behold the duck.\n", out);
  // write more text to output file stream
  fprintf(out, "It does not cluck.\n");
  // write more text to output file stream
  fputs("A cluck it lacks", out);
  fputc('.', out); fputc('\n', out);

// now, close the output file stream
fclose(out);
```

File Streams: Reading from File


```
// open an input file stream
FILE *in = fopen("duck", "r");

  // read text from output file stream until
  // there's nothing more to read ...
  int ch, count = 0;
  while ( (ch = fgetc(in)) != EOF ) {
    fputc(ch, stdout);
    ++count;
  }

fclose(in); // now, close the input file stream
fprintf(stdout, "character count: %d\n", count);
```

File Streams: Appending to File


```
FILE *inout = fopen("the-duck", "r+");

  // set stream position indicator to end of file
  fseek(inout, 0L, SEEK_END);
  // append some text to end of file ...
  fprintf(inout, "It quacks.\n");
  fputs("It is specially fond\n", inout);
  fprintf(inout, "Of a puddle or pond.\n");

fclose(inout);
```

I/O Functions with Standard File Streams


```
// default stream is stdout
int putchar(int ch);
int puts(char const* string);
int printf(char const *format, ...);

// default stream is stdin
int   getchar(void);
char* gets(char *string);
int   scanf(char const *format, ...);
```

I/O Functions with Arbitrary File Streams


```
int putc(int ch, FILE *stream);
int fputc(int ch, FILE *stream);
int fputs(char const* string, FILE *stream);
int fprintf(FILE *stream, char const *format, ...);

int   getc(FILE *stream);
int   fgetc(FILE *stream);
char* fgets(char *string, int N, FILE *stream);
int fscanf(FILE *stream, char const *format, ...);
```

Input Redirection

- In shell, using < or 0< changes default meaning of stdin by substituting file for keyboard
- If executable *a.exe* uses <stdio.h> functions to read from stdin, then command line *a.exe 0<input-file* causes *a.exe* to read characters from *input-file*

Output Redirection

- In shell, using > or 1> changes default meaning of `stdout` by substituting file name for monitor
- If executable *`a.exe`* uses `<stdio.h>` to write to `stdout`, then command line *`a.exe 1>output-file`* causes *`a.exe`* to write characters to *`output-file`*

Redirecting stdout and stderr (1/3)


```
fputs("write to stdout\n", stdout);
fputs("write to stderr\n", stderr);
```

- Above code will print following text to monitor

```
write to stdout
write to stderr
```

- Command line *a.exe 1>output-file* will redirect stdout stream to file *output-file* while stderr stream continues to write to monitor

```
a.exe 1>output-file
```

Redirecting stdout and stderr (2/3)


```
fputs("write to stdout\n", stdout);
fputs("write to stderr\n", stderr);
```

- Command line *a.exe 2>error-file* will redirect stderr stream to file *error-file* while stdout stream continues to write to monitor

```
a.exe 2>error-file
```

Redirecting stdout and stderr (3/3)


```
fputs("write to stdout\n", stdout);
fputs("write to stderr\n", stderr);
```

- Following command line will redirect stderr stream to file *error-file* and redirect stdout stream to file *output-file*

```
a.exe 1>output-file 2>error-file
```

Redirecting stdin, stdout and stderr


```
char str[255];
fgets(str, 254, stdin);
fputs(str, stdout);
fputs(str, stderr);
```

- Following command line will redirect file *input-file* to stdin stream, redirect stderr stream to file *error-file* and redirect stdout stream to file *output-file*

```
a.exe 0<input-file 1>output-file 2>error-file
```