COMPSCI2030 Systems Programming

I/O

Yehia Elkhatib

University of Glasgow

stdin, stdout, stderr

- Linux standard streams
- `stdin  0` – The default source of data, usually the keyboard
- `stdout  1` – The default destination of output, usually the screen
- `stderr  2` – The default destination for error messages and other diagnostic warnings, usually the screen
- You can redirect any of these as you want
  - e.g. files, email

Many functions…

- Source:

https://en.wikipedia.org/wiki/C_file_input

- Experiment to build familiarity

	Byte character	Wide character	Description
File access	fopen		Opens a file (with a non-Unicode filename on Windows and possible UTF-8 filename on Linux)
	freopen		Opens a different file with an existing stream
	fflush		Synchronizes an output stream with the actual file
	fclose		Closes a file
	setbuf		Sets the buffer for a file stream
	setvbuf		Sets the buffer and its size for a file stream
	fwide		Switches a file stream between wide-character I/O and narrow-character I/O
Direct input/output	fread		Reads from a file
	fwrite		Writes to a file
Unformatted input/output	fgetc getc	fgetwc getwc	Reads a byte/wchar_t from a file stream
	fgets	fgetws	Reads a byte/wchar_t line from a file stream
	fputc putc	fputwc putwc	Writes a byte/wchar_t to a file stream
	fputs	fputws	Writes a byte/wchar_t string to a file stream
	getchar	getwchar	Reads a byte/wchar_t from stdin
	~~gets~~	–	Reads a byte string from stdin until a newline or end of file is encountered (deprecated in C99, removed from C11)
	putchar	putwchar	Writes a byte/wchar_t to stdout
	puts	–	Writes a byte string to stdout
	ungetc	ungetwc	Puts a byte/wchar_t back into a file stream
Formatted input/output	scanf fscanf sscanf	wscanf fwscanf swscanf	Reads formatted byte/wchar_t input from stdin, a file stream or a buffer
	vscanf vfscanf vsscanf	vwscanf vfwscanf vswscanf	Reads formatted input byte/wchar_t from stdin, a file stream or a buffer using variable argument list
	printf fprintf sprintf snprintf	wprintf fwprintf swprintf	Prints formatted byte/wchar_t output to stdout, a file stream or a buffer
	vprintf vfprintf vsprintf vsnprintf	vwprintf vfwprintf vswprintf	Prints formatted byte/wchar_t output to stdout, a file stream, or a buffer using variable argument list
	perror	–	Writes a description of the current error to stderr
File positioning	ftell ftello		Returns the current file position indicator
	fseek fseeko		Moves the file position indicator to a specific location in a file
	fgetpos		Gets the file position indicator
	fsetpos		Moves the file position indicator to a specific location in a file
	rewind		Moves the file position indicator to the beginning in a file
Error handling	clearerr		Clears errors
	feof		Checks for the end-of-file
	ferror		Checks for a file error
Operations on files	remove		Erases a file
	rename		Renames a file
	tmpfile		Returns a pointer to a temporary file
	tmpnam		Returns a unique filename

`printf()` and `scanf()`

- Part of the standard ANSI C library
- Are the most versatile means of communicating with programs
- Both work in similar ways
  - use control specifiers and a list of arguments

Specifier	Meaning	Types Converted
`%c`	Single character	`char`
`%d`	Signed decimal integer	`int`, `short`
`%ld`	Signed long decimal integer	`long`
`%f`	Decimal floating-point number	`float`, `double`
`%s`	Character string	`char` arrays
`%u`	Unsigned decimal integer	`unsigned int`, `unsigned short`
`%lu`	Unsigned long decimal integer	`unsigned_long`

scanf()

- Reads data from the keyboard according to a specified format
- Assigns the input data to one or more variables

```
int v;
float rate;
scanf("%d", &v);
scanf("%f", &rate);
```

reads an integer value from the keyboard and assigns it to the integer variable v

reads a floating-point value from the keyboard and assigns it to the variable rate

reads an integer and floating-point values separated by whitespace and assigns them to v and rate

```
scanf("%d %f", &v, &rate);
```

- Returns EOF if End-of-File has been reached or some other failure
- Can misbehave if it gets an unexpected input (e.g. char for int, …)
  - `fgetc` and `fgets` are more reliable alternative

`fgetc()` and `fputc()`

- Reads the next character from a given input stream
  - On success, returns a character as an unsigned char converted to an int
  - On failure, returns EOF
- Can be redirected to other streams
  - e.g. a file using `fopen()`
- `fputc` is the equivalent for output

```c
#include <stdio.h>
int main(void) {
    int f = fgetc(stdin);
    if (fgetc(stdin) == 0) {
        printf("Invalid input!");
        return 1;
    }
    printf("you have entered: %c\n", f);
}
```

fgets() and fputs()

- The equivalent for strings instead of characters

```c
#include <stdio.h>

int main(void) {
    FILE * f;
    char password[20];
    printf("Speak, friend, and enter");
    fgets(password,20,stdin);
    f = fopen("mypassword.txt","a");
    fputs(password,f);
    fclose(f);
    return 0;
}
```

puts() and gets()

- Used to display text messages on the screen
- But it can not display numeric variables
  - limited capability, but less overhead
- puts
  - takes a single string as its argument and displays it
  - it automatically adds a newline at the end

```
puts("Hello, world.");
```

- gets
  - reads a line from stdin
  - stores it in the string pointed to

```
char str[50];
puts("Enter a string :");
gets(str);
```

Escape Sequences

- Provide special formatting and printing control

Sequence	Meaning
\a	Bell (alert)
\b	Backspace
\f	Form feed
\n	Newline
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab
\\	Backslash
\?	Question mark
\'	Single quotation
\"	Double quotation

Trigraph Sequences

- Special sequences of characters in *your source file* that will be interpreted to mean something else
- Are interpreted at compile time, and will be converted

```
printf("??(WOW??)");
printf("[WOW]");
```

```
printf("???-");
printf("?~");
```

Code	Character Equivalent
??=	#
??(	[
??/	\
??)	]
??'	^
??<	{
??!	|
??>	}
??-	~

Question time

1. What is the difference between `puts()` and `printf()`?