Buffered/Standard I/O

GEORGIOS PORTOKALIDIS

CS-392-A SYSTEMS PROGRAMMING

Standard I/O Library

Part of ISO C standard

Based on C streams

C streams operate on FILE objects and file pointers FILE *

- Not an object but a typedef defined type!

Related variables, types, and functions are declared in stdio.h

Three streams are predefined and automatically available to a process through global variables:

- stdin – standard input
- stdout – standard output
- stderr – standard error

Standard I/O uses by terminal when a program is launched through it

Reading and Writing Files

FILE *fopen(const char *path, const char *mode)

Opens the file whose name is the string pointed to by path and associates a stream with it. The string pointed to by mode specifies the mode in which to open the file

Stream Modes

type	Description
`r` or `rb`	open for reading
`w` or `wb`	truncate to 0 length or create for writing
`a` or `ab`	append; open for writing at end of file, or create for writing
`r+` or `r+b` or `rb+`	open for reading and writing
`w+` or `w+b` or `wb+`	truncate to 0 length or create for reading and writing
`a+` or `a+b` or `ab+`	open or create for reading and writing at end of file

Stream Modes

type	Description
`r` or `rb`	open for reading
`w` or `wb`	truncate to 0 length or create for writing
`a` or `ab`	append; open for writing at end of file, or create for writing
`r+` or `r+b` or `rb+`	open for reading and writing
`w+` or `w+b` or `wb+`	truncate to 0 length or create for reading and writin
`a+` or `a+b` or `ab+`	open end o

UNIX does not treat binaries differently so in most cases using *b* does not affect the stream in any way, but it does increase code readability

Mode Restrictions

Restriction	r	w	a	r+	w+	a+
file must already exist	•			•		
previous contents of file discarded		•			•	
stream can be read	•			•	•	•
stream can be written		•	•	•	•	•
stream can be written only at end			•			•

Figure 5.3 Six ways to open a standard I/O stream

Basic Input Functions

int fgetc(FILE *stream);

Read and return one character from the stream or EOF on end of file or error

int getc(FILE *stream);

Same as above but may not be a function, but an optimized macro

int getchar(void);

Same as above for stdin

Basic Output Functions

int fputc(int c, FILE *stream);

Write character c (cast to unsigned char) to stream, returns the character return or EOF on error

int putc(int c, FILE *stream);

Same as above but may not be a function, but an optimized macro

int putchar(int c);

Same as above for stout

Discerning Error from EOF

int feof(FILE *stream);

Return nonzero if EOF was encountered

int ferror(FILE *stream);

Return nonzero if an error occurred in the stream

void clearerr(FILE *stream);

It clears the end-of-file and error indicators for the stream pointed to by stream.

echo Program

```
#include "apue.h"

int
main(void)
{
    int     c;

    while ((c = getc(stdin)) != EOF)
        if (putc(c, stdout) == EOF)
            err_sys("output error");

    if (ferror(stdin))
        err_sys("input error");

    exit(0);
}
```

Figure 5.4 Copy standard input to standard output using `getc` and `putc`

Reading a Line of Text into a String

```
/* Reads a string from stdin until it encounters a new line character or until
 * (len - 1) characters have been read.
 * Return the number of characters read, excluding '\0'.
 */
size_t my_getline(char s[], int len)
{
	int i, c;

	for (i = 0; i < (len - 1) && (c = getchar()) != EOF && c != '\n'; i++)
		s[i] = c;
	if (c == '\n') {
		s[i] = c;
		++i;
	}
	s[i] = '\0';
	return i;
}
```

Line-at-a-time I/O

char *fgets(char *s, int size, FILE *stream);

Reads the next character from stream and returns it as an unsigned char cast to an int, or EOF on end of file or error.

int fputs(const char *s, FILE *stream);

Writes the string s to stream, without its terminating null byte ('\0').

int puts(const char *s);

writes the string s and a trailing newline to stdout.

I/O Beyond String and Characters

size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);

size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream);

```
struct record {
        char name[128];
        char surname [128];
        char ssn[16];
};
```

```
struct record rec;
size_t len;

len = fread(&rec, sizeof(rec), 1, fp);
If (len < 1 && feof(fp) == 0)
        return -1; /* error */
```

I/O Beyond String and Characters

size_t fread(void *ptr**, size_t** size**, size_t** nmemb**, FILE** *stream**);**

size_t fwrite(const void *ptr**, size_t** size**, size_t** nmemb**, FILE** *stream**);**

```
char buf[4096];

len = fread(&rec, 1, sizeof(buf), fp);
If (len < sizeof(buf)  && feof(fp) == 0)
            return -1; /* error */
```

Improved echo

```
#include "apue.h"

int
main(void)
{
    char    buf[MAXLINE];

    while (fgets(buf, MAXLINE, stdin) != NULL)
        if (fputs(buf, stdout) == EOF)
            err_sys("output error");

    if (ferror(stdin))
        err_sys("input error");

    exit(0);
}
```

Figure 5.5 Copy standard input to standard output using `fgets` and `fputs`

Not At the Beginning, Nor at the End

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

- whence: SEEK_SET, SEEK_CUR, or SEEK_END

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

void rewind(FILE *stream**);**

int fgetpos(FILE *stream**, fpos_t** *pos**);**

int fsetpos(FILE *stream**, fpos_t** *pos**);**

Closing Streams

int fclose(FILE *fp);

Flushes the stream pointed to by fp (writing any buffered output data using fflush(3)) and closes the underlying file descriptor

Warning: If you are writing or appending to a file remember to always close it or otherwise your changes may not be written to disk

Formatted I/O

int printf(const char *format**, ...);**

int fprintf(FILE *stream**, const char** *format**, ...);**

int sprintf(char *str**, const char** *format**, ...);**

int snprintf(char *str**, size_t** size**, const char** *format**, ...);**

Printing anything as a string

Example: printf("%d %s %f %llu\n", 10, "hello", 1.10f, 100LLU);

Length Modifiers

Length modifier	Description
`hh`	signed or unsigned `char`
`h`	signed or unsigned `short`
`l`	signed or unsigned `long` or wide character
`ll`	signed or unsigned `long long`
`j`	`intmax_t` or `uintmax_t`
`z`	`size_t`
`t`	`ptrdiff_t`
`L`	`long double`

Figure 5.8 The length modifier component of a conversion specification

Format String

Format strings are very expressive

Do not use user supplied strings as format strings