

# C File Input and Output (I/O)

CSE303

Todd Schiller

November 9, 2009

# Lecture goal

Build a practical toolkit for working with files

# Files in C

- `#include <stdio.h>`
- **FILE** object contains file stream information
- Special files defined in `stdio`:
  - **stdin**: Standard input
  - **stdout**: Standard output
  - **stderr**: Standard error
- **EOF**: end-of-file, a special negative integer constant

# Opening and closing a file

# Opening a file

**FILE\*** **fopen**(char\* filename, char\* mode)

mode strings	
"r"	Open a file for <b>reading</b> . The file must exist.
"w"	Create an empty file for <b>writing</b> . If a file with the same name already exists its content is erased and the file is treated as a new empty file.
"a"	<b>Append</b> to a file. Writing operations append data at the end of the file. The file is created if it does not exist.

## OUTPUT

- If **successful**, returns a pointer to a FILE object
- If **fails**, returns **NULL**

# Opening a file

```
FILE *fp = fopen("myfile.txt", "r");

if (fp == NULL){
    //report error and try to recover
}else{
    //do something with the file
}
```

# Closing a file

```
int fclose ( FILE * stream )
```

## OUTPUT

- On **success**, returns 0
- On **failure**, returns **EOF**

# Reading from a file

# Reading a character from a file

```
int fgetc ( FILE * stream )
```

## OUTPUT

- On **success**, returns the next character
- On **failure**, returns **EOF** and sets end-of-file indicator

Note: **EOF** < 0; so you can test for failure by checking if the output of **fgetc** is negative

# Reading a character from a file

```
UW\n
CSE\n
```

```
FILE *fp = ...
```

```
...
```

```
while ( (c = fgetc(fp)) != EOF) {  
    printf("char: '%c'\n", c);  
}
```

```
char:'U'  
char:'W'  
char:'  
'  
  
char:'C'  
char:'S'  
char:'E'  
char:'  
'
```

# Un-reading a character

```
int ungetc ( int character, FILE * stream );
```

## EFFECT

- *Virtually* puts a character back into the file
- Doesn't modify the file
- May be a different character than the last one read

## OUTPUT

- On **success**, returns the character that was pushed
- On **failure**, returns **EOF** and sets end-of-file indicator

# Un-reading a character

```
...  
  
FILE *fp = ...  
int c;  
...  
  
while ((c = fgetc(fp)) != EOF){  
    if (c == 'a'){  
        ungetc('4', fp);  
    }else{  
        printf("read char %c\n", c);  
    }  
}  
...  
...
```

# Reading a string from a file

```
char * fgets ( char * str, int num, FILE * stream )
```

## BEHAVIOR

- Reads at most ( $num-1$ ) characters from the *stream* into *str*
- Null-terminates the string read (adds a '\0' to the end)
- Stops after a newline character is read
- Stops if the end of the file is encountered
  - Caveat: if no characters are read, *str* is not modified

## OUTPUT

- On **success**, a pointer to *str*
- On **failure**, returns **NULL**

# Reading a string from a file

```
#define BUFFER_SIZE 80
...

FILE *fp = ...

...

char buf[BUFFER_SIZE];
fgets(buf, BUFFER_SIZE, fp);
```

# Are we at the end of a file?

```
int feof ( FILE * stream )
```

## OUTPUT

- If at the end of the file, returns a non-zero value
- If not at the end of the file, returns 0

Note: checks the end-of-file indicator which is set by fgets, fgetc, etc.

# Are we at the end of a file?

```
FILE *fp = ...
```

```
...
```

```
while (!feof(fp)){  
    //read something  
}
```

# Are we at the end of a file

```
UW\n  
CSE\n  
\n
```

```
while ( !feof(fp)){  
    fgets(buf,BUFFER_SIZE,fp);  
    printf("Read line: %s\n",buf);  
}
```

```
Read line: UW  
  
Read line: CSE  
  
Read line: CSE
```

# Reading formatted data from a file

```
int fscanf ( FILE * stream, const char * format, ... )
```

## INPUT

- Format string is analogous to **printf** format string
  - %d for integer
  - %c for char
  - %s for string
- Must have an argument for each format specifier

## OUTPUT

- On **success**, returns the number of items read; can be 0 if the pattern doesn't match
- On **failure**, returns **EOF**

# Reading formatted data from a file

```
1 string1
42 string2
54 string3
...
```

```
FILE *fp = ...
```

```
char buf[BUFFER_SIZE];
int num;
```

```
while (!feof(fp)){
    fscanf(fp, "%d %s", &d, buf)
    //do something
}
```

# What's wrong with this?

```
WA  
MO  
...
```

...

```
FILE *fp = ...  
char state[3];
```

```
while(fscanf(fp, "%s", state) != EOF);  
    printf("I read: %s\n", state);  
}
```

...

# What's wrong with this?

```
WA  
MO  
Florida  
...
```

...

```
FILE *fp = ...  
char state[3];
```

```
while(fscanf(fp, "%s", state) != EOF);  
    printf("I read: %s\n", state);  
}
```

...

# Buffer overruns

- Data is written to locations past the end of the buffer
- **Hackers** can exploit to execute arbitrary code
- User can *always* create an input longer than **fixed** size of buffer

**Don't use: scanf, fscanf, gets**

- Use functions that limit the number of data read

**Use: fgets**

# Writing to a file

# Writing a character to a file

```
int fputc ( int character, FILE * stream )
```

## OUTPUT / EFFECT

- On **success**, writes the character to the file and returns the character written
- On **failure**, returns **EOF** and sets the error indicator

Note: **EOF** < 0; so you can test for failure by checking if the output of **fputc** is negative

# Writing a character to a file

...

```
FILE *fp = fopen("myfile.txt", "w");
char str[] = "Huskies > Trojans";
int i;

if (fp != NULL){
    for (i = 0; i < strlen(str); i++){
        if (fputc(str[i], fp) < 0){
            // Something bad happened
        }
    }
    fclose(fp);
}
```

# Writing a string to a file

```
int fputs ( const char * str, FILE * stream )
```

## OUTPUT / EFFECT

- On **success**, writes the string to the file and returns a non-negative value
- On **failure**, returns **EOF** and sets the error indicator

Note: **EOF** < 0; so you can test for failure by checking if the output of **fputs** is negative

# Writing a string to a file

...

```
FILE *fp = fopen("myfile.txt", "w");  
char str[] = "Huskies > Trojans";
```

```
if (fp != NULL){  
    if (fputs(str, fp) < 0){  
        // Something bad happened  
    }  
    fclose(fp);  
}
```

...

# Writing a formatted string to a file

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

## INPUT

- The format string is same as for **printf**
- Must have an argument for each specifier in the format

## OUTPUT / EFFECT

- On **success**, returns the number of character written
- On **failure**, returns a negative number

# Writing a formatted string to a file

```
...  
FILE *fp = fopen("myfile.txt", "w");  
int h = 16;  
int t = 13;  
char str[] = "Huskies > Trojans";  
  
if (fp != NULL) {  
    fprintf(stdout, "%s | Score: %d to %d\n", str, h, t);  
    fclose(fp);  
}  
...
```

Huskies > Trojans | Score: 16 to 13

# Error Handling

# Was there an error?

```
int ferror ( FILE * stream )
```

## **OUTPUT**

- If the error indicator is set, returns a non-zero integer
- Otherwise returns 0

# Was there an error?

...

```
FILE *fp = ...
```

...

```
fputs("I love CSE303", fp);
```

```
if (ferror(fp)){  
    //Report error and recover  
}
```

...

# Printing an error description

```
void perror ( const char * str )
```

## **EFFECT**

- Prints a description of the file error prefixed by the supplied string *str* and a “:”
- Can pass **NULL** to just print the error description

# Printing an error description

...

```
FILE *fp = ...
```

...

```
fputs("I love CSE303", fp);
```

```
if (ferror(fp)){
```

```
    perror("Could not tell the world how I feel");
```

```
    //recover from the error
```

```
}
```

...

# Clearing error indicator

```
void clearerr ( FILE * stream );
```

## **EFFECT**

- Clears error indicator
- Clears end-of-file indicator

# Moving around a file

# Going to the beginning of a file

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

## **EFFECT**

- Moves file pointer to beginning of file
- Resets end-of-file indicator
- Reset error indicator
- Forgets any virtual characters from **ungetc**

# Moving to a location

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

## INPUT

- Offset is in bytes
- Origin can be
  - `SEEK_SET`: beginning of the file
  - `SEEK_CUR`: current file position
  - `SEEK_END`: end of the file

## OUTPUT / EFFECT

- On **success**
  - returns 0
  - resets end-of-file indicator
  - forgets any virtual characters from **ungetc**
- On **failure**, returns 0

# Moving to a location

...

```
FILE * fp = fopen("myfile.txt" , "w" );  
fputs ( "This is an apple." , fp );  
fseek ( fp , 9 , SEEK_SET );  
fputs ( " sam" , fp );  
fclose ( fp );
```

...

This is a sample

# Working with the filesystem

# Removing a file

```
int remove ( const char * filename )
```

## OUTPUT

- On **success**, returns 0
- On **failure**, returns a non-zero value

# Renaming a file

```
int rename ( const char * oldname, const char * newname );
```

## OUTPUT

- On **success**, returns 0
- On **failure**, returns a non-zero value

# Binary files

# Opening binary files

- Add “b” to the **fopen** mode string
  - “rb” : read a binary file
  - “wb” : write a binary file
  - “ab” : append to a binary file

# Writing to binary files

```
size_t fwrite (const void * ptr, size_t size, size_t count, FILE * stream)
```

## INPUT

- A *ptr* to an array of elements (or just one)
- The size of each element
- The number of elements

## OUTPUT

- Returns the number of elements written
- If return value is different than *count*, there was an error

# Writing to binary files

...

```
FILE *fp = fopen("myfile.bin", "wb");
```

...

```
int nums[] = {1,2,3};  
fwrite(nums, sizeof(int), 3, fp);
```

```
double dub = 3.1;  
fwrite(&dub, sizeof(double), 1, fp);
```

...

# Reading binary files

```
size_t fread ( void * ptr, size_t size, size_t count, FILE * stream )
```

## INPUT

- A *ptr* to some memory of size at least (*size \* count*)
- The size of each element to read
- The number of elements to read

## OUTPUT

- Returns the number of elements read
- If return value is different than *count*, there was an error or the end of the file was reached

# Reading binary files

...

```
FILE *fp = fopen("myfile.bin", "rb");
```

...

```
int nr;
```

```
int nums[3];
```

```
nr = fread(nums, sizeof(int), 3, fp);
```

```
//Check for errors
```

```
double dub;
```

```
nr = fread(&dub, sizeof(double), 1, fp);
```

```
//Check for errors
```

...

# Overview

# Where can I learn more?

<http://www.cplusplus.com/reference/clibrary/cstdio/>