Today's Goals

- Quick Intro to Strings
- File I/O
  - File Handle and File Pointer
  - Character and Line I/O
  - File Positioning

Section 1

File I/O: Streams

- Java's I/O streams consist of bytes and Unicode characters.
- C's streams consist of solely bytes.
- **`<stdio.h>`** provides 3 standard streams:
  - **`stdin`** – keyboard input
  - **`stdout`** – screen output
  - **`stderr`** – screen error output
- Unix allows changing of default meanings through redirection – more details later

Text Files / Binary Files

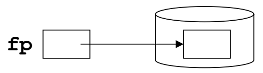
- **`<stdio.h>`** supports both
- Any file is just a sequence of bytes
- In a text file, a byte is always a character

text	00000011	0000010	00000111	00000110	00000111
	'3'	'2'	'7'	'6'	'7'
binary	01111111	11111111			

- In this lecture we cover text file I/O

File Pointers

- Accessing a stream in C is done through file pointers:
  - A variable *point*ing to a file ⇒ **`FILE *fp;`**



  - The type **`FILE`** is defined in **`stdio.h`**
  - Certain streams have predefined pointers with standard names – **`stdin`**, **`stdout`** and **`stderr`**
- **`fflush(FILE *fp)`**

Section 2

Basic File Operations

- Declaration (of a file pointer)
- Opening/Closing
- Reading/Writing

Opening/Closing a File

```
fp = fopen("file.dat", "r")
```

filename

Returns the null pointer **`NULL`** (zero) on error, i.e. trying to read a file that doesn't exist.

`"r"` – reading
`"w"` – writing (overwriting)!
`"a"` – appending

- Full path name as well as the filename may be included btw the quotes
- Always test against NULL
- Closing a file when done: **`fclose(fp);`**

Character I/O

- Reading – returns char read or EOF

```
int fgetc(FILE *fp)
int getc(FILE *fp) // macro
int getchar() <==> int fgetc(stdin)
```

- Writing – returns char written

```
int fputc(int c, FILE *fp)
int putc(int c, FILE *fp) // macro
int putchar(int c) <==> int fputc(…, stdin)
int ungetc(int c, FILE *fp)
```

Example: File Copy by Char

```
FILE *in, *out;
// open both src and dest files
as // in and out, respectively

while ((c = fgetc(in)) != EOF) {
    fputc(c, out);
}
```

filecopy.c

Section 3

Character and String

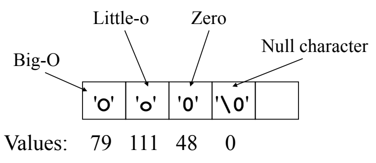
- String is not a special type
  - An **array** of characters
  - Terminated with a special, null character
- Null character
  - Its integer value is 0.
  - Its C representation is `'\0'`.

'\0' ≠ '0' (zero)
'\0' ≠ '\n'

- E.g., **"abc"** is internally

'a'	'b'	'c'	'\0'	

Example



Declaration/Initialization

- Declaration: **char s[5];**
- Initialization: **char t[] = "abc";**

Note: Strings can**not** be assigned using '=' (except initialization).

```
char t[] = "abc";
printf("%c", t[0]);  /* prints a */
printf("%d", t[0]);  /* prints 97 */

printf("%c", t[3]);  /* ??? */
printf("%c", t[4]);  /* ??? */
```

String Output

- Use **printf** with the **%s** specification

  Prints character elements until **\0** is reached

```
char s[] = "abc";

printf("%s", s);  /* prints abc */

printf("string: >%s<", s);
  /* prints string: >abc< */
```

Also possible: %10s

String Input

- The **gets** function

```
#define BUFLEN 200

int main() {

  char buf[BUFLEN];

  gets(buf);
  printf("string: >%s<", buf);
}
```

allocate a large buffer
(e.g., more than 2 lines)

Notes

- **gets** deletes **\n** from input**.**
- If the user presses ENTER without any other characters, the first position will be the null character (called 'empty string').
- In case the user enters a string longer than the buffer, **gets** may cause a serious run-time error.
- Avoid **gets** if you can.

Section 4

Line I/O: Input

- Reading – returns pointer to string read, NULL if end of file

  char* fgets(char *buf, int max, FILE *fp)
- Strings are character arrays in C
- **max** indicates the maximum number of characters to be read.
- **max** should be **1** less than the length of **buf**!
- **gets** is equivalent to **fgets(…, stdin)**

Line I/O: Output

- Writing – returns number of chars written

  int fputs(char *buf, FILE *fp)

Example: File Copy by Line

```
int main() {
  char buf[BUFLEN], inFile[BUFLEN], outFile[BUFLEN];
  FILE *in, *out;

  printf("Enter source filename: ");
  fgets(inFile,BUFLEN-1,stdin); trim_newline(inFile);
  // get outFile as well from user

  in = fopen(inFile, "r");
  out = fopen(outFile, "w");

  if ((in == NULL) || (out == NULL)) {
    printf("*** File open error\n");
    return;
  }

  /* NULL returned at EOF */
  while (fgets(buf, BUFLEN-1, in) != NULL) {
    fputs(buf, out);
  }

  fclose(in);  fclose(out);
  return 0;
}
```

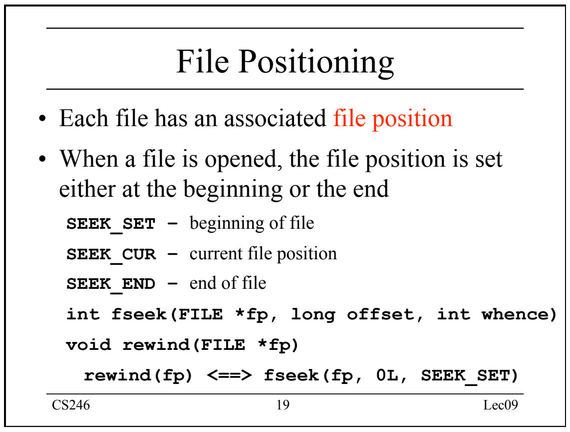
filecopy2.c

Formated I/O

- Reading – returns number of matches or EOF

  int fscanf(FILE *fp, "…", *variableList***);**
- Writing – returns number of chars written

  int fprintf(FILE *fp, "…", *variableList***);**
- **scanf** is equivalent to **fscanf** with **stdin**
- **printf** to **fprintf** with **stdout**

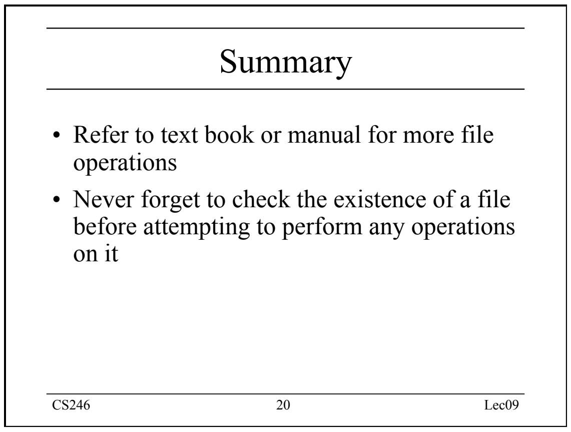


File Positioning

- Each file has an associated file position
- When a file is opened, the file position is set either at the beginning or the end

```
SEEK_SET – beginning of file
SEEK_CUR – current file position
SEEK_END – end of file
int fseek(FILE *fp, long offset, int whence)
void rewind(FILE *fp)
  rewind(fp) <==> fseek(fp, 0L, SEEK_SET)
```

CS246 19 Lec09



Summary

- Refer to text book or manual for more file operations
- Never forget to check the existence of a file before attempting to perform any operations on it

CS246 20 Lec09