

Intro to Programming II

Files

Chris Brooks

Department of Computer Science
University of San Francisco

Department of Computer Science — University of San Francisco — p. 1/77

23-2: Working with files

- In C, you work with files by accessing a file pointer.
- This is declared like this:

```
#include <stdio.h>
...
FILE *fptr;
(notice the caps)
```

Department of Computer Science — University of San Francisco — p. 2/77

23-3: Opening files for reading

- fopen() opens a file and returns a file pointer.
- It takes two arguments:
 - The file name
 - A string indicating whether we're opening for reading or writing.

Department of Computer Science — University of San Francisco — p. 3/77

23-4: Opening files for reading

- For example

```
FILE *fp = fopen("myfile", "r");
opens myfile for reading.
```

Department of Computer Science — University of San Francisco — p. 4/77

23-5: Reading from a file

- fscanf() is used to read from a file.
- Works exactly like scanf, except that the first argument is the file pointer.
- You can also use getc() - it returns the integer representing the next character in the file.
- getc() returns EOF if you're at the end of the file.

Department of Computer Science — University of San Francisco — p. 5/77

23-6: Example: cat

- Let's say we wanted to write a program to display the contents of a file to the screen.
 - In Unix, this program is called 'cat'.
- How might we go about this?

Department of Computer Science — University of San Francisco — p. 6/77

23-7: Using assignments as tests

- One shortcut you can do in C:
 - (This is not legal in Java)
- In C, when you assign a value to a variable, you can use `if` and `while` to check the result of that assignment.
- So, you often see folks do this:

```
while ((c = getc(fp)) != EOF) {
    printf("%c", c);
}
```

23-8: Exercise 1: file reading

- Take the 'cat' program we just talked about and modify it as follows:
 - Have it open two files and read them each character by character.
 - If the files are the same, print 'true'.
 - Otherwise, print 'false'.

23-9: Working with command line arguments

- We'd like to make our program a little more general.
 - It's pretty irritating to have to recompile every time we want to cat a different file.
- Let's provide the filename as a command line argument
- For example: `mycat file1 file2`

23-10: Working with command line arguments

- To do this, we need to specify that `main()` will receive arguments.
 - Remember, `main` is just another function.
- It takes its arguments in a special form:

```
int
main(int argc, char **argv)
```
- `argc` is the number of command line arguments
- `argv` is an array of strings, one for each argument.
- `argv[0]` is the name of the program.

23-11: Exercise 2

- Modify `cat` to:
 - Take two file names as arguments.
 - Check to make sure `argc >= 3`.

23-12: Writing to files

- To open a file for writing, use the "w" argument to `fopen`.

```
fopen("foo", "w");
```
- You can then write to a file with:
 - `fprintf(FILE *fp, char *stringToPrint, arg1, arg2 ...)`
 - `putc(FILE *fp, int c)`

23-13: Exercise 3

- Modify cat to read in the file indicated by the first argument and write it out to the file indicated by the second argument.

23-14: Converting strings to ints

- What if you need to convert a string to an integer?
 - Say you want to provide an integer as a command line argument?
- atoi(char *s) will return an integer representation of the string s.

23-15: Exercise 4

- Modify the multiplication program from Wednesday to take three command line arguments:
 - The number of rows
 - The number of columns
 - A file to write the table to.

23-16: Project 5

- You should now have the necessary tools to do the following parts of project 5:
 - createRandomBoard()
 - readBoardFromFile()
 - printBoard()
 - getAliveNeighbors()
 - isValidCell()
 - updateBoard()
 - main
- I'd suggest starting with createRandom board, readBoardFromFile, and printBoard. Then start on your main, so that you can test each of these.