Lecture 11

CIS 208

Friday February 18th, 2005

Wednesday - Review

- Email me questions and topics
- No new material is scheduled.

Dynamic 2-d arrays

- Must first allocate array of pointers
- Then allocate each sub-array.
- When freeing, go in reverse direction, free sub-arrays, then main array.

File I/O

- Combines Pointers and console I/O
- Most of the same basic functions.
- Works on text files or binary files

FILE pointers

- point to a stream
- FILE *fp;
- stdin, stdout are FILE pointers
- point to certain place in file.

fopen

- Must first open a file

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

- mode is how file is to be opened
- returns NULL if unsuccessful

mode values

r : Open a text file for reading

w : Create a text file for writing

a : Append to a text file

r+ : Open a text file for read/write

w+ : Create a text file for read/write

etc…

Check your book

fopen

```
FILE *fp;
fp = fopen("test.dat","w");
```

- Now we can use the stream pointed to by fp
- "w" will create a new file, if one already exists, it overwrites it.

safe fopen

- should check to see if open was sucessful.
- File might not exist, or protected

```
if ((fp = fopen("test","w")) == NULL) {
   printf("Cannot open file.\n");
   exit(1); }
```

more fopen

- Some systems have limit on number of streams
- Usually at least 8. Platform dependent
- FOPEN_MAX

fclose

- Any open file must be closed
- int fclose(FILE *fp) readies file for closure
  - Finishes any leftover writing
  - Tell OS to formally close file.
- unclosed files cause problems
  - closing == ‘saving’

int fclose(FILE *fp)

```
FILE *fp = fopen("test","w");
...
fclose(fp);
```

- fp is now closed and this pointer can be used on something else.

int fclose(FILE *fp)

- returns 0 if succesful,
- EOF if error

- Errors are rare
  - trying to close an unopened file
  - closing file on removed storage device.

character writing

- Just like console I/O

```
int putc(int ch, FILE *fp);
int fputc(int ch, FILE *fp);
```

- ch is a char, even though listed as int.
  - Only lower byte is sent.

int putc(int ch, FILE *fp)

- prints one character to file stream fp
- if successful, returns the character
- Else, returns EOF

Reading Characters

- Just like reading from stdin

int getc(FILE *fp)

int fgetc(FILE *fp)

Both do the same thing.

Returns 1 character from stream;

Returns EOF if error.

fgetc()

```
char ch;
ch = fgetc(fp);

do {
   ch = getc(fp);
} while (ch != EOF);
```

KTOD DTOS

- Examples on web page

feof()

- in binary files, EOF isn't a number
- int feof(FILE *fp)
  - given a file pointer, true if it's reached EOF
  - false if not
  - works on text files too.

feof()

```
while (!feof(fp)) {
ch = getc(fp);
}
```

FILE and strings

- Already used these before

int fputs(char *str,FILE *fp)

char *fgets(char *str,int length, FILE *fp)

fgets

- reads *length* characters
- terminated by newline
- writes to pre-allocated character array

- must replace newline character with null terminator.

fputs

- write entire string to file

```
char *str = “abcd”;
fputs(str,fp);
```

Errors

- trying to write to a “r” file
- doesn’t destroy file, but isn’t good.
- be careful