601.220 Intermediate Programming

Binary file I/O and dynamic allocation of multi-dimensional arrays

Outline

- Advanced (i.e., binary) file I/O

Advanced file I/O

- Until now, we've only accessed text files from within our C programs
  - These files use ANSI standard mapping; one byte stores one char
- But there are other types of files (that emacs/vim/less don't know how to read)

Binary files

- We call anything that isn't a text file a "binary" file
  - We won't use the ANSI mapping on these files; just give the programmer the bits and allow programmer to interpret them however they wish
- For some types of data, storing as binary can be much more efficient than as text
  - For numbers, ANSI uses one byte per decimal digit. Instead of storing 0-255, one byte is used to store only 0-9
  - Large data files such as images, audio, and video files are typically stored in binary format

Reading and writing to binary files

- To tell C to open a file as a binary file (not necessary on most Unix systems, but good practice anyway), add "b" to the open mode
  - `FILE *fp = fopen("data.dat", "rb");` opens the file in binary read mode

Reading and writing to binary files

- Instead of using only `fscanf/fgets/fprintf`, we can use `fread/fwrite` commands for binary files
  - Work for arrays, structs, arrays of structs
  - Particularly useful for reading/writing large amounts of data in one operation
  - Literally copy bits from disk to memory (fread), or memory to disk (fwrite)
  - Binary files are less portable than text, due to some types being different sizes on some architectures, for example

Reading and writing to binary files

- How do `fread` and `fwrite` work?
  - These functions take a pointer to a block of memory, an element size, a number of elements, and a filehandle
- fread then reads `size_of_el * num_els` bytes of memory from the file beginning at the file cursor location `fp`, and stores them starting at pointer location `where_to`
  - fread returns the number of items successfully written (should be same as num_els if all goes well)
  - `int items_read = fwrite(where_to, size_of_el, num_els, fp);` fread
- fwrite does the opposite, copying data from memory to the specified file
  - `int items_written = fwrite(where_from, size_of_el, num_els, fp);`

Example

```c
// bin_io.c:

#include <stdio.h>

int main()
{
   const int SIZE = 100;
   int arr_write[SIZE];
   for (int i = 0; i < 100; i++) {
      arr_write[i] = i * 10;
   }
   FILE *fp = fopen("data.dat", "wb");
   if (!fp) {
      printf("Error opening data.dat\n");
      return 1;
   }
   // writes an array of integers
   fwrite(arr_write, sizeof(arr_write[0]), SIZE, fp);
   fclose(fp);

   int arr_read[SIZE];
   fp = fopen("data.dat","rb");
   if (!fp) {
      printf("Error opening data.dat\n");
      return 1;
   }
   // reads an array of integers
   int num_of_ints = fread(arr_read,
       sizeof(arr_read[0]), SIZE, fp);
   if(num_of_ints != SIZE) {
      printf("problem reading data.dat\n");
      return 1;
   }
   if (feof(fp)) {
      printf("error: unexpected eof\n");
      return 1;
   }
   if (ferror(fp)) {
      printf("error reading data.dat\n");
   }
   for (int i = 0; i < 100; i++) {
      printf("arr_read[%d] = %d\n", i, arr_read[i]);
   }
   fclose(fp);
}
```

```
$ gcc -std=c99 -Wall -Wextra -pedantic bin_io.c
$ ./a.out
arr_read[0] = 0
arr_read[1] = 10
arr_read[2] = 20
arr_read[3] = 30
```