CS 102
Introduction to Programming Using C++

Chapter 8

Files and Streams

Homework

- Written homework
- R8.3, 6, 7, 8, 9, 11 to 13

- Programs
- p. 379. Choose one of P8.1, 2, and 4.
- Also, choose one of P8.14, 15, and 16.

Sequential Access Files

- The text files we accessed before are sequential access files
- You have to access the records in order, without skipping any
- For example, if you wish to read the fourth line, you have to
  - Read the first line
  - Read the second line
  - Read the third line
  - Then you can read the fourth line
- There is no equivalent to the array/pointer duality for a file

More on Sequential Access Files

- A text file is an example of a sequential access file
- Sequential access files are easy to create
- They are slow to access, unless you want to access every record in order
- It is painful to change a sequential access file
  - To do that, you have to recreate the file

Random Access Files with Uniform Data Records

- An alternative to a sequential access file is a random access file
- In a random access file, you can access any record at any time
  - You don't have to go in order
  - This is what the word random means
- A drawback is that every record must be the same length

Getting Ready to Use a Random Access File

- To declare a random access file variable, use

  fstream random_file;

  random_file.open (filename, status);

  - status should be one of ios::in, ios::out, or ios::binary
  - It could be more than one, separated by |

- You will need this #include

  #include <fstream>

Reading Records from a Random Access File

- You can use read

  file_var.read (&variable, number_of_bytes);

  - variable is a reference since it will be changed by the read
  - number_of_bytes is the number of bytes to read
- Usually, reading and writing is done with a struct

The sizeof Function

- You can use sizeof () to get the number of bytes in a data item
- Suppose, you have these lines in your program

```
struct Data
{
   int key;
   double value;
};
Data x;
int y;
```

- sizeof (x) will tell how many bytes x (or the struct) uses
- sieof (y) will tell how many bytes y (or any int) uses

An Example

```
struct Data
   {
      int  key;
      double value;
   };

int main ()
{
   Data x;
   x.key =  1;       x.value = 100;

   fstream myFile ("data.bin", ios::out);
   myFile.write ((char*)&x, sizeof (x));
   myFile.close();

   fstream myFile2 ("data.bin", ios::in);
   myFile2.read ((char*)&x, sizeof (x));
   cout << "x: " << x.key << "," << x.value << endl;
```

Finding Records in a Random Access File

- When using a random access file, everything is measured in bytes
  - Let's suppose a record is 15 bytes long
  - This means the first record is at Byte 0
  - The second record is at Byte 15
  - The 53rd record is at Byte 52*15
- To get ready to read the nth record, you need to move to the correct byte

```
random_file.seekg ((n-1)*15);
```

An Easy Error

- There is a “longer” integer
- It has more bytes than a regular integer
- You should use it for seekg
- A good way to do this is to declare the record size as long instead of int

- Also, seekg is usually called like this

```
const rec_size = sizeof (Data);
random_file.seekg ((rec_num-1)* rec_size);
```

Binary Files

- Random access files are most often a kind of binary file
- Another kind of binary file is a file where the early bytes in the file tell the record size and perhaps the record offset
  - An example of this is a .bmp file
  - A .bmp file is essentially a screen image
    - That means the file is a “picture” of data on the screen

Random Access Files-Type 2

- Let's examine the .bmp file structure
- There is a "header" record
  - It occupies the beginning bytes
  - It describes the file
  - It tells where the picture data can be found and how long and wide the picture is

Working with a .bmp File

- The header record tells what the file looks like
  - Bytes 2-5: The size of this file in bytes
  - Bytes 10-13: The start of the picture data
  - Bytes 18-21: The width of the picture in pixels
  - Bytes 22-25: The height of the picture in pixels
- There is more information in the header, but we will use only these bytes
- Also, you could set up a struct to hold this information

A Sample .bmp Program

- The program imagemod.cpp on p. 375 creates a "negative" of a bitmap file
- Let's examine the program
- The original data file is Russian letters.bmp
- The negative of the image the original file is in the file Russian letters (negative).bmp

Questions

- Are there any questions?