Chapter 12: Advanced File Operations

Kafi Rahman
Assistant Professor
Truman State University

12.7

Binary Files

Binary Files

- Binary file stores and retrieves everything as characters (bytes)
  - it does not follow ASCII standard
  - hence, we can't understand it if we open a binary file by using any text-editor

- The syntax to open a binary file in read mode is the following:
  - `fin.open("nums.dat", ios::in | ios::binary);`

Binary Files

- We use read and write instead of <<, >> when working with binary files

  - char ch;
  - fin.read(&ch, sizeof(char));
    - read function expects to read chars
    - &ch refers to the address of where to put
    - sizeof(ch) refers to how many bytes to read from the file

  - Similarly, write operation is the following:
    - fout.write(&ch, sizeof(char));

Binary Files: Non char data

- everything must be written/read as bytes in binary files
  - a byte is represented as char in C++
  - we must supply the address of the char when using fread or fwrite functions

- in case we want to use non-char data (int, string, double, structure, etc)
  - we have to convert the address of the non-char data to char * before writing/reading it to/from a binary file

Binary Files:
Non char data

- we need to use reinterpret_cast that can convert the address of any other data type to char*

- the syntax is shown in the following:

```
data_type *var_name = reinterpret_cast
                      <data_type *>(pointer_variable);
```

Binary Files: Non char data (cont)

- int num variable requires 4 bytes to store in the memory
- first convert the address of num to char *, so that it can be written to a binary file.
  - 4 bytes of data will be written in the file starting from the first byte

```
int num = 8;

// send a binary value to a file
outfile.write(reinterpret_cast<char *>&num,
sizeof(int));

// read in a binary number from a file
infile.read(reinterpret_cast<char *>&num,
sizeof(int));
```

num = 8

1st byte	'00000000'
2nd byte	'00000000'
3rd byte	'00000000'
4th byte	'00001000'

12.8

Creating Records with Structures

Creating Records with Structures

- By using binary file,
  - we can write a structure variable to a binary file
  - we can read a structure variable from a binary file

- To work with structures and files,
  - use ios::binary file flag upon open
  - use fread, fwrite member functions

Creating Records with Structures

- Let us review an example that illustrate how to read/write an array of objects in a binary file.
- The example is presented in binaryfile.cpp

12.9

Random-Access Files

Random-Access Files

- Sequential access: start at beginning of file and go through data in file, in order, to end
  - to access 100th entry in file, go through 99 preceding entries first
- Random access: access data in a file in any order
  - can access 100th entry directly

Random Access Member Functions

- seekg (seek get): used with files open for input
- seekp (seek put): used with files open for output
- Used to go to a specific position in a file

Random Access Member Functions

- seekg, seekp arguments:
  - offset: number of bytes, as a long
  - starting point to compute offset
  - ios::beg
    - from the start position
  - ios::cur
    - from the current position
  - ios::end
    - from the end position
- Examples:
  - inData.seekg(25L, ios::beg);
    - // set read position at 26th char
    - // from beginning of file
  - outData.seekp(-10L, ios::cur);
    - // set write position 10 bytes
    - // before the current position

Important Note on Random Access

- If eof is true, it must be cleared before using seekg or seekp functions:
    - gradeFile.clear();
    - gradeFile.seekg(0L, ios::beg);
    - // go to the beginning of the file

Random Access Information

- tellg member function: return current byte position in the input file
  - long int whereAmI;
  - whereAmI = inData.tellg();
- tellp member function: return current byte position in the output file
  - whereAmI = outData.tellp();

12.10

Opening a File for
Both Input and Output

Opening a File for Both Input and Output

- File can be opened for input and output purposes simultaneously
- Supports updating a file:
  - read data from file into memory
  - update data
  - write data back to file

- Use fstream for file object definition:
  - fstream gradeList("grades.dat", ios::in | ios::out);
  - Can also use ios::binary flag for binary data